

CRYPTOGRAPHY IN CYBER SECURITY AND DATA PRIVACY

Project Report submitted to

ST.MARY'S COLLEGE (AUTONOMOUS), THOOTHUKUDI

Affiliated to

MANONMANIAM SUNDARANAR UNIVERSITY,

TIRUNELVELI

In partial fulfillment of the requirement for the award of degree of

Bachelor of Science in Mathematics

Submitted by

NAME

REG.NO.

MAHESWARI.B

19AUMT20

MUKILA.R

19AUMT29

RASHIBA.J

19AUMT35

SAHAYA PERINBA PRAKASI.K

19AUMT37

SNEHA.T

19AUMT44

Under the Guidance of

Dr. Sr. S. KULANDAI THERESE M.Sc., B.Ed., M.Phil., Ph.D.

Assistant Professor of Mathematics

St. Mary's College (Autonomous), Thoothukudi.



Department of Mathematics

St. Mary's College (Autonomous), Thoothukudi

(2021 - 2022)

CERTIFICATE

We hereby declare that the project report entitled **“CRYPTOGRAPHY IN CYBER SECURITY AND DATA PRIVACY”** being submitted to **St. Mary's College (Autonomous), Thoothukudi** affiliated to **Manonmaniam Sundaranar University, Tirunelveli** in partial fulfillment for the award of degree of **Bachelor of Science in Mathematics** and it is a record of work done during the year **2021 - 2022** by the following students :

NAME

REG.NO.

MAHESWARI.B

19AUMT20

MUKILA.R

19AUMT29

RASHIBA.J

19AUMT35

SAHAYA PERINBA PRAKASI.K

19AUMT37

SNEHA.T

19AUMT44



Signature of the Guide

Dr. S. KULANDAI THERESE

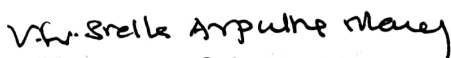
M.Sc., B.Ed., M.Phil., Ph.D.,

Assistant Professor,

Department of Mathematics,

St. Mary's College (Autonomous),

Thoothukudi - 628 001.



Signature of the HOD

Dr. V. L. Sreena Arputha Mary

M.Sc., M.Phil., B.Ed., Ph.D.,

Head & Asst Professor of Mathematics

St. Mary's College (Autonomous)

Thoothukudi-628 001.



Signature of the Examiner



Signature of the Principal

Principal

St. Mary's College (Autonomous)

Thoothukudi-628 001.

DECLARATION

We hereby declare that the project reported herewith, entitled “ **CRYPTOGRAPHY IN CYBER SECURITY AND DATA PRIVACY**”, is true to the best of our knowledge. It has not been submitted to any university for any degree or diploma.

B. Maheswari
(MAHESWARI.B)

R. Mukila
(MUKILA.R)

J. Rashiba
(RASHIBA.J)

K. Sahaya Perinba Prakasi
(SAHAYA PERINBA PRAKASI.K)

T. Sneha
(SNEHA.T)

ACKNOWLEDGEMENT

First and foremost, we thank God Almighty for showering his blessings upon us, to undergo this project successfully.

With immense pleasure, we register our deep sense of gratitude to our guide **Dr. Sr. S. Kulandai Therese M.Sc., B.Ed., M.Phil., Ph.D.** and the Head of the Department, **Dr. V. L. Stella Arputha Mary M.Sc., M.Phil., B.Ed., Ph.D.** for having imparted necessary guidelines throughout the period of our studies.

We thank our beloved Principal, **Rev. Dr. Sr. A.S.J. Lucia Rose M.Sc., M.Phil., Ph.D., PGDCA** for providing us the help to carry out our project effectively.

Last but not the least, we thank all those who extended their helping hands, to accomplish this project.

CRYPTOGRAPHY IN CYBER SECURITY AND DATA PRIVACY

PREFACE

The topic of our Project “CRYPTOGRAPHY” is the art of concealing information to induce secrecy in the communication and transmission of sensitive data is termed cryptography. Diving deep into the etymology of the word ‘cryptography’ shows that this word finds its origin in ancient Greek. Derived from words *kryptos* meaning “hidden” or “secret” and *graphy* meaning “writing”, cryptography literally means writing something secretly.

The idea of cryptography is to convey a private message or piece of information from the sender party to the intended recipient without getting the message intruded on by a malicious or untrusted party. In the world of cryptography, this suspicious third party that is trying to sneak into a private communication to extract something sensitive out of it is called an *adversary*. Cryptography protects us from these unwanted adversaries by offering a range of algorithms required to hide or protect our message in the best way possible and transmit it comfortably over a not-so-secure network.

Chapter 1 presents briefly the idea of What is Cryptography, history of cryptography and its types.

Chapter 2 deals with one of the types of cryptography ‘Hashing’ and explains briefly the types of Hashing.

Chapter 3 focuses on the most important applications of cryptography that is Encryption and Decryption and application of matrices to cryptography.

Chapter 4 deals with the algorithms used in cryptography such as Triple DES, twofish, AES, SHA256, how do block chains work, Visual cryptography and elliptic curve cryptosystems.

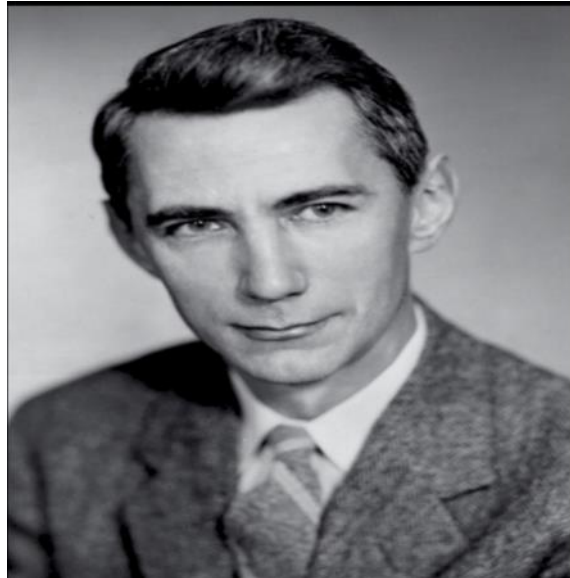
Chapter 5 deals with the ‘Magic of math in Cryptography’.

CONTENT

Introduction	9
1 Cryptography	
1.1 Definition	10
1.2 History of cryptography	10
1.3 Cryptography in everyday life	10
1.4 Types of cryptography	11
2 Hash functions	
2.1 Basics of hash functions	15
2.2 Hashing	19
3 Encryption and Decryption	
3.1 Encryption	24
3.2 Decryption	36
3.3 Application of matrices to cryptography	36
4 Algorithms used in cryptography	
4.1 Triple DES	40
4.2 Twofish	40
4.3 AES	41
4.4 SHA-256	41
4.5 Blockchain	42
4.6 Visual cryptography	43
4.7 Elliptic curve cryptosystem	44
5 Magic of math in Cryptography	
5.1 Clock math	49

5.2	Generating random numbers deterministically	50
5.3	Dividing in clock math	51
5.4	Manipulating secret contents	52
5.5	Universal functions	53
5.6	Applications of FHE	53
	Applications of cryptography in cyber security	54
	Applications of cryptography in data privacy	55
	Conclusion	56
	References	57

INTRODUCTION



Claude E . Shannon is considered by many to be the father of mathematical cryptography . Shannon worked for several years at Bell Labs and during his time there , he produced an article entitled “ A mathematical theory of cryptography ”. The first recorded use of cryptography for correspondence was by the Spartans, who as early as 400 bc employed a cipher device called the scytale for secret communication between military commanders.

CHAPTER 1

CRYPTOGRAPHY

1.1 DEFINITION

Cryptography is the study of secure communications techniques that allow only the sender and intended recipient of a message to view its contents. The term is derived from the Greek word *kryptos*, which means hidden.

1.2 HISTORY OF CRYPTOGRAPHY

As civilizations evolved, human beings got organized in tribes, groups and kingdoms. This led to the emergence of ideas such as power, battles, supremacy and politics. These ideas further fueled the natural need of people to communicate secretly with selective recipient which in turn ensure the continuous evolution of cryptography as well. The roots of cryptography are found in Roman and Egyptian civilizations.

1.2.1 HIEROGLYPH

The first known evidence of cryptography can be traced to the use of ‘Hieroglyph’. Some 4000 years ago, the Egyptians used to communicate by messages written in hieroglyph.



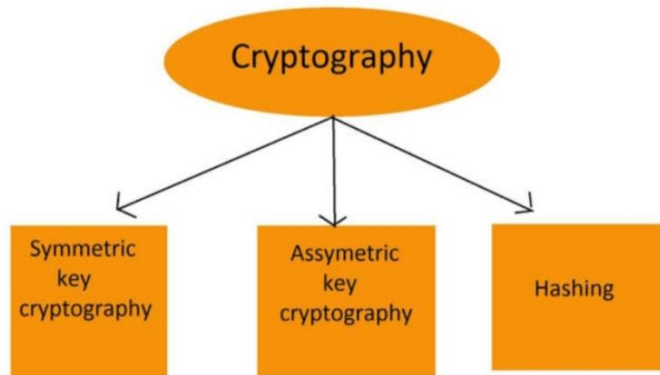
1.3 CRYPTOGRAPHY IN EVERYDAY LIFE

Today, cryptography is used to protect digital data. It is a division of computer science that focuses on transforming data into formats that cannot be recognized by unauthorized users. An example of basic cryptography is a encrypted message in which letters are replaced with other characters.

‘ **Cryptography in everyday life** ’ contains a range of situations where the use of cryptography facilitates the provision of a secure service : cash withdrawal from an ATM, Pay TV, email and file storage using Pretty Good Privacy (PGP) freeware, secure web browsing and use of a GSM mobile phone.

1.4 TYPES OF CRYPTOGRAPHY

Cryptography can be broken down into three different types:

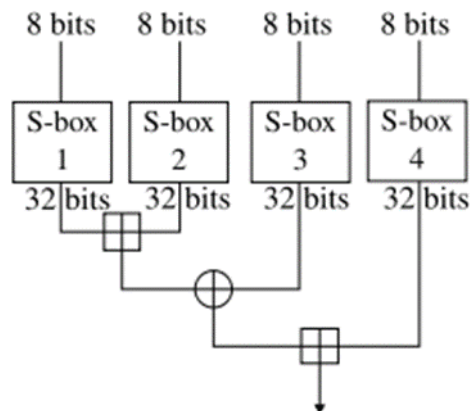


1.4.1 SYMMETRIC-KEY CRYPTOGRAPHY(Secret key cryptography)

Both the sender and receiver share a single key. The sender uses this key to encrypt plaintext and send the cipher text to the receiver. On the other side the receiver applies the same key to decrypt the message and recover the plain text.

EXAMPLE OF SYMMETRIC KEY CRYPTOGRAPHY

Blowfish



Blowfish is a symmetric block cipher that can be used as a drop-in replacement for DES or IDEA. It takes a variable-length key, from 32 bits to 448 bits, making it ideal for both domestic and exportable use. Blowfish is an alternative to DES Encryption Technique.

Features

- Block cipher: 64-bit block
- Variable key length: 32 bits to 448 bits
- Much faster than DES and IDEA
- Unpatented and royalty-free
- No license required

Working

The above diagram shows Blowfish's F-function. The function splits the 32-bit input into four eight-bit quarters, and uses the quarters as input to the S-boxes. The outputs are added modulo 2³² and XORed to produce the final 32-bit output.

1.4.2 ASYMMETRIC-KEY CRYPTOGRAPHY(Public key cryptography)

Public key cryptography is a very advanced form of cryptography. This is the most revolutionary concept in the last 300-400 years. In Public-Key Cryptography two related keys (public and private key) are used. Public key may be freely distributed, while its paired private key, remains a secret. The public key is used for encryption and for decryption private key is used.

EXAMPLE OF ASYMMETRIC KEY CRYPTOGRAPHY

RSA algorithm

It is asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. **Public Key** and **Private Key**. As the name describes that the Public Key is given to everyone and Private key is kept private.

RSA Component Features

- Public/private key generation.
- Encrypt with either public or private key.
- Decrypt with matching public or private key.

- Create digital signatures.
- Verify digital signatures.
- Encrypt and decrypt in-memory strings or byte arrays of any size.
- Encode encrypted output to Base64, Hex, Quoted-Printable, or URL-encoding
- Export public/private key pairs to XML.
- Import key pair from .snk file.
- Import public/private key pairs from XML.
- Import/Export only public-part or private-part of key pair.
- PKCS v1.5 padding for encryption and signatures.
- OAEP Padding Scheme for Encryption/Decryption
- Create/verify signatures with little-endian or big-endian byte ordering.
- Supports key sizes ranging from 512 bits to 4096 bits.
- Supports hash algorithms: MD5, SHA-1, SHA-2 (SHA-256, SHA-384, SHA-512), and
more...
- Thread safe.

1.4.3 HASH FUNCTIONS

No key is used in this algorithm. A fixed-length hash value is computed as per the plain text that makes it impossible for the contents of the plain text to be recovered. Hash functions are also used by many operating systems to encrypt passwords.

Examples of Hash Function

SHA

The **Secure Hash Algorithm (SHA)** hash functions are a set of cryptographic hash functions designed by the National Security Agency (NSA) and published by the NIST as a U.S. Federal Information Processing Standard .

- SHA stands for Secure Hash Algorithm.
- Because of the successful attacks on MD5, SHA – 0 and theoretical attacks on SHA – 1, NIST perceived a need for an alternative, dissimilar cryptographic hash, which became SHA – 3.
- In October 2012, the National Institute of Standards and Technology (NIST) chose the Keccak algorithm as the new SHA- 3 standard.

This type is explained briefly in the following chapter.

CHAPTER 2

HASH FUNCTIONS

2.1 BASICS OF HASH FUNCTIONS

Each key is associated with the values it is mapped to some numbers in the range of 0 to table size -1.

A Hash function is a key to address transformation

$$\text{Key \% Table size}$$

Example

Table size =10

Key = 75

Hash key = Key % Table size

$$= 75 \% 10$$

$$= 5$$

If input keys are random integers then this function is very simple and distribute the keys. If the table size is 10 and all the keys end in zero, then this hash function is a wrong choice.

2.1.1 TYPES OF HASH FUNCTIONS

1. Division method
2. Mid square method
3. Folding method
4. Multiplication method

1. Division Method

This is the most simple and easiest method to generate a hash value. The hash function divides the value k by M and then uses the remainder obtained.

Formula

$$h(K) = k \bmod M$$

Here,

K is the key value, and

M is the size of the hash table.

It is best suited that **M** is a prime numbers as that can make sure the keys are more uniformly distributed. The hash function is dependent upon the remainder of a division.

Example

$$k = 12345$$

$$M = 95$$

$$\begin{aligned} h(12345) &= 12345 \bmod 95 \\ &= 90 \end{aligned}$$

Pros

1. This method is quite good for any value of M .
2. The division method is very fast since it requires only a single division operation.

Cons

1. This method leads to poor performance since consecutive keys map to consecutive hash values in the hash table.
2. Sometimes extra care should be taken to choose value of M .

2. Mid square method

The mid square method is a very good hashing method. It involves two steps to compute the hash value.

1. Square the value of the key k i.e. k^2
2. Extract the middle r digits as the hash value.

Formula

$$h(K) = h(k \times k)$$

here,

k is the key value.

The value of r can be decided based on the size of the table.

Example

Suppose the hash table has 100 memory locations. So $r = 2$ because two digits are required to map the key to the memory location.

$$k = 60$$

$$k \times k = 60 \times 60$$

$$= 3600$$

$$h(60) = 60$$

The hash value obtained is 60.

Pros

1. The performance of this method is good as most or all digits of the result. This is because all digits in the key contribute to generating the middle digits of the squared result.
2. The result is not dominated by the distribution of the top digit or bottom digit of the original key value.

Cons

1. The size of the key is one of the limitations of this method, as the key is of big size then its square will double the number of digits.
2. Another disadvantage is that there will be collisions but we can try to reduce collisions.

3. Digit folding method

This method involves two steps

1. Divide the key value **k** into a number of parts i.e. **k1, k2, k3, ..., kn**, where each part that can have lesser digits than the other parts.
2. Add the individual parts. The hash value is obtained by ignoring the last carry if any.

Formula

$$k = k_1, k_2, k_3, k_4, \dots, k_n$$

$$s = k_1 + k_2 + k_3 + k_4 + \dots + k_n$$

$$h(k) = s$$

here,

s is obtained by adding the parts of the key **k**

Example

$$k = 12345$$

$$k_1 = 12, k_2 = 34, k_3 = 5$$

$$s = k_1 + k_2 + k_3$$

$$= 12 + 34 + 5$$

$$= 51$$

$$h(k) = 51$$

4. Multiplication method

This method involves the following steps

1. Choose a constant value A such that $0 < A < 1$.
2. Multiply the key value with A .
3. Extract the fractional part of kA .
4. Multiply the result of the above step by the size of the hash table i.e. M .
5. The resulting hash value is obtained by taking the floor of the result obtained in step 4.

Formula

$$h(K) = \text{floor}(M (kA \bmod 1))$$

Here,

M is the size of the hash table.

K is the key value.

A is a constant value.

Example

$$k = 12345$$

$$A = 0.357840$$

$$M = 100$$

$$h(12345) = \text{floor}[100 (12345 * 0.357840 \bmod 1)]$$

$$= \text{floor}[100 (4417.5348 \bmod 1)]$$

$$= \text{floor}[100 (0.5348)]$$

$$= \text{floor}[53.48]$$

$$= 53$$

Pros

The advantages of the multiplication method is that it can work with any value of between 0 and 1, although there are some values that tend to give better results than the rest.

Cons

The multiplication method is generally suitable when the table size is the power of two, then the whole process of computing the index by the key using multiplication hashing is very fast.

2.2 HASHING

The implementation of hash table is called as hashing. It can perform insertion, deletion and find operations in a constant average time.

2.2.1 COLLISION RESOLUTION TECHNIQUES

1. Open Hashing

- a. Separate chaining

2. Closed Hashing

- a. Linear Probing
- b. Quadratic Probing
- c. Double Hashing

1. Open Hashing

a. Separate chaining

- Retrieval of an item, r , with hash address, i , is simply retrieval from the linked list at position i .
- Deletion of an item, r , with hash address, i , is simply deleting r from the linked list at position i .

Example

Load the keys **23, 13, 21, 14, 7, 8, and 15**, in this order, in a hash table of size **7** using separate chaining with the hash function:

$$h(\text{key}) = \text{key} \% 7$$

$$h(23) = 23 \% 7 = 2$$

$$h(13) = 13 \% 7 = 6$$

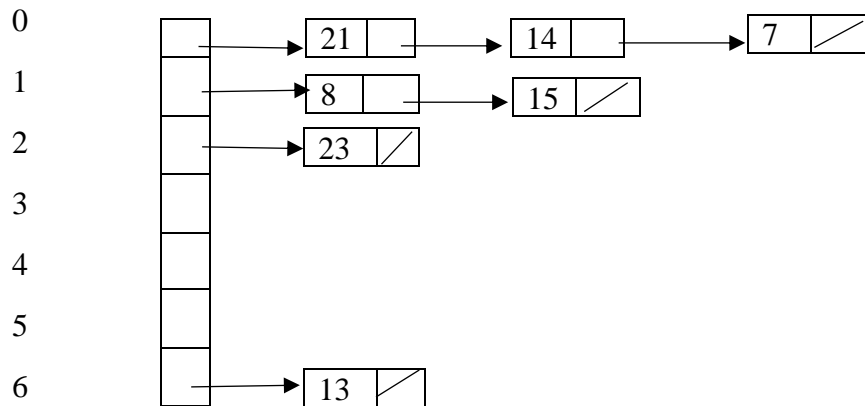
$$h(21) = 21 \% 7 = 0$$

$$h(14) = 14 \% 7 = 0 \text{ collision}$$

$$h(7) = 7 \% 7 = 0 \text{ collision}$$

$$h(8) = 8 \% 7 = 1$$

$$h(15) = 15 \% 7 = 1 \text{ collision}$$



2. Closed Hashing

a. Linear Probing

In linear probing, f is a linear function of i , typically $f(i) = i$. This means to trying cells sequentially (with wraparound) in search of an empty cell.

Example

Table size = 10

$\text{hash}(\text{key}) = \text{key} \% 10$

$f(i) = i$

Inserted keys: 89, 18, 49, 58, 69

$$h(89) = 89 \% 10 = 9$$

$$h(18) = 18 \% 10 = 8$$

$$h(49) = 49 \% 10 = 9 \text{ collision}$$

$$i=1, f(1)=1$$

$h(58) = 58 \% 10 = 8$ collision

$i=1, f(1)=1$

$i=2, f(2)=2$

$i=3, f(3)=3$

$h(69) = 69 \% 10 = 9$ collision

$i=3, f(3)=3$

	89	18	49		58	69
0			49		49	49
1					58	58
2						69
3						
4						
5						
6						
7						
8		18	18		18	18
9	89	89	89		89	89

b. Quadratic Probing

In quadratic probing, f is a quadratic function of i , typically $f(i) = i^2$.

Example

Table size = 10

$\text{hash}(\text{key}) = \text{key} \% 10$

$f(i) = i^2$

Inserted keys: 89, 18, 49, 58, 69

$h(89) = 89 \% 10 = 9$

$h(18) = 18 \% 10 = 8$

$h(49) = 49 \% 10 = 9$ collision

$i=1, f(1) = 1^2=1$

$h(58) = 58 \% 10 = 8$ collision

$$i=1, f(1)=1^2=1$$

$$i=2, f(2)=2^2=4$$

$$h(69)=69 \% 10 = 9 \text{ collision}$$

$$i=2, f(2)=2^2=4$$

	89	18	49	58	69
0			49	49	49
1					
2				58	58
3					69
4					
5					
6					
7					
8		18	18	18	18
9	89	89	89	89	89

c. Double Hashing

Hash function

$$\text{hash}(\text{key}) = \text{key} \% \text{table size}$$

When collision occurs use a second hash function

$$\text{Hash}_2(\text{key}) = R - (\text{key} \% R)$$

R is a greatest prime number smaller than table size

Example

Table size = 10

$$\text{hash}(\text{key}) = \text{key}$$

$$f(i)=i$$

$$\text{hash}_2(\text{key}) = R - (\text{key} \% R)$$

$$R = 7$$

inserted keys: 89, 18, 49, 58, 69

$$h(89)=89 \% 10 = 9$$

$$h(18) = 18 \% 10 = 8$$

$$h(49) = 49 \% 10 = 9 \text{ collision}$$

$$R - (\text{key} \% R)$$

$$7 - (49 \% 7)$$

$$7 - 0 = 7$$

$$h(58) = 58 \% 10 = 8 \text{ collision}$$

$$7 - (58 \% 7)$$

$$7 - 2 = 5$$

$$h(69) = 69 \% 10 = 9 \text{ collision}$$

$$7 - (69 \% 7)$$

$$7 - 6 = 1$$

	89	18	49	58	69
0					69
1					
2					
3				58	58
4					
5					
6			49	49	49
7					
8		18	18	18	18
9	89	89	89	89	89

CHAPTER 3

ENCRYPTION AND DECRYPTION

3.1 ENCRYPTION

Encryption is a means of securing digital data using one or more mathematical techniques, along with a password the encryption process to decrypt the information. The encryption process translates information using an algorithm that makes the original information unreadable. The process for instance can convert an original text, known as plaintext into an alternative form known as cipher text. When an authorized user needs to read the data they may decrypt the data using a binary key. This will convert cipher text back to plaintext so that the authorized user can access the original information.

3.1.1 HOW DOES ENCRYPTION WORK?

At the beginning of the encryption process, the sender must decide what cipher will best disguise the meaning of the message and what variable to use as a key to make the encoded message unique. The most widely used types of ciphers fall into two categories: Symmetric and Asymmetric ciphers.

3.1.2 SYMMETRIC CIPHERS

Symmetric ciphers, also referred to as secret key encryption, use a single key. The key is sometimes referred to as a shared secret because the sender or computing system doing the encryption must share the secret key with all entities authorized to decryption is usually much faster than asymmetric encryption. The most widely used symmetric key cipher is the Advanced Encryption Standard, which was designed to protect government-classified information.

There are basically two types of symmetric ciphers

1. Substitution cipher
2. Transposition cipher

1. SUBSTITUTION CIPHER

A substitution is a technique in which each letter or bit of the plaintext is substituted or replaced by some other letter, number, or symbol to produce cipher text.

For Example: ABC \rightarrow XYZ

Types of Substitution cipher

- a) Caesar cipher
- b) Monoalphabetic cipher
- c) Polyalphabetic cipher
- d) Playfair cipher
- e) One time pad cipher
- f) Hill cipher

a) Caesar cipher

- 1. Letters are replaced by letters or symbols.
- 2. The earliest known and simplest method used by Julius Caesar.
- 3. Replacing each letter of the alphabet with the letter standing three places further down the alphabet.

Formula

1. For each plaintext letter 'P' substitute the cipher letter C.

2. $C = E(P, K) \bmod 26 = (P + K) \bmod 26$ and $P = D(C, K) \bmod 26 = (C - K) \bmod 26$.

TABLE : 1

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

Example

Encrypt 'NESO ACADEMY' using Caesar cipher

Solution

Encryption

Plain text : NESO ACADEMY

N	E	S	O	A	C	A	D	E	M	Y
Q	H	V	R	D	F	D	G	H	P	B

$$C = (P + K) \bmod 26$$

$$= (13 + 3) \bmod 26$$

$$= 16 \bmod 26$$

$$= 16$$

$$C = Q$$

Cipher text : QHVRDFDGHPB

Decryption

Cipher text : QHVRDFDGHPB

Q	H	V	R	D	F	D	G	H	P	B
N	E	S	O	A	C	A	D	E	M	Y

$$P = (C - K) \bmod 26$$

$$= (16 - 3) \bmod 26$$

$$= 13 \bmod 26$$

$$= 13$$

$$P = N$$

Plain text : NESO ACADEMY

b) Monoalphabetic cipher

Monoalphabetic cipher substitution are letter of the alphabet with another letter of the alphabet. However rather than substituting according to a regular pattern any letter can be substituted for any other letter as long as each letter has a unique substitute left and vice versa.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
D	E	A	B	C	U	W	V	F	Y	Z	X	G	T	H	S	R	I	J	Q	P	K	L	O	N	M

Example

Encrypt the message “HELLO” using Monoalphabetic cipher

Solution:

Encryption

Plain text message : HELLO

Cipher text message: VCXXH

Decryption

Cipher text message : VCXXH

Plain text message : HELLO

c) Polyalphabetic cipher

1. Polyalphabetic cipher is any cipher based on substitution alphabets.
2. The vigenere cipher is probably the best known example of polyalphabetic cipher.
3. Vignere cipher is a method of encryption alphabetic text.

Thus technique used for both encryption and decryption the message.

Encryption formula

Converging (A – Z) into number (0-25). The plaintext (p) and key are called modulo 26.

$$E_j = (P_j + K_j) \bmod 26$$

Decryption formula

$$D_j = (E_j - K_j + 26) \bmod 26$$

Example

The plaintext is “JAVATPOINT”, and the key “BEST”

Solution

Encryption

$$E_j = (P_j + K_j) \bmod 26$$

Plain text : JAVATPOINT

KEY = BESTBESTBE

J = 9 and B = 1

$$E_1 = (P_1 + K_1) \bmod 26$$

$$= (J + B) \bmod 26 \quad (\text{using table:1})$$

$$= (9 + 1) \bmod 26 = 10 \bmod 26$$

$$E_1 = 10$$

Plaintext	J	A	V	A	T	P	O	I	N	T
Plaintext value(P)	9	0	21	0	19	15	14	8	13	19
Key	B	E	S	T	B	E	S	T	B	E
Key value(K)	1	4	18	19	1	4	18	19	1	4
Cipher text value(E)	10	4	13	19	20	19	6	1	14	23
Cipher text	K	E	N	T	U	T	G	B	O	X

Cipher text : KENTUTGBOX

Decryption

If any case(D_j) value becomes negative (-ve), in this case, we will add 26 in the negative value.

$K = 10$ and $B = 1$

$$D_j = (E_j - K_j + 26) \bmod 26$$

$$= (E_1 - K_1 + 26) \bmod 26 = (K - B + 26) \bmod 26 \quad (\text{by using Table : 1})$$

$$= (10 - 1 + 26) \bmod 26 = (35) \bmod 26$$

$$= 9$$

$$= J$$

Cipher text	K	E	N	T	U	T	G	B	O	X
Ciphertext value (E)	10	4	13	19	20	19	6	1	14	23
Key	B	E	S	T	B	E	S	T	B	E
Key value(k)	1	4	18	19	1	4	18	19	1	4
Plaintext value (P)	9	0	21	0	19	15	14	8	13	19
Plaintext	J	A	V	A	T	P	O	I	N	T

Plaintext : JAVATPOINT

d) Playfair cipher

Playfair cipher is a digraph substitution cipher. It employs a table where one letter is omitted and the letter are arranged in 5 x 5 grid.

Rules

1. Diagrams
2. Repeating letters – Filler letter

3. Same column $\downarrow\downarrow$ wrap around

4. Same row \rightarrow wrap around

5. Rectangle \leftrightarrow swap

Example

The Plaintext “ATTACK “ and the Key MONARCHY

Solution

Encryption

M	O	N	A	R
C	H	Y	B	D
E	F	G	I/J	K
L	P	Q	S	T
U	V	W	X	Z

Plain text : AT TA CK

Cipher text : RS SR DE

Cipher text : RSSRCE

DECRYPTION

M	O	N	A	R
C	H	Y	B	D
E	F	G	I/J	K
L	P	Q	S	T
U	V	W	X	Z

Cipher text : RS SR DE

Plain text : AT TA CK

Plain text : ATTACK

e) One time pad cipher

1. One time pad (OTP) also called Vernamcipher or the perfect cipher is a crypto algorithm where plaintext is combined with a random key.

2. The key is at least as long as the message or data that must be encrypted.

3. Each key is used only once and both sender and receiver must destroy their key after use.

4. There should only two copies of the key: one for **sender** and one for **receiver**.

Example

The Plain text “ACTIVE” and key “CELOAI”

Solution

ENCRYPTION (+)

Plain text → A C T I V E

key → C E L O A I

0 2 19 8 21 4 (using Table : 1)

2 4 11 14 0 8

2 6 30 22 21 12

-26

4

Cipher text → C G E W V M

DECRYPTION (-)

Cipher text → C G E W V M , key → C E L O A I

2 6 4 22 21 12 (using Table:1)

2 4 11 14 0 8

0 2 -7 8 21 4

+26

19

Plain text \rightarrow A C T I V E

f) Hill cipher

In classical cryptography the **Hill cipher** is a polygraphic substitution cipher based on **linear algebra**. Invented by **Lester S. Hill** in 1929, it was the first polygraphic cipher in which it was practical to operate on more than three symbols at once.

Encryption formula

$$C = KP \bmod 26$$

Decryption formula

$$P = K^{-1}C \bmod 26 \quad \text{where } K^{-1} = \frac{1}{|K|} \text{adj } K$$

Example

Plain text is CD. Find out Cipher text of given plain text using cipher text. Key matrix = $\begin{bmatrix} 2 & 3 \\ 3 & 4 \end{bmatrix}$

Solution

Encryption

Plain text = CD (C = 2, D = 3) (using Table : 1)

$$\text{Key matrix} = \begin{bmatrix} 2 & 3 \\ 3 & 4 \end{bmatrix}$$

$$C = KP \bmod 26$$

$$C = \begin{bmatrix} 2 & 3 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix} \bmod 26$$

$$= \begin{bmatrix} 13 \\ 18 \end{bmatrix} \bmod 26$$

$$= \begin{bmatrix} 13 \\ 18 \end{bmatrix}$$

$$= \begin{bmatrix} N \\ S \end{bmatrix}$$

Cipher text = NS

Decryption

Find out inverse matrix of given key matrix.

$$P = K^{-1}C \bmod 26$$

$$K^{-1} = \frac{1}{|K|} \text{adj}K$$

$$|K| = \begin{vmatrix} 2 & 3 \\ 3 & 4 \end{vmatrix} = 8 - 9 = -1$$

$$\text{Adj } K = \begin{bmatrix} 4 & -3 \\ -3 & 2 \end{bmatrix}$$

$$K^{-1} = \frac{1}{-1} \begin{bmatrix} 4 & -3 \\ -3 & 2 \end{bmatrix}$$

$$= \begin{bmatrix} -4 & 3 \\ 3 & -2 \end{bmatrix}$$

Cipher text = (N = 13 , S = 18) (using table : 1)

$$\text{Key inverse matrix} = \begin{bmatrix} -4 & 3 \\ 3 & -2 \end{bmatrix}$$

$$P = \begin{bmatrix} -4 & 3 \\ 3 & -2 \end{bmatrix} \begin{bmatrix} 13 \\ 18 \end{bmatrix} \bmod 26$$

$$= \begin{bmatrix} 2 \\ 3 \end{bmatrix} \bmod 26$$

$$= \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

$$= \begin{bmatrix} C \\ D \end{bmatrix}$$

Plain text = CD

2. TRANSPOSITION CIPHER

In transposition technique, there is no replacement of alphabets or numbers occurs instead their position are changed or reordering of position of plain text is done to produce cipher text.

For example ABCDE \rightarrow BADEC

Types of Transposition cipher

- a) Rail fence
- b) Row Transposition cipher

a) Rail fence

The plaintext is written down as a sequence of diagonals and then read off as a sequence of rows.

Example

Encrypt the message “PLEASE SAVE ME” with a rail fence of depth 2

Solution

Encryption

Plain text : PLEASE SAVE ME

Depth : 2

```
P E S S V M
L A E A E E
```

Cipher text : PESSVMLAEAE

Decryption

Cipher text : PESSVMLAEAE

```
P E S S V M
L A E E E
```

Plaintext : PLEASESAVEME

Plain text : PLEASE SAVE ME

b) Row Transposition cipher

We write the message in a rectangle, row by row and read the message off, column by column but permute the order of column.

Example

Encrypt the message “ATTACK POSTPONED UNTILL TWO AM”

Solution

Encryption

Plain text : ATTACK POSTPONED UNTILL TWO AM

3 2 1 6 5 4

A	T	T	A	C	K
P	O	S	T	P	O
N	E	D	U	N	T
I	L	T	W	O	A
M	V	W	X	Y	Z

Cipher text : TSDDTWTOELVAPNIMKOTAZCPNOYATUWX

Decryption

Cipher text : TSDDTWTOELVAPNIMKOTAZCPNOYATUWX

3 2 1 6 5 4

T	T	A	K	C	A
S	O	P	O	P	T
D	E	N	T	N	U
T	L	I	A	O	W
W	V	M	Z	Y	X

Plain text : ATTACKPOSTPONEDUNTILTWOAMVWXYZ

Plain text : ATTACK POSTPONED UNTIL TWO AM

3.1.3 ASYMMETRIC CIPHER

Asymmetric ciphers also known as public key, encryption use two different – but logically linked keys. This type of cryptography often uses prime numbers to create keys since

it is computationally different to factor large number and reverse engineer the encryption. The Shamir Adelman (RSA) encryption algorithm currently the most widely used the public key algorithm. With RSA, the public or the private key can be used to encrypt a message whichever key is not used for encryption becomes the decryption key.

3.2 DECRYPTION

The conversion of encrypted data its original form is called decryption. It is generally a reverse process of encryption. It decodes the encrypted information so that authorized user can only decrypt the data because decryption requires a secret key or password.

3.2.1 HOW DOES DECRYPTION WORK?

To understand how decryption typically works, let's consider the case of a veeam backup. When typing to recover information from a Veeam backup, an encrypted backup file and Replication will perform decryption automatically in the backdrop or will require a key.

In case an encryption password is required to gain access to the backup file, if the replication configuration database and Veeam backup is accessible, the key is no longer necessary. The passwords from the database are required to open the backup file. The information is accessible in the backdrop, and data recovery is not much different from that of the unencrypted data.

3.3 APPLICATION OF MATRICES TO CRYPTOGRAPHY

One of the important applications of inverse of a non-singular square matrix is in cryptography. Cryptography is an art of communication between two people by keeping the information not known to others. It is based upon two factors, namely encryption and decryption. **Encryption** means the process of transformation of an information (plain form) into an unreadable form (coded form). On the other hand, **Decryption** means the transformation of the coded message back into original form. Encryption and decryption require a secret technique which is known only to the sender and the receiver.

This secret is called a **key**. One way of generating a key is by using a non-singular matrix to encrypt a message by the sender. The receiver decodes (decrypts) the message to retrieve the original message by using the inverse of the matrix. The matrix used for encryption is called

encryption matrix (encoding matrix) and that used for decoding is called **decryption matrix (decoding matrix)**.

We explain the process of encryption and decryption by means of an example.

3.3.1 EXAMPLE

Suppose that the sender and receiver consider messages in alphabets $A - Z$ only, both assign the numbers 1-26 to the letters $A - Z$ respectively, and the number 0 to a blank space. For simplicity, the sender employs a key as post-multiplication by a non-singular matrix of order 3 of his own choice. The receiver uses post-multiplication by the inverse of the matrix which has been chosen by the sender.

Let the encoding matrix be

$$A = \begin{bmatrix} 1 & -1 & 1 \\ 2 & -1 & 0 \\ 1 & 0 & 0 \end{bmatrix}.$$

Let the message to be sent by the sender be “**WELCOME**”.

Since the key is taken as the operation of post-multiplication by a square matrix of order 3, the message is cut into pieces (**WEL**), (**COM**), (**E**), each of length 3, and converted into a sequence of row matrices of numbers:

[23 5 12] , **[3 15 13]** , **[5 0 0]**.

Note that, we have included two zeros in the last row matrix.

The reason is to get a row matrix with 5 as the first entry.

Next, we encode the message by post-multiplying each row matrix as given below:

Uncoded row matrix	Encoding matrix	Coded row matrix
$[23 \ 5 \ 12]$	$\begin{bmatrix} 1 & -1 & 1 \\ 2 & -1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$	$= [45 \ -28 \ 23];$
$[3 \ 15 \ 13]$	$\begin{bmatrix} 1 & -1 & 1 \\ 2 & -1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$	$= [46 \ -18 \ 3];$
$[5 \ 0 \ 0]$	$\begin{bmatrix} 1 & -1 & 1 \\ 2 & -1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$	$= [5 \ -5 \ 5].$

So the encoded message is $[45 \ -28 \ -23] \ [46 \ -18 \ 3] \ [5 \ -5 \ 5]$

The receiver will decode the message by the reverse key, post-multiplying by the inverse of A.

So the decoding matrix is

$$A^{-1} = \frac{1}{|A|} \text{adj } A = \begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 2 \\ 1 & -1 & 1 \end{bmatrix}.$$

The receiver decodes the coded message as follows:

Coded row matrix	Decoding matrix	Decoded row matrix
$[45 \ -28 \ 23]$	$\begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 2 \\ 1 & -1 & 1 \end{bmatrix}$	$= [23 \ 5 \ 12];$
$[46 \ -18 \ 3]$	$\begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 2 \\ 1 & -1 & 1 \end{bmatrix}$	$= [3 \ 15 \ 13];$
$[5 \ -5 \ 5]$	$\begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 2 \\ 1 & -1 & 1 \end{bmatrix}$	$= [5 \ 0 \ 0].$

So, the sequence of decoded row matrices is $[23\ 5\ 12]$, $[3\ 15\ 13]$, $[5\ 0\ 0]$.

Thus, the receiver reads the message as “**WELCOME**”.

CHAPTER 4

ALGORITHMS USED IN CRYPTOGRAPHY

4.1 TRIPLE DES

Triple DES is an encryption technique which uses three instance of DES on same plain text. It uses there different types of key choosing technique in first all used keys are different and in second two keys are same and one is different and in third all keys are same. 3DES is an improvement over des, but each has their benefits and opportunities for improvements.

4.1.1 The encryption-decryption process is as follows

- Encrypt the plaintext blocks using single DES with key K1.
- Now decrypt the output of step 1 using single DES with key K2.
- Finally, encrypt the output of step 2 using single DES with key K3.
- The output of step 3 is the ciphertext.
- Decryption of a ciphertext is a reverse process. User first decrypt using K3, then encrypt with K2, and finally decrypt with K1.

4.2 TWOFISH

Twofish is a symmetric block cipher; a single key is used for encryption and decryption. It has a block size of 128 bits, and accepts a key of any length up to 256 bits.

4.2.1 Features

- 128 bit block cipher
- Uses 16 rounds of Feistel network
- Key length of 128 bit,192 bits and 256 bits
- No weak keys

4.3 AES

The Advanced Encryption Standard (AES) is a specification for the encryption of electronic data. AES is six times faster than Triple DES. AES is much faster than RSA.

4.3.1 Features

- Symmetric key symmetric block cipher
- 128-bit data, 128/192/256-bit keys
- Stronger and faster than Triple-DES
- Provide full specification and design details
- Software implementable in C and Java

AES performs all its computations on bytes rather than bits. Hence, AES treats the 128 bits of a plaintext block as 16 bytes. These 16 bytes are arranged in four columns and four rows for processing as a matrix. The number of rounds in AES is variable and depends on the length of the key. AES uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. Each of these rounds uses a different 128-bit round key, which is calculated from the original AES key.

4.4 SHA-256

SHA-256 is a one-way function that converts a text of any length into a string of 256 bits. This is known as a hashing function. SHA256 stands for Secure Hash Algorithm 256-bit and it's used for cryptographic security.

SHA-256 generates an almost-unique 256-bit (32-byte) signature for a text. **SHA-256** is one of the successor hash functions to SHA-1. It is one of the strongest hash functions available. SHA-256 is not much more complex to code than SHA-1, and has not yet been compromised in any way. The 256-bit key makes it a good partner-function for AES.

4.5 BLOCKCHAIN

It's decentralized nature and cryptographic algorithm make it immune to attack. In fact, **hacking a Blockchain** is close to impossible. In a world where cyber security has become a key issue for personal, corporate, and national security, **Blockchain** is a potentially revolutionary technology.

4.5.1 Features

1. Cannot be Corrupted
2. Decentralized Technology
3. Enhanced Security
4. Distributed Ledgers
5. Consensus
6. Faster Settlement

One of the application of blockchain is bitcoin.

4.6 VISUAL CRYPTOGRAPHY

Visual cryptography is a cryptographic technique which allows visual information (pictures, text, etc.) to be encrypted in such a way that decryption can be done just by sight reading.

4.6.1 Features

- 1) The independence of pixel's encryption.
- 2) Easy matrix generation.
- 3) Simple operations.

Visual Cryptography is a special encryption technique to hide information in images in such a way that it can be decrypted by the human vision if the correct key image is used. Visual Cryptography uses two transparent images. One image contains random pixels and the other image contains the secret information. It is impossible to retrieve the secret information from one of the images. Both transparent images and layers are required to reveal the information.

4.6.2 Applications

- Safe websites
- Secure online transactions
- For encryption of files
- Military communications
- Encryption in WhatsApp
- Sim card Authentication
- Electronic Money

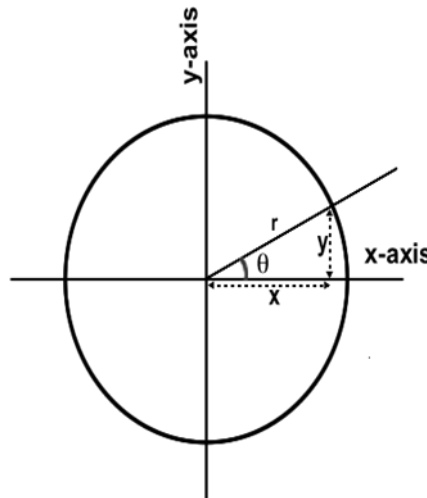
4.7 ELLIPTIC CURVE CRYPTOSYSTEM

We start by giving a short introduction to the mathematical concept of elliptic curves, independent of their cryptographic applications. ECC is based on the generalized discrete logarithm problem. Hence, what we try to do first is to find a cyclic group on which we can build our cryptosystem. Of course, the mere existence of a cyclic group is not sufficient. The DL problem in this group must also be computationally hard, which means that it must have good one-way properties.

We start by considering certain polynomials (e.g., functions with sums of exponents of x and y), and we plot them over the real numbers.

Example 1

Let's look at the polynomial equation $x^2 + y^2 = r^2$ over the real number \mathbb{R} . If we plot all the pairs (x, y) which fulfill this equation in a coordinate system, we obtain a circle as shown in the figure.

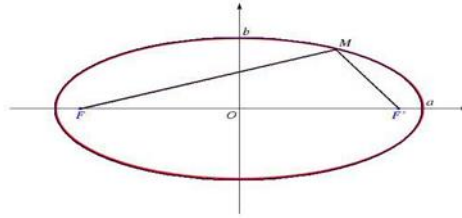


Plot of all points (x, y) which fulfill the equation $x^2 + y^2 = r^2$

We now look at other polynomial equations over the real numbers.

Example 2

A slight generalization of the circle equation is to introduce coefficients to the two terms x^2 and y^2 , i.e., we look at the set of solutions to the equation $a \cdot x^2 + b \cdot y^2 = c$ over the real numbers. It turns out that we obtain an ellipse, as shown in the figure.

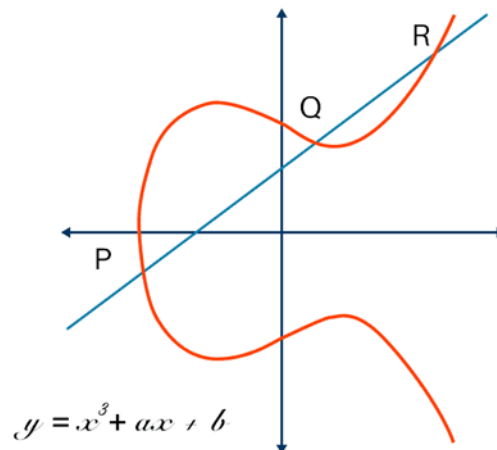


Plot of all points (x,y) which fulfil the equation $ax^2+by^2 = c$.

4.7.1 DEFINITION

From the two examples above, we conclude that we can form certain types of curves from polynomial equations. By “curves”, we mean the set of points (x,y) which are solutions of the equations.

- Elliptic curve cryptography is a key based technique for encrypting data. ECC focuses on pairs of public and private keys for decryption and encryption.
- It provides equal security with smaller key size as compared to non ECC algorithms.
- It make use of elliptic curves.
- Elliptic curves are defined by some mathematical function $f_x = y^2 = x^3 + ax + b$.



- Symmetric to x-axis.
- If we draw a line, it will touch a maximum of 3 parts.
- The definition of elliptic curve requires that the curve is nonsingular. Geometrically speaking, this means that the plot has no self-intersections or vertices.
- For cryptographic use we are interested in studying the curve over a prime field as in the definition. However, if we plot such an elliptic curve over Zp , we do not get anything remotely resembling a curve. However, nothing prevents us from taking an elliptic curve equation and plotting it over the set of real numbers.

4.7.2 GROUP OPERATIONS OF ELLIPTIC CURVE

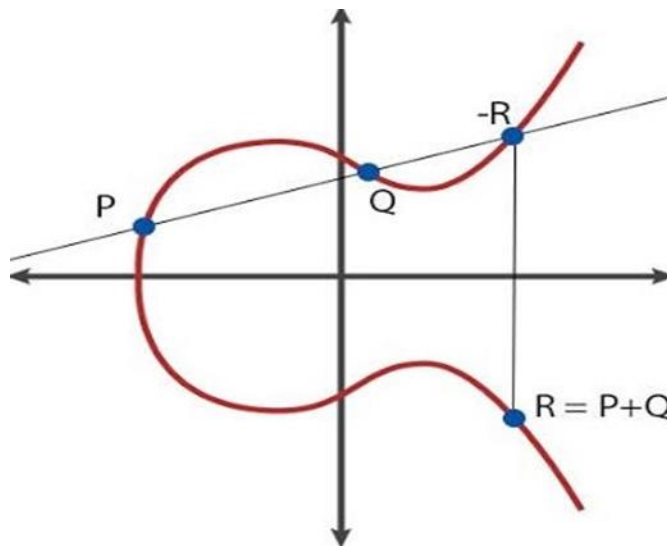
We denote the group operation with the addition symbol “+”. “Addition” means that given two points and their coordinates, say $P = (x_1, y_1)$ and $Q = (x_2, y_2)$, we have to compute the coordinates of a third point R such that:

$$\begin{aligned} P+Q &= R \\ (x_1, y_1)+(x_2, y_2) &= (x_3, y_3) \end{aligned}$$

As we will see below, it turns out that this addition operation looks quite arbitrary. Luckily, there is a nice geometric interpretation of the addition operation if we consider a curve defined over the real numbers. For this geometric interpretation, we have to distinguish two cases: the addition of two distinct points (named point addition) and the addition of one point to itself (named point doubling).

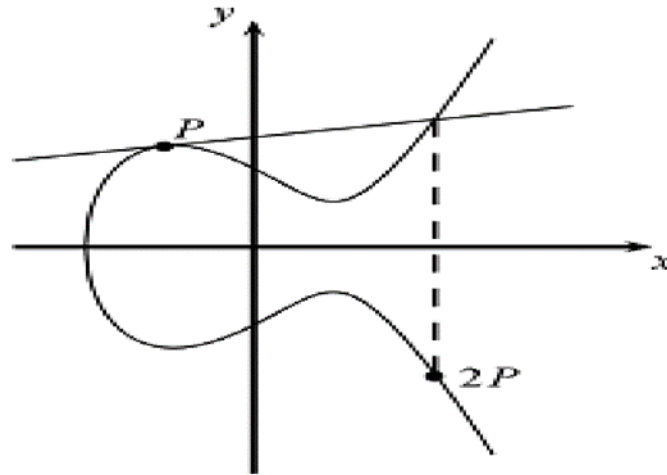
Point Addition $P+Q$

This is the case where we compute $R = P+Q$ and $P \neq Q$. The construction works as follows: Draw a line through P and Q and obtain a third point of intersection between the elliptic curve and the line. Mirror this third intersection point along the x -axis. This mirrored point is, by definition, the point R .



Point Doubling $P+P$

This is the case where we compute $P+Q$ but $P=Q$. Hence, we can write $R = P+P = 2P$. We need a slightly different construction here. We draw the tangent line through P and obtain a second point of intersection between this line and the elliptic curve. We mirror the point of the second intersection along the x -axis. This mirrored point is the result R of the doubling.



We might wonder why the group operations have such an arbitrary looking form. Historically, this *tangent-and-chord* method was used to construct a third point if two points were already known, while only using the four standard algebraic operations add, subtract, multiply and divide. It turns out that if points on the elliptic curve are *added* in this very way, the set of points also fulfill most conditions necessary for a group, that is, closure, associativity, existence of an identity element and existence of an inverse.

Of course, in a cryptosystem we cannot perform geometric constructions. However, by applying simple coordinate geometry, we can express both of the geometric constructions from above through analytic expressions, i.e., formulae. As stated above, these formulae only involve the four basic algebraic operations. These operations can be performed in any field, not only over the field of the real numbers. In particular, we can take the curve equation from above, but we now consider it over prime fields $GF(p)$ rather than over the real numbers. This yields the following analytical expressions for the group operation.

4.7.3 ELLIPTIC CURVE POINT ADDITION AND POINT DOUBLING

$$\begin{aligned}
 & x_3 = s^2 - x_1 - x_2 \bmod p \text{ and} \\
 & y_3 = s(x_1 - x_3) - y_1 \bmod p \quad \text{where} \\
 & s = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} \bmod p & ; \text{ if } P \neq Q \text{ (point addition)} \\ \frac{3x_1^2 + a}{2y_1} \bmod p & ; \text{ if } P = Q \text{ (point doubling)} \end{cases}
 \end{aligned}$$

Note that the parameter s is the slope of the line through P and Q in the case of point addition, or the slope of the tangent through P in the case of point doubling.

PROBLEM - 1

On an elliptic curve $y^2 = x^3 - 36$ find a point $P + Q$ and $2P$

Solution

Let $p = (-3, 9)$, $Q = (-2, 8)$, $a = -3$

$$X_3 = \left(\frac{8-9}{-2+3} \right)^2 - (-3) - (-2)$$

$$= 1^2 + 3 + 2$$

$$X_3 = 6$$

$$Y_3 = -9 + \left(\frac{8-9}{-2+3} \right)(-3-6)$$

$$= -9 - 1(-9)$$

$$Y_3 = 0$$

$$P + Q = (6, 0)$$

Now to find $2P$

$$X_3 = \left(\frac{3(9)+(-36)}{2 \times 9} \right)^2 - 2(-3)$$

$$= \left(\frac{27-36}{18} \right)^2 + 6$$

$$= \frac{1}{4} + 6$$

$$X_3 = \frac{25}{4}$$

$$Y_3 = -9 + \left(\frac{3 \times 9 - 36}{18} \right) \left(-3 - \frac{25}{4} \right)$$

$$Y_3 = \frac{-35}{8}$$

$$2P = \left(\frac{25}{4}, \frac{-35}{8} \right)$$

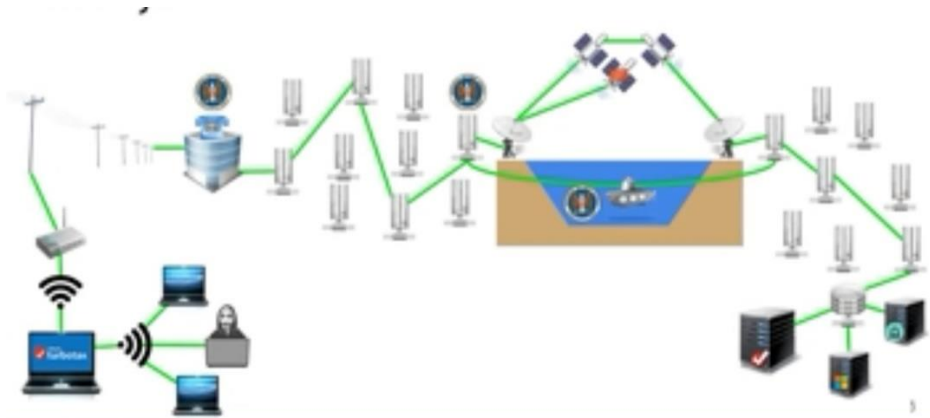
CHAPTER 5

MAGIC OF MATH IN CRYPTOGRAPHY

THE PERCEPTION



REALITY



5.1 CLOCK MATH

Cryptography is based on math that everyone understands: Clock math!

- Clocks have a finite range of numbers, which loop around rather than continuing to infinity.
- If we can understand the math of clocks, we can understand math that secures our computers and networks!

5.1.1 HOW ENCRYPTION WORKS

1. Choose message **m** to be encrypted.

2. Choose a number **r** from the clock.

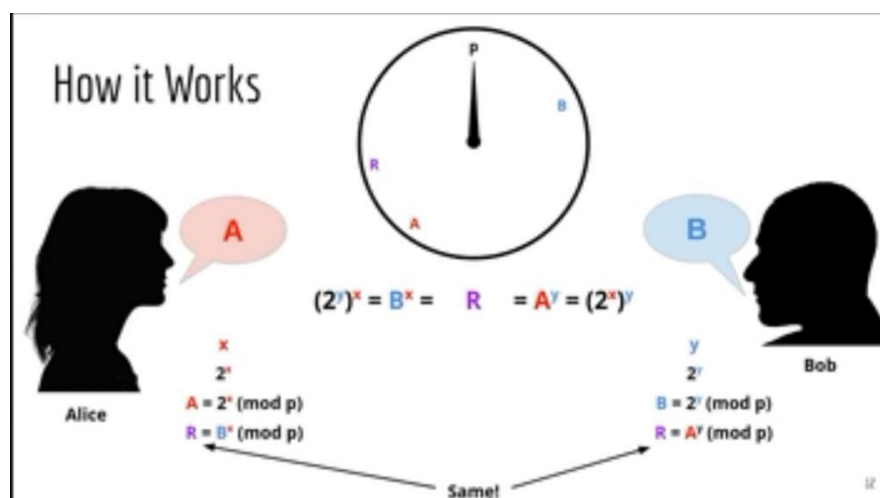
- Must be chosen at random.
- Every number must have equal chance!

3. Add **r** to **m**.

- The result, **c** called the ciphertext, is the encryption of message **m** with key **r**.



5.1.2 CHOOSING SECRETS IN PUBLIC

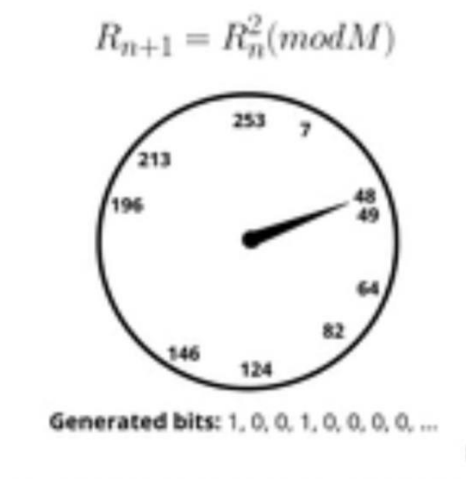


5.2 GENERATING RANDOM NUMBERS DETERMINISTICALLY

1. It sounds like an oxymoron.
 - How can a deterministic process generates random number?
2. Yet this is the foundation of almost all cryptography. Every encryption algorithm is an algorithm that can take a small key, of say 16 characters and use it to produce an endless sequence of random bits.
 - These bits can then be used to encrypt the messages of any size or multiple messages.
 - HDs full of data can be encrypted with 1 random value this long : "abcdefghijklmnop"

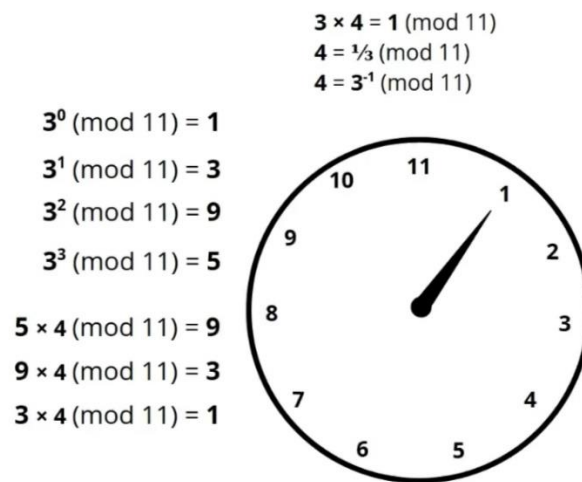
5.2.1 TURNING ONE RANDOM NUMBER INTO MANY

Seed = 7 , M = 253	even or odd?	output bit
$49 = 7^2 \mod 253$	odd	1
$124 = 49^2 \mod 253$	even	0
$196 = 124^2 \mod 253$	even	0
$213 = 196^2 \mod 253$	odd	1
$82 = 213^2 \mod 253$	even	0
$146 = 82^2 \mod 253$	even	0
$64 = 146^2 \mod 253$	even	0
$48 = 64^2 \mod 253$	even	0



5.3 DIVIDING IN CLOCK MATH

- What does it mean to “divide”?
 - Is it possible to divide on a clock?
- We can! By multiplying by an “inverse”
 - Division by N = multiplying by (1/N)
 - E.g.: $(x/3) = (x * 1/3)$
- Despite there being no fractions, there are inverses in clock math. An inverse is a number that when multiplied gives 1.
 - $3 * 1/3 = 1$
 - E.g.: $3 * 4 \pmod{11} = 1$



5.4 MANIPULATING SECRET CONTENTS

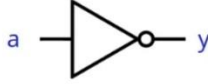

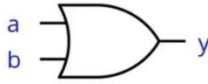
In 1978, it was discovered that an encryption algorithm called RSA let people multiply two encrypted values together without decrypting them.

$$\begin{array}{ccccc}
 \text{[Noise]} & \times & \text{[Noise]} & = & \text{[Green Circle with 6]} \\
 E(a) & \times & E(b) & = & E(a \times b)
 \end{array}$$

Then, in 1999, it was discovered that an encryption algorithm called Paillier let people add two encrypted values together without decrypting them.

$$\begin{array}{ccccc}
 \text{[Noise]} & + & \text{[Noise]} & = & \text{[Green Circle with 8]} \\
 E(a) & + & E(b) & = & E(a + b)
 \end{array}$$

5.5 UNIVERSAL FUNCTIONS

Logical Operation	Logic Gate Symbol	Algebraic Representation
NOT		$y = (1 - a)$
AND		$y = (a \cdot b)$
OR		$y = (a + b) - (a \cdot b)$

5.6 APPLICATIONS OF FULLY HOMOMORPHIC ENCRYPTION (FHE)

- Having a genetic test done without revealing our DNA
 - Encrypt each base pair, send encrypted bits to be processed
 - We get back an encrypted bit, which will decrypt to 1 if we have the disease, 0 if not
 - Prototypes of this have already been tested
- Systems that can handout encryption keys without seeing them
- Millions of others, literally everything we can think of can be done !
 - But there is a downside : efficiency it's much slower and less efficient to run a computation on encrypted data (each encrypted bit is 10000 bits inside)
 - A 750 MB genome that fits on a CD becomes 7.5 TB (would fill a large HD)

APPLICATIONS OF CRYPTOGRAPHY IN CYBER SECURITY

SECRECY IN TRANSMISSION

Some existing secrecy systems for transmission access a private key system for converting transmitted data because it is the quickest approach that functions with rational guarantee and low overhead.

If the multiple conversing parties is minute, key distribution is implemented periodically with a courier service and key preservation based on physical security of the keys over the method of use and destruction after new keys are disseminated.

SECRECY IN STORAGE

Secrecy in storage is frequently preserved by a one-key system where the user provide the key to the computer at the commencement of a session, and the system creates concern of encryption and decryption during the phase of normal use.

AUTHENTICATION OF IDENTITY

Authenticating the identity of individuals or systems to each other has been a difficulty for a very long time. Simple passwords have been used to test identity. More compound protocols such as sequence of keywords exchanged between sets of parties are generally display in the movies or on television.

CREDENTIALING SYSTEMS

A credential is generally a file that introduces one party to another by referencing a usually known trusted party. When credit is used for, references are usually requested. The credit of the references is determined and they are contacted to discover out the tested of the applicant. Credit cards are generally used to credential an individual to achieve more credit cards.

APPLICATIONS OF CRYPTOGRAPHY IN DATA PRIVACY

INTEGRITY IN TRANSMISSION

Some users of communication systems are not as much worried concerning secrecy as about integrity. In a computer funds transfer, the sum sent from one account to another is usually public knowledge.

If an operating tapper can bring in a false transfer, funds can be shared illegally. An inaccuracy in an individual bit can cause millions of dollars to be wrongly credited or debited. Cryptographic methods are generally used to provide that intentional or accidental modification of transmitted data does not cause flawed actions to appear.

INTEGRITY IN STORAGE

The central meaning of assuring integrity of accumulated data has previously been access control. Access control contains systems of locks and keys, guards, and other approaches of a physical or logical feature.

The recent advent of computer viruses has altered this to an important degree, and the use of cryptographic checksums for assuring the integrity of stored data is becoming broad.

ELECTRONIC SIGNATURES

Electronic signatures are a means of monetary a lawfully binding transaction among two or more parties. It can be as functional as a physical signature, electronic signatures should be at least as hard to fake at least as simple to use, and accepted in a court of law as binding upon some parties to the operation.

The necessity for these electronic signatures is especially intense in business dealings wherein the parties to an agreement are not in the similar physical vicinity.

CONCLUSION

How math secures the Internet

- Hiding data that everyone can see.
- Agreeing on secrets in public.
- Speech that can't be impersonated.

Math that will change the world

- Protecting data without having it.
- Checking proofs we can't see.
- Working on data that we can't access.

REFERENCES

- [1] ANSI X9.31-1998, American National Standard X9.31, Appendix A.2.4, Public Key Cryptography Using Reversible Algorithms for the Financial Services Industry (rDSA). Technical report, Accredited Standards Committee X9, Available at <http://www.x9.org>, 2001.
- [2] J.L. Carter and M.N. Wegman. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 1981.
- [3] Manuel Blum and Shafi Goldwasser. An efficient probabilistic public-key encryption scheme which hides all partial information, In *CRYPTO '84: Proceedings of the 4th Annual International Cryptology Conference, Advances in Cryptology*, 1984.
- [4] Dan Boneh and Richard J. Lipton. Algorithms for black-box fields and their application to cryptography (extended abstract). In *CRYPTO '96: Proceedings of the 16th Annual International Cryptology Conference, Advances in Cryptology*, Springer, 1996.
- [5] ANSI X9.62-1999. The Elliptic Curve Digital Signature Algorithm (ECDSA), Technical report, American Bankers Association, 1999.
- [6] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*, Cambridge University Press, New York, NY, USA, 2004.

STUDIES ON DYNAMIC PROGRAMMING

Project Report submitted to

ST.MARY'S COLLEGE (AUTONOMOUS), THOOTHUKUDI.

Affiliated to

MANONMANIAM SUNDARANAR UNIVERSITY, TIRUNELVELI.

In partial fulfillment of the requirement for the award of degree of

Bachelor of Science in Mathematics

Submitted by

NAME	REG.NO.
ANTONY ROSI. A	19AUMT03
MAHALAKSHMI. G	19AUMT18
MARIASUJI. B	19AUMT25
MICHAL FELIX. M	19AUMT27
PALKANI. P	19AUMT31

Under the Guidance of

Dr. V. L. STELLA ARPUTHA MARY M.Sc., M. Phil., B.Ed., Ph. D.,

Head & Assistant Professor of Mathematics

ST.MARY'S COLLEGE (AUTONOMOUS), THOOTHUKUDI.



Department of Mathematics

St. Mary's College (Autonomous), Thoothukudi

(2021 - 2022)

CERTIFICATE

We hereby declare that the project report entitled "STUDIES ON DYNAMIC PROGRAMMING" being submitted to St. Mary's College (Autonomous), Thoothukudi affiliated to Manonmaniam Sundaranar University, Tirunelveli in partial fulfillment for the award of degree of Bachelor of Science in Mathematics and it is a record of work done during the year 2021 - 2022 by the following students:

ANTONY ROSI. A
MAHALAKSHMI. G
MARIASUJI. B
MICHAL FELIX. M
PALKANI. P

19AUMT03
19AUMT18
19AUMT25
19AUMT27
19AUMT31

V.L. Stella Arputha Mary
Signature of the Guide

V.L. Stella Arputha Mary
Signature of the HOD
Dr. V.L. Stella Arputha Mary
M.Sc., M.Phil., B.Ed., Ph.D.,
Head & Asst Professor of Mathematics
St. Mary's College (Autonomous)
Thoothukudi-628 001.

SAD
Signature of the Examiner

Lucia Rose
Signature of the Principal
St. Mary's College (Autonomous)
Thoothukudi - 628 001.

DECLARATION

We hereby declare that the project reported entitled "STUDIES ON DYNAMIC PROGRAMMING", is our original work. It has not been submitted to any University for any degree or diploma.

A. Anth 2i

(ANTONY ROSI. A)

Mahalakshmi. G

(MAHALAKSHMI. G)

B. Mariasujji

(MARIASUJI. B)

M. Michal Felix.

(MICHAL FELIX. M)

P. Palkani

(PALKANI. P)

ACKNOWLEDGEMENT

First of all, we thank Lord Almighty for showering his blessings to undergo this project.

With immense pleasure, we register our deep sense of gratitude to our guide and the Head of the Department, Dr. V. L. Stella Arputha Mary M.Sc., M. Phil., B.Ed., Ph.D. for having imparted necessary guidelines throughout the period of our investigation.

We thank our beloved Principal, Rev. Dr. Sr. A.S.J. Lucia Rose M.Sc., PGDCA., M. Phil., Ph.D., for providing us the help to carry out our project work successfully.

Finally, we thank all those who extended their helping hands regarding this project.

DYNAMIC PROGRAMMING

CONTENT

1.1. Introduction.....	1
1.2. Distinguishing Characteristics of Dynamic Programming.....	2
1.3. Dynamic Programming Approach.....	4
1.4. Formulation of Dynamic Programming Problems.....	5
1.5. Optimal Subdivision Problem.....	36
1.6. System Reliability.....	53
1.7. Solution of L.P.P. by Dynamic Programming.....	56
1.8. Application of Dynamic Programming.....	61
1.9. Deterministic Dynamic Programming.....	62
1.10. Probabilistic Dynamic Programming.....	62
1.11. Applications.....	65
1.12. Conclusion.....	67
1.13. Reference.....	68

DYNAMIC PROGRAMMING

“True optimization is the revolutionary contribution of modern research to decision processes.”

- George Dantzig

While considering the situations of allocation, transportation, assignment, scheduling and planning, it was assumed that the values of decision variables do not change over the planning horizon. Thus these problems were of static nature and were solved as specific situations occurring at a certain moment. However, we come across a number of situations where the decision variables vary with time, and these situations are considered to be dynamic in nature. The technique dealing with these types of problems is called Dynamic Programming. It will be shown in the body of the chapter that time element is not an essential variable, rather any multistage situation in which a series of decisions are to be made is considered a dynamic programming problem.

Dynamic Programming is a technique in a computer programming that helps to efficiently solve a class of problems that have overlapping subproblems and optimal substructure property.

1.1 INTRODUCTION

In optimization problems involving a large number of decision variables or the inequality constraints, it may not be possible to use the methods of calculus for obtaining a solution. Classical mathematics handles the problems in a way to find the optimal values for all the decision variables simultaneously which for large problems rapidly increases the computations that become uneconomical or difficult to handle even by the available computers. The obvious solution is to split up the original large problem into smaller subproblems involving a few variables and that is precisely what the dynamic programming does. It uses recursive equations to solve a large, complex problem, broken into a series of interrelated decision stages (subproblems) wherein the outcome of the decision one stage affects the decisions at the remaining stages.

Dynamic programming is a mathematical technique dealing with the optimization of multistage decision problems. The technique was originated in 1952 by Richard Bellman and G.B. Dantzig, and was initially referred to as the *stochastic linear*

programming. Today dynamic programming has been developed as a mathematical technique to solve a wide range of decision problems and it forms an important part of every operation researcher's tool kit.

Though the originator of the technique, Richard Bellman, himself, has said, "we have coined the term 'dynamic programming' to emphasize that there are problems in which time plays an essential role", yet, in many dynamic programming problems time is not a relevant variable. For example, a decision regarding allocation of a fixed quantity of resources to a number of alternative uses constitutes one decision to be taken at one time, but the situation can be handled as a dynamic programming problem. As another instance, suppose a company has marked capital C to be spent on advertising its products through three different media i.e., of newspaper, radio and television. In each media the advertisement can appear a number of times per week. Each appearance has associated with it certain costs and returns. How many times the product should be advertised in each media so that the returns are maximum and the total cost is within the prescribed limit? In this situation time is not a variable, but the problem can be divided into stages and solved by dynamic programming.

1.2 DISTINGUISHING CHARACTERISTICS OF DYNAMIC PROGRAMMING

The important features of dynamic programming which distinguish it from other quantitative techniques of decision-making can be summarized as follows :

1. *Dynamic programming splits the original large problem into smaller subproblems (also called stages) involving only a few variables, wherein the outcome of decision at one stage affects the decisions at the remaining stages.*
2. *It involves a multistage process of decision-making.* The points at which decisions are called for are called stages. The stages may be certain time intervals or certain subdivisions of the problems, for which independent feasible decisions are possible. Each stage can be thought of having a beginning and an end. The stages come in a sequence, the end of a stage forming the beginning of the next stage.
3. *In dynamic programming, the variable that links up two stages is called a state variable.* At any stage, the status of the problem can be described by the values the state variable can take. These values are referred to as states. Each stage may have, associated with it, a certain number of states. It is not essential to know about the previous decisions and how the states arise. This enables us to consider decisions one at a time.

4. *In dynamic programming the outcome of decisions depends upon a small number of variables; that is, at any stage only a few variables should define the problem. For example, in the production smoothening problem, all that one needs to know at any stage is the production capacity, cost of production in regular and overtime, storage costs and the time remaining to the last decision.*
5. *A stage decision does not alter the number of variables on which the outcome depends, but only changes the numerical value of these variables. For the production smoothening problem, the number of variables which describe the problem i.e., production capacity, production costs, storage costs and time to the last decisions, remain the same at all stages. No variable is added or dropped. The effect to decision at any stage will be to alter the used production capacity, storage cost, production cost and time remaining to the last decision.*
6. *Principle of Optimality. Dynamic programming is based on Bellman's Principle of Optimality, which states, "An optimal policy (a sequence of decisions) has the property that whatever the initial state and decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision". This principle implies that a wrong decision taken at one stage does not prevent from taking of optimum decisions for the remaining stages. For example, in a production scheduling problem, wrong decisions made during first and second months do not prevent taking correct decisions during third, fourth month, etc. Using this principle of optimality, we find the best policy by solving one stage at time, and then adding a series of one-stage-problems until the overall optimum of the original problem is attained.*
7. *Bellman's principle of optimality forms the basis of dynamic programming technique. With this principle in mind, recursive equations are developed to take optimal decision at each stage. A recursive equation expresses subsequent state conditions and it is based on the fact that a policy is 'optimal' if the decision made at each stage results in overall optimality over all the stages and not only for the current stage.*
8. *Dynamic programming provides a systematic procedure wherein starting with the last stage of the problem and working backwards one makes an optimal decision for each stage of the problem. The information for the last stage is the information derived from the previous stages. It may be noted that dynamic programming problems can also be solved by working forward i.e., starting with the first stage and then working forward upto the last stage.*

1.3 DYNAMIC PROGRAMMING APPROACH :

Before discussing the solutions to numerical problems, it will be worthwhile to know a little more about some fundamental concepts of dynamic programming. The first concept is *stage*. As already discussed, the problem is broken down into subproblems and each subproblem is referred to as a stage. A stage signifies a portion of decision problem for which a separate decision can be made. At each stage there are a number of alternatives and the decision-making process involves the selection of one feasible alternative which may be called as *stage decision*. The stage decision may not be optimal for the considered stage, but contributes to make an overall optimal decision for the entire problem.

The other important concept is *state*. A state represents the status of the problem at a particular stage. The variables which specify the condition of decision process and summarize the current '*status*' of the system are called state variables. For example, in the capital budgeting problem, the capital is the state variable. The amount of capital allocated to the present stage and the preceding stages (or the capital remaining) defines the status of the problem. The number of state variables should be as small as possible. With the increase in number of state variables, increases the difficulty of problem solving.

The procedure adopted in the analysis of dynamic programming problems can be summarized as follows:

1. Define the problem variables, determine the objective function and specify the constraints.
2. Define the stages of the problem. Determine the state variables whose values constitute the state at each stage and the decision required at each stage. Specify the relationship by which the state at one stage can be expressed as a function of the state and decisions at the next stage.
3. Develop the recursive relationship for the optimal return function which permits computation of the optimal policy at any stage. Decide whether to follow the forward or the backward method to solve the problem. Specify the optimal return function at stage 1, since it is generally a bit different from the general optimal return function for the other stages.
4. Make a tabular representation to show the required values and calculations for each stage.
5. Find the optimal decision at each stage and then the overall optimal policy. There may be more than one such optimal policy.

1.4 FORMULATION OF DYNAMIC PROGRAMMING PROBLEMS

Consider a situation wherein a certain quantity 'R' of a resource (such as men, machines, money, material, etc.) is to be distributed among 'n' number of different activities. The return 'P' depends upon the activities and the quantities of resource allotted to them and the objective is to maximize the total return.

If $p_i(R_i)$ denotes the return from the i th activity with the resource R_i , then the total return may be expressed as

$$P(R_1, R_2, \dots, R_n) = p_1(R_1) + p_2(R_2) + \dots + p_n(R_n). \quad \dots (1.1)$$

The quantity of the resource R is limited, which gives rise to the constraint

$$R = R_1 + R_2 + \dots + R_n, \quad R_i \geq 0, i = 1, 2, \dots, n. \quad \dots (1.2)$$

The problem is to maximize the total return given by equation (1.1) subject to constraint (1.2). If,

$$\begin{aligned} f_n(R) &= \text{Max}_{0 \leq R_i \leq R} [P(R_1, R_2, \dots, R_n)] \\ &= \text{Max}_{0 \leq R_i \leq R} [p_1(R_1) + p_2(R_2) + \dots + p_n(R_n)], \end{aligned} \quad \dots (1.3)$$

then $f_n(R)$ is the maximum return from the distribution of the resource R to the n activities. Let us now allocate the resource to the activities, one by one, starting from the last i.e., nth activity. An expression connecting $f_n(R)$ and $f_{n-1}(R)$ for arbitrary values of R and n may now be obtained with the help of principle of optimality. If R_n is the quantity of resource allocated to the nth activity such that $0 \leq R_n \leq R$, then regardless of the values of R_n , a quantity $(R - R_n)$ of the resource will be distributed amongst the remaining $(n - 1)$ activities. Let $f_{n-1}(R - R_n)$ denote the return from the $(n - 1)$ activities, then the total return from all the n activities will be

$$p_n(R_n) + f_{n-1}(R - R_n).$$

An optimal choice of R_n will maximize the above function and thus the fundamental dynamic programming model may be expressed as

$$f_n(R) = \text{Max}_{0 \leq R_i \leq R} [p_n(R_n) + f_{n-1}(R - R_n)], \quad n = 2, 3, \dots, \quad \dots (1.4)$$

where $f_n(R)$, when $n = 1$ is obtained from equation (1.3) as

$$f_1(R) = p_1(R). \quad \dots (1.5)$$

Equation (1.5) gives the return from the first activity when whole of the resource R is allocated to it. Only $f_1(R)$ is known, equation (1.4) provides a relation to evaluate $f_2(R)$, $f_3(R)$, ... This recursive process ultimately leads to the value of $f_{n-1}(R)$ and finally $f_n(R)$ at which the process stops.

EXAMPLE 1.4 - 1 (Employment Smoothing Problem)

A firm has divided its marketing area into three zones. The amount of sales depends upon the number of salesmen in each zone. The firm has been collecting the data regarding sales and salesmen in each area over a number of past years.

The information is summarized in table 1.1. For the next year firm has only 9 salesmen and the problem is to allocate these salesmen to three different zones so that sales are maximum.

TABLE 1.1
Profit in thousands of rupees

No. of Salesman	Zone 1	Zone 2	Zone 3
0	30	35	42
1	45	45	54
2	60	52	60
3	70	64	70
4	79	72	82
5	90	82	95
6	98	93	102
7	105	98	110
8	100	100	110
9	90	100	110

Solution. In this problem the three zones represent the three states and the number of salesman the state variables.

Stage 1: We start with zone 1. The amount of sales corresponding to different number of salesmen allocated to zone 1 are given in table 1.1 and are reproduced in table 1.2.

TABLE 1.2

Zone 1

<i>No. of salesmen :</i>	0	1	2	3	4	5	6	7	8	9
<i>Profit (000' of ₹) :</i>	30	45	60	70	79	90	98	105	100	90

Stage 2: Now consider the first two zones, zone 1 and 2. Nine salesmen can be divided among two zones in 10 different ways: as 9 in zone 1 and 0 in zone 2, 8 in zone 1 and 1 in zone 2, 7 in zone 1 and 2 in zone 2, etc. Each combination will have associated with it certain returns. The returns for all number of salesmen (total) 9, 8, 7, ..., 0 are shown in table 1.3.

For a particular number of salesmen, the profits for all possible combinations can be read along the diagonal. Max. profits are marked by*.

TABLE 1.3

<i>Zone 1</i> x_1 $f_1(x_1) :$		0	1	2	3	4	5	6	7	8	9
<i>Zone 2</i> x_2 $f_2(x_2) :$		30	45	60	70	79	90	98	105	100	90
0	35	65*	80*	95*	105*	114	125*	133	140	135	125
1	45	75	90	105*	115*	124	135*	143*	150	145	
2	52	82	97	112	122	131	142	150	157		
3	64	94	109	124	134	143*	154*	162			
4	72	102	117	132	142	151	162				
5	82	112	127	142	152	161					
6	93	123	138	153	163*						
7	98	128	143	158							
8	100	130	145								
9	100	130									

Stage 3: Now consider the distribution of 9 salesmen in three zones 1, 2 and 3. The decision at this stage will result in allocating certain number of salesmen to zone 3 and the remaining to zone 2 and 1 combined; and then by following the backward process, they will be distributed to zones 2 and 1.

For a total of 9 salesmen to be allocated to the three zones, the returns are shown in table 1.4 below.

TABLE 1.4

No. of salesmen :	0	1	2	3	4	5	6	7	8	9
Total profit $f_2(x_2) + f_1(x_1)$:	65	80	95	105	115	125	135	143	154	163
Salesman in zone 2 + zone 1 ($x_2 + x_1$):	0+0	0+1	0+2	0+3	1+3	0+5	1+5	3+4	3+5	6+3
				1+2				1+6		
No. of salesmen in zone 3:	9	8	7	6	5	4	3	2	1	0
Profit $f_3(x_3)$:	110	110	110	102	95	82	70	60	54	42
Total profit $f_3(x_3) + f_2(x_2) + f_1(x_1)$:	175	190	205	207	210	207	205	203	208	205

From table 1.4, the maximum profit for 9 salesman is ₹ 2,10,000 if 5 salesmen are allotted to zone 3 and from the remaining four, 1 is allotted to zone 2 and 3 to zone 1.

EXAMPLE 1.4 - 2 (Capital Budgeting Problem)

A manufacturing company has three sections producing automobile parts, bicycle parts and sewing machine parts respectively. The management has allocated ₹ 20,000 for expanding the production facilities. In the auto parts and bicycle parts sections, the production can be increased either by adding new machines or by replacing some old inefficient machines by automatic machines. The sewing machine parts section was started only a few years back and thus the additional amount can be invested only by adding new machines to the sections. The cost of adding and replacing the machines, along with the associated expected returns in the different sections is given in table 1.5. Select a set of expansion plans which may yield the maximum return.

TABLE 1.5

Alternatives		Auto parts section		Bicycle parts Section		Sewing machine parts section	
		Cost (₹)	Return (₹)	Cost (₹)	Return (₹)	Cost (₹)	Return (₹)
1.	No expansion	0	0	0	0	0	0
2.	Add new machines	4,000	8,000	8,000	12,000	2000	8000
3.	Replace old machines	6,000	10,000	12,000	18,000	-	-

Solution: Here each section of the company is a stage. At each stage there are a number of alternatives for expansion. Capital represents the state variable. Let us consider the first stage - the auto parts sections. There are three alternatives: no expansion, add new machines and replace old machines. The amount that may be allocated to stage 1 may vary from 0 to ₹ 20,000; of course, it will be overspending if it is more than ₹ 6000. The returns of the various alternatives are given in table 1.6.

TABLE 1.6

Stage 1 : Auto parts section

State x_1 (000' of ₹)	Evaluation of alternatives (Values in thousands of rupees)			Optimal solution	
	1	2	3		
	Cost $C_{11} = 0$ Return	Cost $C_{12} = 4$ Return	Cost $C_{13} = 6$ Return	Optimal Return	Decision
0	0	-	-	0	1
2	0	-	-	0	1
4	0	8	-	8	2
6	0	8	10	10	3

8	0	8	10	10	3
10	0	8	10	10	3
12	0	8	10	10	3
14	0	8	10	10	3
16	0	8	10	10	3
18	0	8	10	10	3
20	0	8	10	10	3

When the capital allocated is zero or ₹ 2,000, only first alternative (no expansion) is possible. Return is, of course, zero. When the amount allocated is ₹ 4,000, alternative 1 or 2 are possible with returns of ₹ 0 and ₹ 8,000. So we select alternative 2 and when the amount allocated is ₹ 6,000, all the three alternatives are possible, giving returns of zero, ₹ 8,000 and ₹ 10,000 respectively. So we select alternative 3 with a return of ₹ 10,000 and so on.

Stage 2: Let us now move to stage 2. Here, again, three alternatives are available. The computations are carried out in table 1.7.

TABLE 1.7

Stage 2 : Bicycle parts section (+ Auto parts section)

State x_2 (000' of ₹)	Evaluation of alternatives (Values in thousands of rupees)			Optimal solution	
	1	2	3		
	Cost $C_{12} = 0$ Return	Cost $C_{22} = 8$ Return	Cost $C_{23} = 12$ Return	Optimal Return	Decision
0	$0 + 0 = 0$	-	-	0	1
2	$0 + 0 = 0$	-	-	0	1
4	$0 + 8 = 8$	-	-	8	1

6	$0 + 10 = 10$	-	-	10	1
8	$0 + 10 = 10$	$12 + 0 = 12$	-	12	2
10	$0 + 10 = 10$	$12 + 0 = 12$	-	12	2
12	$0 + 10 = 10$	$12 + 8 = 20$	$18 + 0 = 18$	20	2
14	$0 + 10 = 10$	$12 + 10 = 22$	$18 + 0 = 18$	22	2
16	$0 + 10 = 10$	$12 + 10 = 22$	$18 + 8 = 26$	26	3
18	$0 + 10 = 10$	$12 + 10 = 22$	$18 + 10 = 28$	28	3
20	$0 + 10 = 10$	$12 + 10 = 22$	$18 + 10 = 28$	28	3

Here state x_2 represents the total amount allocated to the current stage (stage 2) and the preceding stage (stage 1). Similarly, the return also is the sum of the current stage and the preceding stage (Principle of optimality). Thus when $x_2 < ₹8,000$, only the first alternative (no expansion) is possible. But with $x_2 = ₹8,000$, a return of ₹12,000 is possible by selecting the second alternative (add new machines). With $x_2 = ₹12,000$, three alternatives are possible with the maximum return of ₹20,000 from alternative 2. The optimal policy consists of a set of two decisions, namely adopt alternative 2 at second stage (table 1.7) and again alternative 2 at the first stage (table 1.6).

Stage 3 : The computations for stage 3 are given in table 1.8

TABLE 1.8

Stage 3 : Sewing machine parts section (+ Bicycle parts section + Auto parts section)

State x_2 (000' of ₹)	Evaluation of alternatives (Values in thousands of rupees)		Optimal solution	
	Core $C_{31} = 0$ Return	Cost $C_{32} = 2$ Return	Optimal Return	Decision
0	$0 + 0 = 0$	-	0	1
2	$0 + 0 = 0$	$8 + 0 = 8$	8	2

4	$0 + 8 = 8$	$8 + 0 = 8$	8	1,2
6	$0 + 10 = 10$	$8 + 8 = 16$	16	2
8	$0 + 12 = 12$	$8 + 10 = 18$	18	2
10	$0 + 12 = 12$	$8 + 12 = 20$	20	2
12	$0 + 20 = 22$	$8 + 12 = 20$	20	1,2
14	$0 + 22 = 22$	$8 + 20 = 28$	28	2
16	$0 + 26 = 26$	$8 + 22 = 30$	30	2
18	$0 + 28 = 28$	$8 + 26 = 34$	34	2
20	$0 + 28 = 28$	$8 + 28 = 36$	36	2

For $x_3 = 20,000$, the optimal decision for stage 3 is alternative 2, which gives a total return of ₹ 36,000. This involves a cost of ₹ 2,000 and leaves ₹ 18,000 to be allocated for stages 2 and 1 combined. From table 1.7, for allocation of ₹ 18,000, alternative 3 is to be chosen which costs ₹ 12,000. For remaining sum of ₹ 6,000, from table 1.6, decision alternative 3 is to be selected. Thus the optimal policy of expanding production facilities is 3-3-2, which can be elaborated as: in bicycle parts section, and add new machines to the sewing machines parts section. This policy gives the optimal return of ₹ 36,000.

EXAMPLE 1.4 - 3

The owner of a chain of four grocery stores has purchased six crates of fresh strawberries. The following table gives the estimated profits at each store when it is allocated various number of boxes:

TABLE 1.9

		Stores			
		1	2	3	4
Number of Boxes	0	0	0	0	0
	1	4	2	6	2

2	6	4	8	3
3	7	6	8	4
4	7	8	8	4
5	7	9	8	4
6	7	10	8	4

The owner does not wish to split crates between stores, but is willing to make zero allocations. Find the allocation of six crates so as to minimize the profits.

Solution :

This problem is similar to the allocation of salesmen to different zones. Here stores represent the stages and number of boxes represent the state variables. Thus the problem involves 4 stages and 6 state variables. Let x_1, x_2, x_3 and x_4 be the number of crates allocated to the 4 stores and $f_1(x_1), f_2(x_2), f_3(x_3)$ and $f_4(x_4)$ be the respective profits. Then the problem is

$$\text{maximize } Z = f_1(x_1) + f_2(x_2) + f_3(x_3) + f_4(x_4),$$

$$\text{subject to } x_1 + x_2 + x_3 + x_4 \leq 6,$$

where x_1, x_2, x_3, x_4 are non-negative integers.

Stage 1: The estimated profits corresponding to different number of boxes allocated to store 1 are given in table 1.9 and are reproduced in table 1.10

TABLE 1.10

Store 1

No. of boxes, x_1 :	0	1	2	3	4	5	6
Profit $f_1(x_1)$:	0	4	6	7	7	7	7

Stage 2: Now consider the first two stores, store 1 and 2. Six boxes can be divided among the two stores in 7 different ways: as 6 in store 1 and 0 in store 2, 5 in store 1 and 1 in store 2, etc. Each combination will have associated with it certain profits. The profits for all the total number of boxes, such as 6, 5, 4, ..., 0 are shown in table 1.11.

TABLE 1.11

<i>Store 1</i> $x_1 :$	0	1	2	3	4	5	6
$f_1(x_1) :$	0	4	6	7	7	7	7
<i>Store 2</i> x_2							
$f_2(x_2)$							
0	0	0*	4*	6*	7	7	7
1	2	2	6*	8*	9	9	9
2	4	4	8*	10*	11	11	11
3	6	6	10*	12*	13	13	13
4	8	8	12*	14*	14*	14*	14*
5	9	9	13	13	13	13	13
6	10	10	13	13	13	13	13

For a particular number of boxes, the profits for all possible combinations can be read along the diagonal. Maximum profits are marked by *. Thus the optimal profits and corresponding allocations of boxes to the two stores are given by:

TABLE 1.12

<i>Boxes</i> :	0	1	2	3	4	5	6
$f_2(x_2) + f_1(x_1) :$	0	4	6	8	10	12	14
$x_2 + x_1 :$	0+0	0+1	1+1	2+1	3+1	4+1	4+2
			0+2	1+2	2+2	3+2	

Stage 3: Now Consider the distribution of 6 boxes to three stores 1, 2 and 3. The decision at this stage will result in allocating certain number of boxes to store 3 and the remaining to stores 2 and 1 combined and then by following the backward process, they will be distributed to stores 2 and 1. The profits for all the total number of boxes, such as 6, 5, 4, ..., 0 are shown in table 1.13.

TABLE 1.13

Store 1+2	Boxes ($x_2 + x_1$)	0	1	2	3	4	5	6
	$f_1(x_1) + f_2(x_2)$	0	4	6	8	10	12	14
Store x_3	$f_3(x_3)$							
0	0	0*	4	6	8	10	12	14
1	6	6*	10*	12*	14*	16*	18*	
2	8	8	12*	14*	16*	18*		
3	8	8	12	14	16			
4	8	8	12	14				
5	8	8	12					
6	8	8						

For any particular number of boxes, the profits for all possible combinations can be read along the diagonal. Maximum profits are marked by *. Thus the optimal profits and corresponding allocations of boxes to the three stores are given by table 1.14.

TABLE 1.14

Boxes	:	0	1	2	3	4	5	6
$f_1(x_1) + f_2(x_2) + f_3(x_3)$:	0	6	10	12	14	16	18
$x_3 + (x_2 + x_1)$:	0 + 0	1 + 0	1 + 1	2 + 1	2 + 2	2 + 3	2 + 4
					1 + 2	1 + 3	1 + 4	1 + 5

Stage 4: Now consider the distribution of 6 boxes to four stores. The corresponding profits for all possible combinations are given in table 7.15.

TABLE 1.15

<i>Boxes</i>	$\sum_{j=1}^3 x_j$:	0	1	2	3	4	5	6
	$\sum_{j=1}^3 f_j(x_j)$:	0	6	10	12	14	16	18
	x_4	:	6	5	4	3	2	1	0
	$f_4(x_4)$:	4	4	4	4	3	2	0
	$\sum_{j=1}^4 f_j(x_j)$:	4	10	14	16	17	18*	18*

Thus the maximum possible profit is 18 for $x_4 = 1$ or 0. Going back, eight optimal allocations can be traced, each yielding profit of 18.

TABLE 7.16

		x_1^*	x_2^*	x_3^*	x_4^*
<i>Possible Optimal allocations</i>	1	1	2	2	1
	2	1	3	1	1
	3	1	3	2	0
	4	1	4	1	0
	5	2	1	2	1
	6	2	2	1	1
	7	2	2	2	0
	8	2	3	1	0

EXAMPLE 1.4 – 4

An oil company has 8 units of money available for exploration of three sites. If oil is present at a site, the probability of finding it depends upon the amount allocated for exploiting the site, as given below.

TABLE 1.17
Units of money allocated

	0	1	2	3	4	5	6	7	8
Site 1	0.0	0.0	0.1	0.2	0.3	0.5	0.7	0.9	1.0
Site 2	0.0	0.1	0.2	0.3	0.4	0.6	0.7	0.8	1.0
Site 3	0.0	0.1	0.1	0.2	0.3	0.5	0.8	0.9	1.0

The probability that the oil exists at sites 1, 2 and 3 is 0.4, 0.3 and 0.2 respectively. Find the optimal allocation of money.

Solution: In this oil exploration problem, the objective is to maximize the probability of finding oil by allocating the available amount of money to the three potential oil sites.

Let x_1 , x_2 and x_3 be the units of money allocated to the sites 1, 2 and 3 respectively, and $p_1(x_1)$, $p_2(x_2)$ and $p_3(x_3)$ be the corresponding probabilities of finding oil, if it exists. Then actual probabilities of finding oil at the three sites are $p_1(x_1) \times 0.4$, $p_2(x_2) \times 0.3$ and $p_3(x_3) \times 0.2$.

Thus the objective function can be written as

$$\begin{aligned} \text{maximize } Z &= 0.4 p_1(x_1) + 0.3 p_2(x_2) + 0.2 p_3(x_3), \\ \text{subject to constraint } &x_1 + x_2 + x_3 \leq 8, \end{aligned}$$

where x_1, x_2, x_3 are non-negative integers.

The probabilities of finding the oil, taking into consideration the availabilities of oil at different sites, in the percentage form can be expressed as below.

TABLE 1.18
Unit of money allocated

	0	1	2	3	4	5	6	7	8
Site 1, $f_1(x_1)$:	0	0	4	8	12	20	28	36	40
Site 2, $f_2(x_2)$:	0	3	6	9	12	18	21	24	30
Site 3, $f_3(x_3)$:	0	2	2	4	6	10	16	18	20

Here the three sites are regarded as the three stages and the money allocated is the stage variable.

Stage 1: We start with site 1. The actual probabilities of finding the oil when expressed as percentages are shown in the table 1.19.

TABLE 1.19
Site 1

<i>Units of money allocated :</i>	0	1	2	3	4	5	6	7	8
$f_1(x_1)$:	0	0	4	8	12	20	28	36	40

Stage 2 : Now consider the first two sites 1 and 2. Eight units of money can be divided among the two sites in 9 different ways as shown in table 1.20.

TABLE 1.20

x_1 :	0	1	2	3	4	5	6	7	8
$f_1(x_1)$:	0	0	4	8	12	20	28	36	40
x_2 $f_2(x_2)$									
0 0	0*	0	4	8	12*	20*	28*	36*	40*
1 3	3*	3	7	11	15	23	31	39	
2 6	6*	6	10	14	18	26	34		
3 9	9*	9	13	17	21	29			
4 12	12*	12	16	20	24				
5 18	18	18	22	26					
6 21	21	21	25						
7 24	24	24							
8 30	30								

The optimal values of $f_2(x_2) + f_1(x_1)$ are given in table 1.21.

TABLE 1.21

Unit of money :	0	1	2	3	4	5	6	7	8
$f_2(x_2) + f_1(x_1) :$	0	3	6	9	12	20	28	36	40
$x_2 + x_1 :$	0 + 0	1 + 0	2 + 0	3 + 0	4 + 0	0 + 5	0 + 6	0 + 7	0 + 8
					0 + 4				

Stage 3: Now consider the allocation of 8 units of money to the three sites. The corresponding probabilities expressed as percentage are shown in table.

TABLE 1.22

Units of money :	0	1	2	3	4	5	6	7	8
$f_2(x_2) + f_1(x_1) :$	0	3	6	9	12	20	28	36	40
$x_2 + x_1 :$	0 + 0	1 + 0	2 + 0	3 + 0	4 + 0	0 + 5	0 + 6	0 + 7	0 + 8
					0 + 4				
$x_3 :$	8	7	6	5	4	3	2	1	0
$f_3(x_3) :$	20	18	16	10	6	4	2	2	0
$f_2(x_2) + f_1(x_1) + f_3(x_3) :$	20	21	22	19	18	24	30	38	40

Thus the maximum probability is 40%, which is obtained if $x_3 = 0$, $x_2 = 0$ and $x_1 = 8$ i.e., if entire 8 units of money are allocated to site 1 only.

EXAMPLE 1.4 – 5

A company manufacturing a certain product has a contract of supplying 40 units at the end of month 1 and 60 units at the end of month 2. The cost of manufacturing x units in any month is $c(x) = 100x + 0.4x^2$. The company has enough production facilities to manufacture 100 units a month. If the company produces more than 40 units in month 1, any excess units can be carried over to month 2. However, there is an

Inventory carrying cost of ₹ 1.60 for each unit carried over from month 1 to 2. How many units should the company produce each month to minimize the total cost assuming that there is no initial inventory?

Solution: Here month 1 and month 2 are the two stages and the number of units to be produced each month are the state variables. Let x_1 and x_2 be the number of units of the product to be produced in month 1 and 2 respectively and W be the amount of inventory at the end of month 1, which is to be carried to month 2.

Then $x_1 = 40 + W$ and $x_2 = 60 - W$.

$$\begin{aligned}\text{Cost incurred in month 1, } c_1(x_1) &= 100x_1 + 0.4x_1^2 \\ &= 100(40 + W) + 0.4(40 + W)^2 \\ &= 4,640 + 132W + 0.4W^2,\end{aligned}$$

$$\begin{aligned}\text{and Cost incurred in month 2, } c_2(x_2) &= 100x_2 + 0.4x_2^2 \\ &= 100(60 - W) + 0.4(60 - W)^2 \\ &= 7,440 - 148W + 0.4W^2,\end{aligned}$$

∴ Total cost incurred in the two months, including the inventory carrying cost,

$$c_1(x_1) + c_2(x_2) = 12,080 - 16W + 0.8W^2 + 1.60W.$$

This cost is minimum if $\frac{d}{dW}(12,080 - 16W + 0.8W^2 + 1.60W) = 0$

$$\text{or if } -16 + 1.6W + 1.60 = 0$$

$$\text{or if } W = 9 \text{ units. } \therefore x_1 = 49 \text{ units, } x_2 = 51 \text{ units.}$$

$$\text{Minimum total cost} = 12,080 - 16 \times 9 + 0.8 \times 9^2 + 1.60 \times 9$$

$$\text{Minimum total cost} = ₹ 12,015.20.$$

EXAMPLE 1.4 - 6

A manufacturer has entered into a contract for the supply of the following number of units of a product at the end of each month:

Month	:	January	March	August	October	November	December
No. of units	:	10	5	20	3	6	30

The units manufactured during a month are available for supply at the end of the month or they may be kept in storage at a cost of ₹ 2 per unit per month. Each time the

manufacture of a batch of units is undertaken, there is a set-up cost of ₹ 400. Determine the production schedule which will minimize the total cost.

Solution. Here the six months represent the 6 stages and number of units to be manufactured are the state variables. We shall start from the last month of December and move backwards.

Month of December

The best decision is to produce 30 units with a cost of ₹ 400 towards the set-up cost and there is no storage cost.

Month of November

There are two alternatives:

1. Produce $(6 + 30) = 36$ units to satisfy the demand of November and December.

$$\text{Total cost} = ₹ (400 + 30 \times 2 \times 1) = ₹ 400.$$

2. Produce 6 units in Nov. + 30 units in Dec. involving 2 set-ups and no storage cost.

$$\text{Total cost} = ₹ (400 + 400) = ₹ 800.$$

∴ The optimum decision is to produce 36 units in Nov. and no units in Dec.

Month of October

Various alternatives are:

1. Produce $(3 + 6 + 30) = 39$ units in Oct.

$$\text{Total cost} = ₹ (400 + 6 \times 2 \times 1 + 30 \times 2 \times 2) = ₹ 532.$$

2. Produce $(3 + 6) = 9$ units in Oct. and 30 units in Dec.

$$\text{Total cost} = ₹ (400 \times 2 + 6 \times 2 \times 1) = ₹ 812.$$

3. Produce 3 units in Oct. and 36 units in Nov.

$$\text{Total cost} = ₹ (400 \times 2 + 30 \times 2 \times 1) = ₹ 860.$$

Note that as per the decision made in Nov., producing 3 units in Oct., 6 in Nov. and 30 in Dec. is already ruled out as it involves higher cost.

Thus the optimum decision is to produce 39 units in Oct. and nothing in Nov. and Dec.

Month of August

The various possible alternatives are:

1. Produce $(20 + 3 + 6 + 30) = 59$ units in August.

$$\text{Total cost} = ₹ (400 + 3 \times 2 \times 2 + 6 \times 2 \times 3 + 30 \times 2 \times 4) = ₹ 688.$$

2. Produce $(20 + 3 + 6) = 29$ units in August and 30 units in Dec.

$$\text{Total cost} = ₹ [(400 \times 2) + 3 \times 2 \times 2 + 6 \times 2 \times 3] = ₹ 848.$$

3. Produce $(20 + 3) = 23$ units in August and 36 units in Nov.

$$\text{Total cost} = ₹ [400 \times 2 + 3 \times 2 \times 2 + 30 \times 2 \times 1] = ₹ 872.$$

4. Produce 20 units in August and 39 units in Nov.

$$\text{Total cost} = ₹ [400 \times 2 + 6 \times 2 \times 1 + 30 \times 2 \times 2] = ₹ 932.$$

Thus the optimum decision is to produce 59 units in August and none in the following months.

Month of March

The various possible alternatives are :

1. Produce all $(5 + 20 + 3 + 6 + 30) = 64$ units in March.

$$\begin{aligned} \text{Total cost} &= ₹ [400 + 20 \times 2 \times 5 + 3 \times 2 \times 7 + 6 \times 2 \times 8 + 30 \times 2 \times 9] \\ &= ₹ 1,278. \end{aligned}$$

2. Produce $(5 + 20 + 3 + 6) = 34$ units in March and 30 units in Dec.

$$\begin{aligned} \text{Total cost} &= ₹ [400 \times 2 + 20 \times 2 \times 5 + 3 \times 2 \times 7 + 6 \times 2 \times 8] \\ &= ₹ 1,138. \end{aligned}$$

3. Produce $(5 + 20 + 3) = 28$ units in March and 36 units in Nov.

$$\begin{aligned} \text{Total cost} &= ₹ [400 \times 2 + 20 \times 2 \times 5 + 3 \times 2 \times 7 + 30 \times 2 \times 1] \\ &= ₹ 1,102. \end{aligned}$$

4. Produce $(5 + 20) = 25$ units in March and 39 units in Oct.

$$\begin{aligned} \text{Total cost} &= ₹ [400 \times 2 + 20 \times 2 \times 5 + 6 \times 2 \times 1 + 30 \times 2 \times 2] \\ &= ₹ 1,132. \end{aligned}$$

5. Produce 5 units in March and 59 units in August.

$$\begin{aligned} \text{Total cost} &= ₹ [400 \times 2 + 3 \times 2 \times 2 + 6 \times 2 \times 3 + 30 \times 2 \times 4] \\ &= ₹ 1,088. \end{aligned}$$

∴ The optimum decision is to produce 5 units in March and 59 units in August.
The total cost involved is ₹ 1,088.

Month of January

The various possible alternatives are:

1. Produce all 74 units in January.

$$\begin{aligned} \text{Total cost} &= ₹ [400 + 5 \times 2 \times 2 + 20 \times 2 \times 7 + 3 \times 2 \times 9 + 6 \times 2 \times 10 + 30 \times \\ &\quad 2 \times 11] \\ &= ₹ 1,534. \end{aligned}$$

2. Produce $(10 + 5 + 20 + 3 + 6) = 44$ units in January and 30 units in Dec.

$$\begin{aligned} \text{Total cost} &= ₹ [400 \times 2 + 5 \times 2 \times 2 + 20 \times 2 \times 7 + 3 \times 2 \times 9 + 6 \times 2 \times 10] \\ &= ₹ 1,274. \end{aligned}$$

3. Produce $(10 + 5 + 20 + 3) = 38$ units in January and 36 units in Nov.

$$\text{Total cost} = ₹ [400 \times 2 + 5 \times 2 \times 2 + 20 \times 2 \times 7 + 3 \times 2 \times 9 + 30 \times 2 \times 1] \\ = ₹ 1,214.$$

4. Produce $(10 + 5 + 20) = 35$ units in January and 39 units in Oct.

$$\text{Total cost} = ₹ [400 \times 2 + 5 \times 2 \times 2 + 20 \times 2 \times 7 + 6 \times 2 \times 1 + 30 \times 2 \times 2] \\ = ₹ 1,232.$$

5. Produce $(10 + 5) = 15$ units in January and 59 units in August.

$$\text{Total cost} = ₹ [400 \times 2 + 5 \times 2 \times 2 + 3 \times 2 \times 2 + 6 \times 2 \times 3 + 30 \times 2 \times 4] \\ = ₹ 1,108.$$

6. Produce 10 units in January, 5 units in March and 59 units in August.

$$\text{Total cost} = ₹ [400 \times 3 + 3 \times 2 \times 2 + 6 \times 2 \times 3 + 30 \times 2 \times 4] \\ = ₹ 1,488.$$

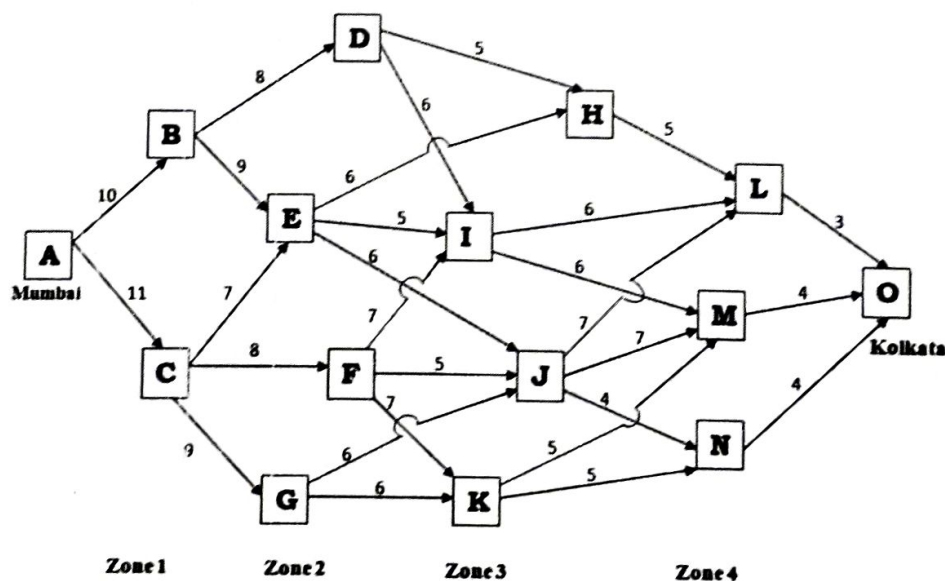
Thus the optimum decision is to produce 15 units in January and 59 units in August.

Therefore, the best production schedule that will minimize total cost and satisfy the demand from January till December is to produce 15 units in January and 59 units in August.

EXAMPLE 1.4 - 7

A salesman is planning a business tour from Mumbai to Kolkata in the course of which he proposes to cover one city from each of the company's different marketing zones en route.

FIGURE 1.1



As he has limited time at his disposal, he has to complete his tour in the shortest possible time. The network in the following figure shows the number of days' time involved for covering any of the various intermediate cities (time includes travel as well as working time). Determine the optimum tour plan.

Solution. Starting from A, the cities of various marketing zones may be considered as distinct stages.

Stage 1 : B or C ?

Stage 2 : D, E, F or G ?

Stage 3 : H, I, J or K?

Stage 4 : L, M or N?

Stage 5 : Best route to O.

Stage 1: At this stage it is not known whether B lies on the overall shortest route; but if it does, the shortest route from A to B is AB.

$$\left. \begin{array}{l} A \text{ to } B = 6 \\ A \text{ to } C = 11 \end{array} \right\} \text{the only routes}$$

Stage 2: It is not known whether D lies on the overall shortest route; but if it does, the only route from A is ABD, involving a total travel time of $= 10 + 8 = 18$ days.

Similarly,

$$ABE = 10 + 9 = 19$$

$$ACE = 11 + 7 = 18$$

$$ACF = 11 + 8 = 19$$

$$ACG = 11 + 9 = 20.$$

From the above, shortest routes are :

$$\left. \begin{array}{l} A \text{ to } D = 18 \\ A \text{ to } E = 18 \\ A \text{ to } F = 19 \\ A \text{ to } G = 20. \end{array} \right\}$$

Stage 3: It is not known whether H lies on the overall shortest route; but if it does, is it through D or E ?

Both D and E are reached in 18 days by the quickest route from (from the optimal result from stage 2).

Therefore,

$$ABH = 18 + 5 = 23$$

$$AEH = 18 + 6 = 24$$

Similarly,

$$ADI = 18 + 6 = 24$$

$$AEI = 19 + 7 = 26$$

$$AFJ = 18 + 6 = 24$$

$$\begin{aligned}
 AGJ &= 19 + 5 = 23 \\
 AGK &= 20 + 6 = 26 \\
 AFK &= 19 + 7 = 26 \\
 AGK &= 20 + 6 = 26.
 \end{aligned}$$

From the above, shortest routes from A are:

$$\left. \begin{aligned}
 A \text{ to } H &= 23 \\
 A \text{ to } I &= 23 \\
 A \text{ to } J &= 24 \\
 A \text{ to } K &= 26.
 \end{aligned} \right\}$$

Stage 4: Proceeding in the same way as for stage 3, we have

$$\begin{aligned}
 AHL &= 23 + 5 = 28 \\
 AIL &= 23 + 6 = 29 \\
 AHL &= 23 + 5 = 28 \\
 AJL &= 24 + 7 = 31 \\
 AIM &= 23 + 6 = 29 \\
 AJM &= 24 + 7 = 31 \\
 AKM &= 26 + 5 = 31 \\
 AJN &= 24 + 4 = 28 \\
 AKN &= 26 + 5 = 31.
 \end{aligned}$$

∴ The shortest routes from A are:

$$\left. \begin{aligned}
 A \text{ to } L &= 28 \\
 A \text{ to } M &= 29 \\
 A \text{ to } N &= 28.
 \end{aligned} \right\}$$

Final Stage: There are three alternatives to reach O from the 4th stage viz. LO, MO and NO.

Using the optimal times at 4th stage,

$$\left. \begin{aligned}
 ALO &= 28 + 3 = 31 \\
 AMO &= 29 + 4 = 33 \\
 ANO &= 28 + 4 = 32.
 \end{aligned} \right\}$$

Thus the shortest time from A to O = 31. Now we retrace the steps backwards along the network to identify the intermediate cities along the shortest route.

A – O — Final
 A – L – O — Stage 4
 A – H – L – O — Stage 3
 A – D – H – L – O — Stage 2
 A – B – D – H – L – O — Optimal route.

The problem of finding the shortest route is known as the stagecoach problem.

EXAMPLE 1.4 - 8

A dealer has to dispose of certain goods within 5 weeks' time. The market prices are fluctuating from week to week. It is estimated that the chances of getting ₹ 2,000 for the whole stock are 45%, chances of getting ₹ 2,500 are 35% and chances for getting ₹ 3,000 are 20%. If the goods are not sold in the first 4 weeks, then they will have to be disposed of in the fifth week at the prevailing market price in that week. When should the stocks be sold ?

Solution. The five weekly periods can be treated as five stages and the sale prices are the state variables. At each stage the dealer has to make a choice among the alternatives 'sell' and 'wait'. If the prevailing market price is more than that he expects in the following weeks, he should 'sell' and if it is less, he should 'wait'.

These types of problems can be conveniently handled by starting from the last stage and solving by *the backward induction or backward process*.

Fifth Week (stage): As it is essential to sell the stocks by the fifth week, he will get ₹ 2,000 or ₹ 2,500 or ₹ 3,000 depending upon the price prevailing in the market in that week.

Fourth Week (stage): The prevailing price may be ₹ 2,000, ₹ 2,500 or ₹ 3,000. If it is ₹ 3,000, he will naturally sell the stock. If it is less than ₹ 3,000, he may decide to sell or wait. He will sell if he will naturally sell the stock. If it is less than ₹ 3,000, he may decide to sell or wait. He will sell if the market price in 4th week is more than the expected return in the 5th week, otherwise he will wait.

$$\begin{aligned}\text{Now expected return in the 5th week} &= ₹ [2,000 \times .45 + 2,500 \times .35 + 3,000 \times .20] \\ &= ₹ 2,375.\end{aligned}$$

Thus if the market price in the 4th week is ₹ 3,000 or ₹ 2,500 he should sell; otherwise he should wait.

Third week (stage): If the prevailing price is ₹ 3,000 he should sell. If the price is less, he will still sell if he can get more than the expected return in the 4th week; otherwise he will wait.

$$\begin{aligned}\text{Expected return in the 4th week} &= ₹ [3,000 \times 0.20 + 2,500 \times 0.35 + 2,375 \times 0.45] \\ &= ₹ 2,544.\end{aligned}$$

Thus if the market price in the 3rd week is ₹ 3,000, he should sell; otherwise he should wait.

Second week (stage): If the prevailing price is ₹ 3,000 he should sell. If not, he should see the expected return in the third week.

$$\begin{aligned}\text{Expected return in the 3rd week} &= ₹ [3,000 \times 0.20 + 2,544 \times 0.35 + 2,635 \times 0.45] \\ &= ₹ 2,635.\end{aligned}$$

So, he should wait.

First week (stage): If the prevailing price is ₹ 3,000 he should sell; if not, expected return in the 2nd week will determine his decision.

$$\begin{aligned}\text{Expected return in the 2nd week} &= ₹ [3,000 \times 0.20 + 2,635 \times 0.35 + 2,635 \times 0.45] \\ &= ₹ 2,708.\end{aligned}$$

So, he should wait.

Thus we reach at the following optimal decision policy: The dealer should sell goods if the market price in the first, second and third weeks is ₹3,000; otherwise wait. In the fourth week, if he can get ₹ 2,500 or ₹ 3,000 he should sell; if it is less than ₹ 2,500 he should wait. If the goods remain unsold upto the 5th week he should dispose of the stock at whatever value he gets in that week.

EXAMPLE 1.4 - 9 (Cargo-Loading Problem)

In a cargo loading problem, there are 4 items of different weights/unit and different value / unit as given below.

Item (i)	Weight / unit (w_i kg/unit)	Value / unit (p_i ₹/unit)
1	1	1
2	3	5
3	4	7
4	6	11

The maximum cargo load is restricted to 17. How many units of each item be loaded to maximize the value?

Solution. It is a four-stage problem, each item represents a stage. The state of the system is represented by the weight capacity available for allocation to stages 1, 2, 3,

4 and is denoted by x_i which varies from 0 to 17. If a_i is the number of units of item i , then the problem is

$$\begin{aligned} \text{maximize } Z &= \sum_{i=1}^4 a_i p_i, \\ \text{subject to } \sum_{i=1}^4 a_i w_i &\leq W. \end{aligned}$$

Stage 1: Here, $w_1 = 1 \text{ kg / unit}$, $p_1 = ₹1/\text{unit}$; $\frac{W}{w_1} = \frac{17}{1} = 17$.

$$\therefore a_1 = 0, 1, 2, \dots, 17.$$

Stage 2: Here, $w_2 = 3 \text{ kg / unit}$, $p_2 = ₹5/\text{unit}$;

$$\frac{W}{w_2} = \frac{17}{3} = 5.67 (= 5, \text{integral value})$$

$$\therefore a_2 = 0, 1, 2, \dots, 5.$$

TABLE 1.23

x_i	Stage 1		Stage 2		Stage 3		Stage 4		$f_i^*(x_i)$
	$w_1 = 1, p_1 = 1$ $a_1 = 0, 1, 2, \dots, 17$		$w_2 = 3, p_2 = 5$ $a_2 = 0, 1, 2, \dots, 5$		$w_3 = 4, p_3 = 7$ $a_2 = 0, 1, 2, 3, 4$		$w_4 = 6, p_4 = 11$ $a_2 = 0, 1, 2$		
	a_1	$f_1(x_1)$	a_2	$f_2(x_2)$	a_3	$f_3(x_3)$	a_4	$f_4(x_4)$	
0	0	0*	0	—	0	—	0	—	0
1	1	1*	0	—	0	—	0	—	1
2	2	2*	0	—	0	—	0	—	2
3	3	3	1	5 + 0 = 5*	0	—	0	—	5
4	4	4	1	5 + 1 = 6	1	7 + 0 = 7*	0	—	7
5	5	5	1	5 + 2 = 7	1	7 + 1 = 8*	0	—	8
6	6	6	2	10 + 0 = 10	1	7 + 2 = 9	1	11 + 0 = 11*	11
7	7	7	2	10 + 1 = 11	1	7 + 5 = 12*	1	11 + 1 = 12*	12
8	8	8	2	10 + 2 = 12	2	14 + 0 = 14*	1	11 + 2 = 13	14
9	9	9	3	15 + 0 = 15	2	14 + 1 = 15	1	11 + 5 = 16*	16
10	10	10	3	15 + 1 = 16	2	14 + 2 = 16	1	11 + 7 = 18*	18
11	11	11	3	15 + 2 = 17	2	14 + 5 = 19*	1	11 + 8 = 19*	19
12	12	12	4	20 + 0 = 20	3	21 + 0 = 21	2	22 + 0 = 22*	22
13	13	13	4	20 + 1 = 21	3	21 + 1 = 22	2	22 + 1 = 23*	23
14	14	14	4	20 + 2 = 22	3	21 + 2 = 23	2	22 + 2 = 24*	24
15	15	15	5	25 + 0 = 25	3	21 + 5 = 26	2	22 + 5 = 27*	27
16	16	16	5	25 + 1 = 26	4	28 + 0 = 28	2	22 + 7 = 29*	29
17	17	17	5	25 + 2 = 27	4	28 + 1 = 29	2	22 + 8 = 30*	30

Stage 3: Here, $w_3 = 4 \text{ kg / unit}$, $p_3 = ₹7/\text{unit}$; $\frac{W}{w_3} = \frac{17}{4} = 4.25$.

$$\therefore a_3 = 0, 1, 2, 3, 4.$$

Stage 4: Here, $w_4 = 6 \text{ kg / unit}$, $p_4 = ₹11/\text{unit}$; $\frac{W}{w_4} = \frac{17}{6} = 2.83$.

$$\therefore a_2 = 0, 1, 2.$$

Let $f_1(x_1)$, $f_2(x_2)$, $f_3(x_3)$ and $f_4(x_4)$ be the values of the loaded items at stage 1, 2, 3 and 4 respectively. The computation for different stages are given in table 1.23.

As seen from table, for total load of 17 kg, the maximum value of cargo items is ₹ 30 (= 22 + 8 = 22 + 7 + 1), which is achieved if we load 1 unit of item 1, 1 unit of item 3 and 2 units of item 4.

EXAMPLE 1.4 - 10 (Selection of Advertising Media)

A cosmetics manufacturing company is interested in selecting the advertising media for its product and the frequency of advertising in each media. The data collected over the past two years regarding the frequency of advertising in three media of newspaper, radio and television and the related sales of the product give the following results :

TABLE 1.24
Expected sales in thousands of rupees

Frequency/week	Television	Radio	Newspaper
1	220	150	100
2	275	250	175
3	325	300	225
4	350	320	250

The cost of advertising in newspaper is ₹ 500 per appearance, while in radio and in television, it is ₹ 1,000 and ₹ 2,000 respectively. The budget provides ₹ 4,500 per week for advertisement. The problem is of determining the optimal combination of advertising media and advertising frequency.

Solution: The problem can be decomposed into three stages corresponding to the three media of advertising. In each media four alternatives (frequencies) are possible.

Each alternative, has associated with it certain cost and return (expected sales). Here again, the capital marked for allocation to different media is the state variable. A combination of media and frequency is to be selected in such a way as to maximize the total sales with expenditure not exceeding the specified limit of ₹ 4,500.

Let us consider the advertising media of television as the first stage. If x_1 is the capital allocated to stage 1, and $R_{1j}(C_{1j})$ is the return (expected sales) corresponding to cost C_{1j} , then, the optimal return is

$$f_1(x_1) = \max[R_{1j}(C_{1j})],$$

$$j = 0, 1, 2, 3, 4, \text{ with } 0 \leq x_1 \leq C.$$

By applying this equation at various levels of expenditure, the various alternatives are evaluated and the one giving the largest expected sales is selected. The selected frequencies and the optimal return for different values of x_1 are given in table 1.25.

TABLE 1.25

Stage 1

Cost per appearance = ₹ 2,000		
State x_1	Return (in thousands of rupees)	Frequency
500	—	0
1,000	—	0
1,500	—	0
2,000	220	1
2,500	220	1
3,000	220	1
3,500	220	1
4,000	275	2
4,500	275	2

For $x_1 = 0, ₹500, ₹1,000$ and $₹1,500$; it is not possible to advertise in this media, since the cost of one appearance per week is ₹2000. For $x_1 = ₹2,000, ₹2,500, ₹3,000$ and $₹3,500$, the product can be advertised only once, giving a return of ₹2,20,000.

With $x_1 = ₹4,000, ₹4,500$, two appearances can occur giving a return of ₹2,75,000.

TABLE 1.26
Stage 2
Cost per appearance = ₹1,000
Return in thousands of rupees

State x_2	0	1	2	3	4	Return	Frequency
500	0	0	—	—	—	0	0
1,000	0	150	—	—	—	150	1
1,500	0	150+0	—	—	—	150	1
2,000	220	150+0	250	—	—	250	2
2,500	220	150+0	250+0	—	—	250	2
3,000	220	150+220	250+0	300	—	370	1
3,500	220	150+220	250+0	300+0	—	370	1
4,000	275	150+220	250+220	300+0	320	470	2
4,500	275	150+220	250+220	300+0	320+0	470	2

Now, let us move to the second stage. Again for advertising in radio, four alternatives (frequencies) are possible. Here, the state x_2 will signify the expenditure incurred at the first stage and at the current stage.

At any value of state x_2 ($0 \leq x_2 \leq C$),

$$\begin{aligned}\text{Optimal return } f_2(x_2) &= \max[R_{2j}(C_{2j}) + f_1(x_1)], \text{ for } j = 0 \text{ to } 4 \\ &= \max[R_{2j}(C_{2j}) + f_1(x_2 - C_{2j})], \text{ for } j = 0 \text{ to } 4\end{aligned}$$

The evaluation of alternatives is carried in the tabular form shown in table 1.26. To illustrate, at $x_2 = 3,000$, four alternatives are possible, i.e., do not advertise, advertise once, twice or thrice. It is not possible to advertise four times because that needs a sum of ₹ 4,000. If we do not purchase any advertisement (frequency = 0), the amount of ₹ 3,000 can purchase one advertisement in media T, giving expected sales of ₹ 2,20,000. If one advertisement is purchased in media R, this will cost ₹ 1,000, and with amount of ₹ 2,000 left one advertisement can be purchased in media T, giving total return of ₹ $(150 + 220) \times 1,000 = ₹ 3,70,000$. If two advertisements are purchased in R, costing ₹ 2,000, the balance amount of ₹1,000 will be of no use in media T, and thus will give total sales as

₹ (250 + 0) × 1,000 = ₹ 2,50,000. The maximum return comes when we purchase one advertisement in media R. This is the optimal decision for $x_2 = ₹ 3,000$.

Now we move to the third stage.

$$f_3(x_3) = \max[R_{3j}(C_{3j}) + f_2(x_2)], \text{ for } j = 0 \text{ to } 4$$

$$= \max[R_{3j}(C_{3j}) + f_2(x_3 - C_{3j})], \text{ for } j = 0 \text{ to } 4.$$

TABLE 1.27

Stage 3

Cost per appearance = ₹ 500

Return in thousands of rupees

State x_3						<i>Optimal Decision</i>	
	0	1	2	3	4	Total sales	Frequency
500	0	100	—	—	—	100	1
1,000	150	100+0	175	—	—	175	2
1,500	150	100+150	175+0	225	—	250	1
2,000	250	100+150	175+150	225+0	250	325	2
2,500	250	100+250	175+150	225+150	250+0	375	3
3,000	370	100+250	175+250	225+150	250+150	425	2
3,500	370	100+370	175+250	225+250	250+150	475	3
4,000	470	100+370	175+370	225+250	250+250	545	2
4,500	470	100+470	175+370	225+370	250+250	595	3

The computations are given in table 1.27. For the allocated capital of ₹ 4,500, the maximum sales that can be expected are of ₹ 5,95,000. From table 1.27, the optimal decision is: purchase three advertisements in newspaper. This will cost ₹ 1,500. The amount left is ₹ 3,000, and corresponding to that at stage 2, the optimal decision is: purchase one advertisement in radio. This costs ₹ 1,000 which leaves behind an amount of ₹ 2,000 which can purchase one advertisement in television (stage 1).

Similarly, if the firm wants to spend only ₹ 4,000 per week, the optimal policy will be: purchase two advertisements in newspaper costing ₹ 1,000, one in radio costing

₹ 1,000, and one in television costing ₹ 2,000. This will give an optimal expected sales worth ₹ 5,45,000.

EXAMPLE 1.4 - 11

A man is engaged in buying and selling identical items. He operates from a warehouse that can hold 500 items. Each month he can sell any quantity that he chooses to stock at the beginning of the month. Each month, he can buy as much as he wishes for delivery at the end of the month, so long as his stock does not exceed 500 items. For the next four months he has the following error - free forecasts of cost and selling prices:

Month : 1 2 3 4

Cost : 27 26 24 28

Selling price : 28 25 25 27

If he currently has a stock of 200 units, what quantities should he sell and buy in the next four months ? Find the solution using dynamic programming.

Solution.

The problem can be analyzed by treating the four months as the four stages.

Let x_i be the number of items to be sold during the month i ,
 y_i be the number of items to be ordered during the month i ,
 b_i be the stock level in the beginning of month i ,
 p_i be the selling price in month i ,
 c_i be the purchase price in month i ,
 w be the warehouse capacity, which is 500.

The problem can be solved starting with the 4th month and then proceeding backward.

If $f_n(b_n)$ is the return when there are n more months to follow and the initial stock at the beginning of the month n is b_n , then n varies from 1 to 4 as the months vary from 4th to 1st.

n : 4 3 2 1

Month : I II III IV

The recursive equation can be written as

At stage 1:

$$f_1(b_n) = \max_{x_n, y_n} [p_1 x_n - c_1 y_n],$$

where $x_n \leq b_n$ and $b_n - x_n + y_n \leq W$.

For any other stage,

$$f_n(b_n) = \max_{x_n, y_n} [p_n x_n - c_n y_n - f_1(b_n - x_n + y_n)].$$

When $n = 1$,

$$f_1(b_1) = \max_{x_1, y_1} [p_1 x_1 - c_1 y_1]$$

Since c_1 is positive, to maximize $f_1(b_1)$, $y_1 = 0$; and since no stock should be left at the end of the 4th month, the amount to be sold during the month should be equal to be the amount at the beginning of the month, i.e. $x_1 = b_1$.

$$f_1(b_1) = p_1 b_1 = 27b_1.$$

When $n = 2$,

$$\begin{aligned} f_2(b_2) &= \max_{x_2, y_2} [p_2 x_2 - c_2 y_2 - f_1(b_1)] \\ &= \max_{x_2, y_2} [p_2 x_2 - c_2 y_2 - f_1(b_2 - x_2 + y_2)] \\ &= \max_{x_2, y_2} [25x_2 - 26y_2 + 27(b_2 - x_2 + y_2)] \\ &= \max_{x_2, y_2} [y_2 - 2x_2 + 27b_2]. \end{aligned}$$

Since

$$\begin{aligned} y_2 &\leq W - b_2 + x_2 \\ &\leq 500 - b_2 + x_2 \\ f_2(b_2) &= \max_{x_2, y_2} [500 - b_2 + x_2 - 2x_2 + 27b_2] \\ &= \max_{x_2} [26b_2 - x_2 + 500]. \end{aligned}$$

To maximize $f_2(b_2)$, x_2 can be taken as zero.

$$\therefore f_2(b_2) = 26b_2 + 500.$$

When $n = 3$,

$$\begin{aligned} f_3(b_3) &= \max_{x_3, y_3} [p_3 x_3 - c_3 y_3 - f_2(b_2)] \\ &= \max_{x_3, y_3} [25x_3 - 24y_3 + 26b_2 + 500] \end{aligned}$$

Now

$$\begin{aligned} b_2 &= b_3 - x_3 + y_3. \\ \therefore f_3(b_3) &= \max_{x_3, y_3} [25x_3 - 24y_3 + 26(b_3 - x_3 + y_3) + 500] \\ &= \max_{x_3, y_3} [26b_3 - x_3 + 2y_3 + 500], \end{aligned}$$

and

$$y_3 \leq 500 - b_3 + x_3.$$

$$\begin{aligned}\therefore f_3(b_3) &= \max_{x_3} [26b_3 - x_3 + 2(500 - b_3 + x_3) + 500] \\ &= \max_{x_3} [24b_3 + x_3 + 1,500].\end{aligned}$$

Now to maximize $f_3(b_3)$, x_3 should be maximum permissible, which is b_3 since $x_3 \leq b_3$.

$$\therefore f_3(b_3) = 25b_3 + 1,500.$$

When $n = 4$,

$$\begin{aligned}f_4(b_4) &= \max_{x_4, y_4} [p_4x_4 - c_4y_4 + f_3(b_3)] \\ &= \max_{x_4, y_4} [28x_4 - 27y_4 + 25b_3 + 1,500].\end{aligned}$$

But

$$\begin{aligned}b_3 &= b_4 - x_4 + y_4. \\ \therefore f_4(b_4) &= \max_{x_4, y_4} [28x_4 - 27y_4 + 25(b_4 - x_4 + y_4) + 1,500] \\ &= \max_{x_4, y_4} [25b_4 + 3x_4 - 2y_4 + 1,500].\end{aligned}$$

To maximize $f_4(b_4)$, y_4 should be zero as $y_4 \geq 0$, and x_4 which is $\leq b_4$, should at minimum be equal to b_4 .

$$\begin{aligned}\text{i.e.,} \quad y_4 &= 0 \text{ and } x_4 = b_4, \text{ which gives} \\ f_4(b_4) &= 28b_4 + 1,500.\end{aligned}$$

Now, it is given that stock level at the beginning of the first month is 200.
Thus $b_4 = 200$.

$$\begin{aligned}\therefore f_4(b_4) &= 28 \times 200 + 1,500 = 7,100, \\ x_4 &= 200, y_4 = 0, \\ b_3 &= b_4 - x_4 + y_4 = 0, \\ x_3 &= b_3 = 0, \\ y_3 &= 500 - b_3 + x_3 = 500, \\ b_2 &= b_3 - x_3 + y_3 = 500, \\ x_2 &= 0, y_2 = 500 - b_2 + x_2 = 0, \\ b_1 &= b_2 - x_2 + y_2 = 500, \\ x_1 &= b_1 = 500, \text{ and} \\ y_1 &= 0.\end{aligned}$$

The optimal policy can be expressed as

<i>Month</i> :	I	II	III	IV
<i>n</i> :	4	3	2	1
<i>Purchase</i> :	0	500	0	0
<i>Sale</i> :	200	0	0	500

1.5 OPTIMAL SUBDIVISION PROBLEM

This problem deals with the division of a given quantity into a given number of parts. Let Q be the quantity to be divided in n number of parts (u_1, u_2, \dots, u_n) . Then problem can be expressed as

$$\text{maximize } \prod_{i=1}^n u_i \text{ or maximize } u_1 \cdot u_2 \cdot u_3 \dots u_n,$$

$$\text{subject to } \sum_{i=1}^n u_i = Q,$$

$$u_i \geq 0, i = 1, 2, \dots, n.$$

The problem can be handled by dynamic programming, by considering each part as a stage. The alternatives at each stage are infinite, since u_i is continuous and may assume any non-negative value, satisfying the constraints

$$\sum_{i=1}^n u_i = Q.$$

The state of the system x_i , at any stage i , represents the part of resource Q , allocated to stage 1 through i inclusive. The reaction formula is then given as

$$f_1(x_1) = \max_{u_1 = x_1} \{u_1\}, \quad \dots (1.6)$$

$$\begin{aligned} f_i(x_i) &= \max_{0 \leq u_i \leq x_i} \{u_i f_{i-1}(x_i - u_i)\}, \\ &= \max_{u_i} \{u_i(x_i - u_i)\} \quad \dots (1.7) \end{aligned}$$

EXAMPLE 1.5 - 1

Determine the value of u_1, u_2, u_3 so as to maximize (u_1, u_2, u_3) , subject to $u_1 + u_2 + u_3 = 10$ and $u_1, u_2, u_3 \geq 0$.

Solution. In this example $Q = 10$, is to be divided into three parts u_1, u_2 and u_3 such that their product is maximum.

This Dynamic Programming problem can be regarded as a three-stage problem with state variables x_1, x_2, x_3 and returns $f_1(x_1), f_2(x_2)$ and $f_3(x_3)$ respectively.

At stage 3, $x_3 = u_1 + u_2 + u_3$,

at stage 2, $x_2 = x_3 - u_3 = u_1 + u_2$,

at stage 1, $x_1 = x_2 - u_2 = u_1$.

$$\begin{aligned}\therefore f_1(x_1) &= u_1 = x_2 - u_2, \\ f_2(x_2) &= \max\{u_2(x_2 - u_2)\}, 0 \leq u_2 \leq x_2, \\ &= \max\{u_2x_2 - u_2^2\}, 0 \leq u_2 \leq x_2.\end{aligned}$$

Differentiating w.r.t. u_2 , and equating the differential to zero,

$$\frac{\partial f_2(x_2)}{\partial u_2} = x_2 - 2u_2 = 0 \quad \text{or} \quad u_2 = \frac{x_2}{2}$$

$$\therefore f_2(x_2) = \frac{x_2}{2} \left(x_2 - \frac{x_2}{2} \right).$$

$$\text{Now} \quad f_2(x_2) = \max_{u_3} \left\{ u_3 \cdot \frac{x_2^2}{2} \right\} = \max_{u_3} \left\{ u_3 \cdot \frac{(x_3 - u_3)^2}{2} \right\}.$$

Differentiating w.r.t. u_3 , and equating the differential to zero,

$$\frac{\partial}{\partial u_3} \left[\frac{(u_3x_3^2 + u_3^3 - 2u_3^2x_3)}{4} \right] = 0$$

$$\text{or} \quad x_3^2 + 3u_3^2 - 4u_3x_3 = 0$$

$$\text{or} \quad x_3^2 - 3u_3x_3 + 3u_3^2 - u_3x_3 = 0$$

$$\text{or} \quad x_3(x_3 - 3u_3) - u_3(x_3 - 3u_3) = 0$$

$$\text{or} \quad (x_3 - u_3)(x_3 - 3u_3) = 0.$$

\therefore Either $u_3 = x_3$ which is trivial since

$$x_3 = u_1 + u_2 + u_3;$$

$$\text{or } u_3 = \frac{x_3}{3} = \frac{10}{3}$$

$$\therefore x_2 = 10 - \frac{10}{3} = \frac{20}{3},$$

$$\text{and } u_2 = \frac{x_2}{2} = \frac{10}{3} \text{ and hence } u_1 = \frac{10}{3}.$$

$$\therefore u_1 = u_2 = u_3 = \frac{10}{3},$$

$$\text{and maximum product} = u_1 \cdot u_2 \cdot u_3 = \frac{1,000}{27}.$$

EXAMPLE 1.5 - 2

Let us consider the general case of dividing Q into n parts u_1, u_2, \dots, u_n so as to maximize,

$$\prod_{i=1}^n u_i, u_i \geq 0.$$

Solution.

Using the recursive equations (1.6) and (1.7),

$$f_1(x_1) = \max_{u_1} \{u_1\}$$

$$f_i(x_i) = \max_{0 \leq u_i \leq x_i} \{u_i f_{i-1}(x_i - u_i)\}$$

$$\text{For } i = 1, \quad f_1(x_1) = x_1 \text{ which means } u_1^* = x_1.$$

$$\text{For } i = 2, \quad f_2(x_2) = \max_{0 \leq u_2 \leq x_2} \{u_2 \cdot f_1(x_2 - u_2)\}$$

$$= \max\{u_2 \cdot f_1(x_2 - u_2)\},$$

$$= \{u_2 \cdot (x_2 - u_2)\}, \text{ since } f_1(x_2 - u_2) = x_2 - u_2.$$

Differentiating w.r.t. u_2 and equating to zero,

$$\frac{\partial}{\partial u_2} (u_2 x_2 - u_2^2) = 0$$

or $x_2 - 2u_2 = 0$ or $u_2^* = \frac{x_2}{2}$

$$f_2(x_2) = \frac{x_2}{2} \left(x_2 - \frac{x_2}{2} \right) = \left(\frac{x_2}{2} \right)^2.$$

It can be shown that the second derivative is negative, which is a sufficient condition for maxima.

Since $x_2 = u_1 + u_2$,

$$u_1 = x_2 - u_2 = x_2 - \frac{x_2}{2} = \frac{x_2}{2}.$$

Thus for a two-stage problem,

$$u_1 = u_2 = \frac{x_2}{2}.$$

For $i = 3$, $f_3(x_3) = \max_{u_3} \{u_3 \cdot f_2(x_3 - u_3)\}$

$$= \max \left\{ u_3 \cdot \left(\frac{x_3 - u_3}{2} \right)^2 \right\} \text{ since } f_2(x_2) = \left(\frac{x_2}{2} \right)^2.$$

Differentiating w.r.t. u_3 , and equating to zero,

$$\frac{\partial}{\partial u_3} \left[\frac{u_3 x_3^2 + u_3^3 - 2u_3^2 x_3}{4} \right] = 0$$

or $x_3^2 + 3u_3^2 - 4u_3 x_3 = 0$,

which gives either $u_3 = x_3$ or $u_3 = \frac{x_3}{3}$.

Since only $u_3 = \frac{x_3}{3}$ satisfies the sufficiency condition for a maxima,

(also $u_3 \neq x_3$ since $u_1 + u_2 + u_3 = x_3$)

$$f_3(x_3) = \frac{x_3}{3} \frac{\left(x_3 - \frac{x_3}{3} \right)^2}{4} = \left(\frac{x_3}{3} \right)^3.$$

$\therefore u_3^* = \frac{x_3}{3}.$

Since $x_3 = u_1 + u_2 + u_3$,

$$x_2 = u_1 + u_2 = x_3 - u_3 = x_3 - \frac{x_3}{3} = \frac{2}{3} x_3.$$

But $u_1 = u_2 = \frac{x_2}{2} = \frac{x_3}{3}.$

∴ For a three-stage problem, $u_1 = u_2 = u_3 = \frac{x_3}{3}$ with $f_3(x_3) = \left(\frac{x_3}{3}\right)^3$.

For an n - stage problem,

$$u_1 = u_2 = u_3 \dots = u_n = \frac{x_n}{n} \text{ with } f_n(x_n) = \left(\frac{x_n}{n}\right)^n.$$

But $x_n = Q$

$$\therefore u_1 = u_2 = u_3 \dots = u_n = \frac{Q}{n} \text{ with } f_n(Q) = \left(\frac{Q}{n}\right)^n.$$

Remark:

Since
$$f_n(Q) = \left(\frac{Q}{n}\right)^n$$

$$= \left(\frac{\sum_{l=1}^n u_l}{n}\right)^n,$$

and

$$f_n(Q) = \max\{u_1, u_2, u_3 \dots u_n\}$$

$$= \max_{u_l} \left[\prod_{l=1}^n u_l \right] \geq \prod_{l=1}^n u_l,$$

$$\frac{\sum_{l=1}^n u_l}{n} \geq \left(\prod_{l=1}^n u_l \right)^{1/n}$$

∴ *Arithmetic mean* \geq *Geometric mean*.

This inequality is known as Cauchy's geometric-arithmetic mean inequality, which states that the arithmetic mean of ' n ' numbers is always greater than their geometric mean except when all the numbers are equal.

EXAMPLE 1.5 - 3

$$\begin{aligned} \text{Minimize } Z &= y_1^2 + y_2^2 + y_3^2, \\ \text{subject to } y_1 + y_2 + y_3 &\geq 15, \\ y_1, y_2, y_3 &\geq 0. \end{aligned}$$

Solution.

Let the state variable be x_1 , x_2 and x_3 such that,

$$x_3 = y_1 + y_2 + y_3,$$

$$x_2 = x_3 - y_3 = y_1 + y_2, \text{ and}$$

$$x_1 = x_2 - y_2 = y_1.$$

The recursive equations are

$$f_3(x_3) = \min_{y_3} \{y_3^2 + f_2(x_2)\}$$

$$f_2(x_2) = \min_{y_2} \{y_2^2 + f_1(x_1)\}, \text{ and}$$

$$f_1^*(x_1) = \min_{y_1} \{y_1^2\} = y_1^2.$$

Since $x_1 = x_2 - y_2$ and $f_1(x_1) = y_1^2$,

$$f_2(x_2) = \min_{y_2} \{y_2^2 + (x_2 - y_2)^2\}.$$

Differentiating $\{y_2^2 + (x_2 - y_2)^2\}$ w.r.t. y_2 and equating to zero,

$$2y_2 + 2(x_2 - y_2)(-1) = 0$$

$$\text{or } -2x_2 + 4y_2 = 0$$

$$\text{or } y_2 = \frac{x_2}{2}.$$

$$\therefore f_2^*(x_2) = \left(\frac{x_2}{2}\right)^2 + \left(x_2 - \frac{x_2}{2}\right)^2 = \frac{x_2^2}{2}.$$

$$\text{Now } f_3(x_3) = \min_{y_3} \{y_3^2 + f_2(x_2)\}.$$

$$\text{Since } x_2 = x_3 - y_3 \text{ and } f_2(x_2) = \frac{x_2^2}{2},$$

$$f_3(x_3) = \min_{y_3} \left\{ y_3^2 + \frac{(x_3 - y_3)^2}{2} \right\}.$$

Differentiating $\left\{ y_3^2 + \frac{(x_3 - y_3)^2}{2} \right\}$ w.r.t. y_3 and equating to zero,

$$2y_3 - (x_3 - y_3) = 0$$

$$\text{or } y_3 = \frac{x_3}{3}.$$

$$\therefore f_3^*(x_3) = \left(\frac{x_3}{3}\right)^2 + \frac{\left(x_3 - \frac{x_3}{3}\right)^2}{2} = \frac{x_3^2}{3}.$$

Since $x_3 = y_1 + y_2 + y_3 \geq 15$, for minimization of $f_3(x_3)$, $y_1 + y_2 + y_3 = 15$
or $x_3 = 15$.

$$\therefore f_3^*(x_3) = \frac{(15)^2}{3} = 75,$$

and

$$y_3 = \frac{x_3}{3} = 5,$$

$$y_2 = \frac{x_2}{2} = \frac{x_3 - y_3}{2} = \frac{15 - 5}{2} = 5,$$

$$y_1 = x_2 - y_2 = 10 - 5 = 5.$$

Thus minimum value of $y_1^2 + y_2^2 + y_3^2 = 75$ with $y_1 = y_2 = y_3 = 5$.

EXAMPLE 1.5 - 4

Minimize $y_1^2 + y_2^2 + y_3^2$,

subject to $y_1 + y_2 + y_3 = 10$,

when (a) y_1, y_2, y_3 are non-negative,

(b) y_1, y_2, y_3 are non-negative integers.

Solution.

(a) When y_1, y_2, y_3 are continuous non-negative variables, the solution can be obtained in the same way as in example 1.5 - 3.

Minimum value of $y_1^2 + y_2^2 + y_3^2 = \frac{(10)^2}{3} = \frac{100}{3}$,

with $y_1 = y_2 = y_3 = \frac{10}{3}$

(b) When the variables y_1, y_2, y_3 are non-negative integers, the problem can conveniently be solved by the tabular or enumeration method, treating it as a three-stage problem.

At stage 1, the state variables x_1 can take any integer value from 0 to 10, with return,

$$f_1(x_1) = \underset{0 \leq y_1 \leq 10}{\text{minimum}} \{y_1^2\} = y_1^2$$

At stage 2,

$$f_2(x_2) = \underset{0 \leq y_2 \leq 10}{\text{minimum}} \{y_2^2 + f_1^*(x_2 - y_2)^2\}.$$

The computations for 2nd stage are shown below, where the minimum $f_2(x_2)$ values are identified by *.

$y_1 :$	0	1	2	3	4	5	6	7	8	9	10	
$y_1^2 :$	0	1	4	9	16	25	36	49	64	81	100	
y_2	y_2^2											
0	0	0*	1*	4	9	16	25	36	49	64	81	100
1	1	1*	2*	5*	10	17	26	37	50	65	82	
2	4	4	5*	8*	13*	20	29	40	53	68		
3	9	9	10	13*	18*	25*	34	45	58			
4	16	16	17	20	25*	32*	41*	52				
5	25	25	26	29	34	41*	50*					
6	36	36	37	40	45	52						
7	49	49	50	53	58							
8	64	64	65	68								
9	81	81	82									
10	100	100										

The optimal values of $f_2(x_2)$ are

$$y_2 : 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10$$

$$f_2^*(x_2) : 0 \quad 1 \quad 2 \quad 5 \quad 8 \quad 13 \quad 18 \quad 25 \quad 32 \quad 41 \quad 50$$

At stage 3,

$$f_3(x_3) = \min \{y_3^2 + f_2^*(x_2)\}$$

$$= \min \{y_3^2 + f_2^*(x_3 - y_3)^2\}$$

$y_3 :$	0	1	2	3	4	5	6	7	8	9	10
$x_3 - y_3 :$	10	9	8	7	6	5	4	3	2	1	0
$y_3^2 :$	0	1	4	9	16	25	36	49	64	81	100
$(x_3 - y_3)^2 :$	50	41	32	25	18	13	8	5	2	1	0
$f_3(x_3) :$	50	42	36	34*	34*	38	44	54	66	82	100

$$f_3^*(x_3) = 34 \text{ for } y_3^* = 3 \text{ or } 4$$

For $y_3^* = 3$, $f_2^*(x_2) = 25$, for which $y_1 = 3$ and $y_2 = 4$
or $y_1 = 4$ and $y_2 = 3$

For $y_3^* = 4$, $f_2^*(x_2) = 18$, for which $y_1 = 3, y_2 = 3$.

Thus minimum value of 34 corresponds to

$$(y_1, y_2, y_3) = (3, 3, 4), (3, 4, 3), (4, 3, 3).$$

EXAMPLE 1.5 - 5

Using Bellman's principle of optimality,

$$\text{minimize } Z = y_1 + y_2 + y_3 + \dots + y_n,$$

$$\text{Subject to } y_1 \cdot y_2 \cdot y_3 \dots y_n = b,$$

$$y_i \geq 0; i = 1, 2, \dots, n.$$

Solution.

This problem comprises of factorizing a constant b into n factors, such that the sum of the factors is minimum. It can be treated as n -stage dynamic programming problem.

Let $f_n(b)$ be the minimum attainable value of

$$\sum_{i=1}^n y_i,$$

when $y_1, y_2, y_3, \dots, y_n$ are factors of b .

Considering the problem as single - stage, i.e., $n = 1$,

$$f_1(b) = b \text{ or}$$

$$f_1(b) = \min_{y_1=b} \{y_1\} = b\{y_1\}$$

Considering the problem as two - stage, i.e., $n = 2$ and factorizing b into two factors y_1 and y_2 ,

$$y_1 \cdot y_2 = b$$

$$\text{If } y_2 = x, \quad y_1 = \frac{b}{x}$$

$$\text{and } f_2(b) = \min \{y_1 + y_2\} = \min \{y_2 + y_1\}$$

$$= \min_{0 \leq x \leq b} \left\{ x + \frac{b}{x} \right\}$$

or
$$f_2(b) = \min_{0 \leq x \leq b} \left\{ x + f_1^* \left(\frac{b}{x} \right) \right\}, \text{ since } f_1 \left(\frac{b}{x} \right) = \frac{b}{x}.$$

Next consider $n = 3$ with $y_1 \cdot y_2 \cdot y_3 = b$.

If
$$y_3 = x, \quad y_1 \cdot y_2 = \frac{b}{x} \text{ and}$$

$$\begin{aligned} f_3(b) &= \min\{y_1 + y_2 + y_3\} = \min\{y_3 + y_1 + y_2\} \\ &= \min \left\{ x + f_2^* \left(\frac{b}{x} \right) \right\}. \end{aligned}$$

In the general form, for an i -stage problem,

$$f_i(b) = \min \left\{ x + f_{i-1}^* \left(\frac{b}{x} \right) \right\},$$

where
$$y_i = x \text{ and } y_1 \cdot y_2 \cdot y_3 \cdots y_{i-1} = \frac{b}{x}.$$

Now the optimal policy can be determined.

$$\begin{aligned} f_1^*(b) &= b, \quad f_2(b) = \min \left\{ x + f_1^* \left(\frac{b}{x} \right) \right\} \\ &= \min_{0 \leq x \leq b} \left\{ x + \frac{b}{x} \right\}, \text{ since } f_1^*(b) = b. \end{aligned}$$

Differentiating $\left\{ x + \frac{b}{x} \right\}$ w.r.t. x and equating to zero,

$$1 - \frac{b}{x^2} = 0 \text{ or } x = b^{1/2}.$$

$$\therefore f_2^*(b) = b^{1/2} + \frac{b}{b^{1/2}} = 2b^{1/2}.$$

Now
$$\begin{aligned} f_3(b) &= \min_{0 \leq x \leq b} \left\{ x + f_2^* \left(\frac{b}{x} \right) \right\} \\ &= \min_{0 \leq x \leq b} \left\{ x + 2 \left(\frac{b}{x} \right)^{1/2} \right\}, \text{ since } f_2^*(b) = 2b^{1/2} \end{aligned}$$

Differentiating $\left\{ x + 2 \left(\frac{b}{x} \right)^{1/2} \right\}$ w.r.to x and equating to zero,

$$1 - b^{1/2} x^{-3/2} = 0 \text{ or } x = b^{2/3}.$$

$$\therefore f_3^*(b) = b^{1/3} + 2 \left(\frac{b}{b^{1/3}} \right)^{\frac{1}{2}} = 3b^{1/3} \text{ and so on.}$$

For n -stage problem,

$$f_n^*(b) = nb^{\frac{1}{n}} \text{ and } x = b^{\frac{1}{n}},$$

Hence the optimal solution is

$$y_1 = y_2 = y_3 = \dots = y_n = b^{\frac{1}{n}},$$

and $f_n^*(b) = nb^{\frac{1}{n}}.$

EXAMPLE 1.5 – 6

Solve the following problem :

$$\begin{aligned} \text{Minimize } Z &= \sum_{i=1}^n y_i^2 \\ \text{subject to } \prod_{i=1}^n y_i &= b, \\ y_i &\geq 0; i = 1, 2, \dots, n \end{aligned}$$

Solution: This problem can be treated as n -stage dynamic programming problem. Let $f_n(b)$ be the minimum attainable value of Z , when b is factorized into n parts.

For $n = 1$, $y_1 = b$,

and hence $f_1^*(b) = \min_{y_1=b} [y_1^2] = b^2.$

For $n = 2$, $y_1 y_2 = b$.

Let $y_2 = x$, $y_1 = b/x$.

Then
$$\begin{aligned} f_2(b) &= \min[y_1^2 + y_2^2] = \min[y_2^2 + y_1^2] \\ &= \min_{0 \leq x \leq b} \left[x^2 + \left(\frac{b}{x} \right)^2 \right] \end{aligned}$$

or
$$f_1^*(b) = \min_{0 \leq x \leq b} \left[x^2 + f_1^* \left(\frac{b}{x} \right) \right], \text{ since } f_1(b) = b^2.$$

For $n = 3$, $y_1 \cdot y_2 \cdot y_3 = b$.

Let $y_3 = x$, $y_1 y_2 = \frac{b}{x}$.

Then
$$f_3(b) = \min[y_1^2 + y_2^2 + y_3^2] = \min[y_3^2 + y_1^2 + y_2^2]$$
$$= \min_{0 \leq x \leq b} \left[x^2 + f_2^* \left(\frac{b}{x} \right) \right].$$

Thus for i -stage problem,

$$f_i^*(b) = \min_{0 \leq x \leq b} \left[x^2 + f_{i-1}^* \left(\frac{b}{x} \right) \right].$$

The optimal policy can now be determined.

$$f_1^*(b) = b^2,$$
$$f_2(b) = \min_{0 \leq x \leq b} [x^2 + f_1(b)]$$
$$= \min_{0 \leq x \leq b} \left[x^2 + \left(\frac{b}{x} \right)^2 \right], \quad \text{since } f_1(b) = b^2.$$

Differentiating $\left\{ x^2 + \left(\frac{b}{x} \right)^2 \right\}$ w.r.t. x and equating to zero,

$$2x - \frac{2b}{x^3} = 0 \quad \text{or} \quad x^4 = b^2$$

or $x = b^{\frac{1}{2}} \quad \therefore y_1 = y_2 = b^{\frac{1}{2}},$

and
$$f_2^*(b) = \left(b^{\frac{1}{2}} \right)^2 + \left(\frac{b}{b^{\frac{1}{2}}} \right)^2 = 2 \left(b^{\frac{1}{2}} \right)^2.$$

Now,
$$f_3(b) = \min_{0 \leq x \leq b} \left[x^2 + f_2 \left(\frac{b}{x} \right) \right]$$
$$= \min_{0 \leq x \leq b} \left[x^2 + 2 \frac{b}{x} \right], \quad \text{since } f_2(b) = 2b.$$

Differentiating $\left\{ x^2 + \frac{2b}{x} \right\}$ w.r.t. x and equating to zero,

$$2x - \frac{2b}{x^2} = 0 \quad \text{or} \quad x = b^{\frac{1}{3}}.$$

$$\therefore y_3 = x = b^{\frac{1}{3}} \quad \text{and} \quad y_1 \cdot y_2 = \frac{b}{x} = b^{\frac{2}{3}}.$$

$$\therefore y_1 = y_2 = b^{\frac{1}{3}} \quad \text{since } y_1 = y_2$$

and
$$f_3^*(b) = \left(b^{\frac{1}{3}} \right)^2 + 2 \frac{b}{b^{\frac{1}{3}}} = 3b^{\frac{2}{3}}.$$

Continuing in the same way,

$$f_n^*(b) = nb^{\frac{2}{n}},$$

and $y_1 = y_2 = y_3 = \dots = y_n = b^{\frac{1}{n}}.$

EXAMPLE 1.5 - 7

Solve the following problem :

$$\begin{aligned} \text{Maximize } Z &= u_1^2 + u_2^2 + u_3^2, \\ \text{subject to } &u_1 \cdot u_2 \cdot u_3 = 6, \\ &u_1, u_2, u_3 \text{ all positive integers.} \end{aligned}$$

Solution.

It can be treated as a three-stage problem.

Let $x_3 = u_1 \cdot u_2 \cdot u_3 = 6,$
 $x_2 = u_1 \cdot u_2 = \frac{6}{u_3},$ and
 $x_1 = u_1 = \frac{x_2}{u_2}.$

Stage 1: $x_1 = u_1$, where u_1 can vary from 1 to 6.

$$\therefore f_1(x_1) = \max_{0 \leq x_1 \leq 6} \{x_1^2\} = \max_{0 \leq u_1 \leq 6} \{u_1^2\} \text{ with } u_1 \text{ integer.}$$

$u_1 :$	1	2	3	4	5	6
$f_1^*(x_1) :$	1	4	9	16	25	36

Stage 2:

$$x_2 = u_1 u_2 = 6 \text{ with } u_1, u_2 \text{ integers.}$$

$$u_1 = 1 \ 2 \ 3 \ 4 \ 5 \ 6$$

$$u_2 = \frac{6}{u_1} = 6 \ 3 \ 2 \ - \ - \ 1$$

$$\therefore f_2(x_2) = \max[u_2^2 + f_1^*(x_1)] = \max \left[u_2^2 + \left(\frac{x_2}{u_2} \right)^2 \right].$$

$u_1 :$		1	2	3	6
$f_1(x_1) :$		1	4	9	36
u_2	u_2^2				
1	1	2*	5*	10*	37*
2	4	5*	8	13	
3	9	10*	13		
6	36	37*			

∴ The optimal values of $f_2(x_2)$ are

$x_2 :$	1	2	3	6
$f_2^*(x_2) :$	2	5	10	37

Stage 3:

$$x_3 = u_1 u_2 u_3 = 6,$$

$$u_3 = \frac{6}{u_1 u_2} = \frac{6}{x_2}.$$

$$\therefore f_2(x_2) = \max \left[u_3^2 + \left(\frac{6}{x_2} \right)^2 \right] = \max [u_3^2 + f_2^*(x_2)].$$

$x_2 :$		1	2	3	6
$f_2^*(x_2) :$		2	5	10	37
u_3	u_3^2				
1	1	3*	6*	11*	38*
2	4	6	9	14	
3	9	11*	14		
6	36	38*			

∴ The optimal values of $f_3(x_3)$ are

$x_3 :$	1	2	3	6
$f_3^*(x_3) :$	3	6	11	38

For $x_3 = 6$, optimal $Z = 38$ with $u_3 = 1$ or 6 .

Proceeding backwards, following sets of values for u_1, u_2, u_3 can be traced:

$(u_1, u_2, u_3) = (1, 6, 1); (1, 1, 6); (6, 1, 1)$.

EXAMPLE 1.5 - 8

Use dynamic programming to show that

$$\sum_{i=1}^n p_i \log p_i$$

subject to the constraint,

$$\sum_{i=1}^n p_i \log p_i = 1, \quad p_i \geq 0$$

for all i is minimum when $p_1 = p_2 = \dots = p_n = \frac{1}{n}$.

Solution.

This problem can be treated as an n -stage dynamic programming problem.

Let $f_n(1)$ be the minimum attainable value of $\sum p_i \log p_i$.

When $n = 1$, $p_1 = 1$.

$$\therefore f_1^*(1) = \min [p_1 \log p_1] = 1 \log 1.$$

When $n = 2$, $p_1 + p_2 = 1$.

$$\begin{aligned} \therefore f_2(1) &= \min [p_1 \log p_1 + p_2 \log p_2] \\ &= \min [p_2 \log p_2 + p_1 \log p_1]. \end{aligned}$$

If $p_2 = z$, $p_1 = 1 - z$.

$$\begin{aligned} \therefore f_2(1) &= \min_{0 \leq z \leq 1} [z \log z + (1 - z) \log(1 - z)] \\ &= \min_{0 \leq z \leq 1} [z \log z + f_1(1 - z)], \end{aligned}$$

since $f_1(1 - z) = (1 - z) \log(1 - z)$.

By simple calculus, it can be shown that the function $f(z) = z \log z + (1 - z) \log(1 - z)$ is minimum for $z = 1/2$. Thus for a two-stage problem,

$$p_1 = p_2 = \frac{1}{2},$$

and $f_2^*(1) = \frac{1}{2} \log \frac{1}{2} + \frac{1}{2} \log \frac{1}{2} = 2 \left(\frac{1}{2} \log \frac{1}{2} \right).$

Next, when $n = 3$, $p_1 + p_2 + p_3 = 1$.

If $p_3 = z$, $p_1 + p_2 = 1 - z$.

$$\begin{aligned} f_2(1) &= \min [p_3 \log p_3 + p_2 \log p_2 + p_1 \log p_1] \\ &= \min_{0 \leq z \leq 1} [z \log z + f_2(1 - z)]. \end{aligned}$$

Since $f_2(1) = 2 \left(\frac{1}{2} \log \frac{1}{2} \right),$

$$f_2(1 - z) = 2 \frac{1-z}{2} \log \frac{1-z}{2}.$$

$$\therefore f_3(1) = \min_{0 \leq z \leq 1} [z \log z + 2 \left(\frac{1-z}{2} \right) \log \left(\frac{1-z}{2} \right)].$$

Again by differential calculus, it can be shown that

$f(z) = z \log z + 2 \left(\frac{1-z}{2} \right) \log \left(\frac{1-z}{2} \right)$ is minimum for $z = \frac{1}{3}$, which gives

$$p_1 = p_2 = p_3 = \frac{1}{3},$$

and $f_3^*(1) = \frac{1}{3} \log \frac{1}{3} + 2 \left(\frac{1}{3} \right) \log \left(\frac{1}{3} \right) = 3 \left(\frac{1}{3} \log \frac{1}{3} \right).$

For n -stage problem, the result can be generalized as

$$p_1 = p_2 = \dots = p_n = \frac{1}{n},$$

and $f_n^*(1) = n \left(\frac{1}{n} \log \frac{1}{n} \right).$

EXAMPLE 1.5 - 9

$$\text{Maximize } Z = \sum_{i=1}^n b_i x_i,$$

$$\text{subject to } \sum_{i=1}^n x_i = c; \quad x_i \geq 0, i = 1, 2, \dots, n.$$

Solution.

Let $f_n(c)$ be the return function.

For $n = 1$, i.e., a single stage,

$$x_1 = c.$$

$$\therefore f_1^*(c) = \max_{x_1=c} [b_1 x_1] = b_1 c.$$

For $n = 2$, $x_1 + x_2 = c.$

$$\therefore f_2(c) = \max [b_1 x_1 + b_2 x_2].$$

Let $x_2 = z$ and $x_1 = c - z.$

$$\begin{aligned} \therefore f_2(c) &= \max_{0 \leq z \leq c} [b_2 z + b_1(c - z)] \\ &= \max_{0 \leq z \leq c} [b_2 z + f_1(c - z)], \text{ because } f_1(c) = b_1 c. \end{aligned}$$

For $n = i$, the general recursive equation is

$$f_i(c) = \max_{0 \leq z \leq c} [b_i z + f_i(c - z)].$$

Again, when $n = 2$,

$$\begin{aligned} f_2(c) &= \max_{0 \leq z \leq c} [b_2 z + b_1(c - z)] \\ &= \max_{0 \leq z \leq c} [(b_2 - b_1)z + b_1 c]. \end{aligned}$$

The function $\{(b_2 - b_1)z + b_1 c\}$ will be maximum when z takes its maximum value c and $(b_2 - b_1) > 0.$

$$\therefore f_2^*(c) = [(b_2 - b_1)c + b_1 c] = b_2 c,$$

and $x_2 = z = c, x_1 = 0.$

When $n = 3$, $x_1 + x_2 + x_3 = c.$

Taking $x_3 = z, x_1 + x_2 = c - z,$

$$\begin{aligned} f_3(c) &= \max_{0 \leq z \leq c} [b_3 z + f_2^*(c - z)] \\ &= \max_{0 \leq z \leq c} [b_3 z + b_2(c - z)], \text{ because } f_2^*(c) = b_2 c \\ &= \max_{0 \leq z \leq c} [(b_3 - b_2)z + b_2 c]. \end{aligned}$$

The function $f_3(c)$ will be maximum when z takes its maximum value of c and $(b_3 - b_2) > 0$, which gives

$$f_3^*(c) = b_3 c,$$

$$x_3 = z = c \text{ and } x_1 = x_2 = 0.$$

Generalizing the result for n -stage problem,

$$f_n(c) = b_n c,$$

$$x_n = c \text{ and } x_1 = x_2 = x_3 = \dots = x_n = 0.$$

Thus the optimal policy is

$$(x_1, x_2, x_3, \dots, x_{n-1}, x_n) = (0, 0, 0, \dots, 0, c) \text{ with } f_n(c) = b_n c.$$

1.6 SYSTEM RELIABILITY

EXAMPLE 1.6 – 1

An electronic device consists of four components, each of which must function for the system to function. The system reliability can be improved by installing parallel units in one or more of the components. The reliability (R) of a component with one, two or three parallel units and the corresponding cost (C) are given in table 1.28. The maximum amount available for this device is 100. The problem is to determine the number of parallel units in each component.

TABLE 1.28

Number of units	Components							
	1		2		3		4	
	R	C	R	C	R	C	R	C
1	0.70	10	0.50	20	0.70	10	0.60	20
2	0.80	20	0.70	40	0.90	30	0.70	30
3	0.90	30	0.80	50	0.95	40	0.90	40

Solution.

The reliability of a system is the product of the reliability of its components. If $R_i u_i$ is the reliability of component having u_i units in parallel, then the reliability of the system comprising of n components in series is

$$\prod_{i=1}^n R_i u_i.$$

The problem, then, becomes,

$$\begin{aligned} \text{maximize } R &= \prod_{i=1}^n R_i u_i, \\ \text{subject to } \sum_{i=1}^n C_i u_i &\leq C, \end{aligned}$$

where $C_i u_i$ is the cost of components, when it has u_i units in parallel, and C is the total capital available. The problem can be solved by considering the components as stages and the capital allocated as state of the system, x_i . The state x_i ($0 \leq x_i \leq C$) is the capital allocated to stages 1 through i , inclusive. The reliability of the components of the return function at stage i may be expressed as $f_i(x_i)$.

The recursive equations can be written as

$$f_1(x_1) = \max_{u_1} \{R_1 u_1\}, \quad 0 \leq C_1 u_1 \leq x_1;$$

$$f_i(x_i) = \max_{u_i} \{R_i u_i \times f_{i-1}(x_i - C_i u_i)\}, \quad 0 \leq C_i u_i \leq x_i.$$

Since the return functions of different components are multiplied by each other, the procedure is called *multiplicative decomposition*.

In the given example, device will consist of at least one unit in each component.

$$\begin{aligned} \therefore C_{11} &\leq x_1 \leq C - C_{21} - C_{31} - C_{41} \quad \text{or} \quad 10 \leq x_1 \leq 50, \\ C_{11} + C_{21} &\leq x_2 \leq C - C_{31} - C_{41} \quad \text{or} \quad 30 \leq x_2 \leq 70, \\ C_{11} + C_{21} + C_{31} &\leq x_3 \leq C - C_{41} \quad \text{or} \quad 40 \leq x_3 \leq 80, \\ C_{11} + C_{21} + C_{31} + C_{41} &\leq x_4 \leq C \quad \text{or} \quad 60 \leq x_4 \leq 100. \end{aligned}$$

The computations for different stages are given below in the tabular form.

TABLE 1.29

Stage 1

x_i	$f_1(u_1/x_1) = R_1 u_1$			<i>Optimal Solution</i>	
	$u_1 = 1$ $R = 0.7, C = 10$	$u_1 = 2$ $R = 0.8, C = 20$	$u_1 = 3$ $R = 0.9, C = 30$	$f_1(x_1)$	u_1^*
10	0.7	—	—	0.7	1
20	0.7	0.8	—	0.8	2
30	0.7	0.8	0.9	0.9	3
40	0.7	0.8	0.9	0.9	3
50	0.7	0.8	0.9	0.9	3

Stage 2

x_1	$f_2(u_2/x_2) = R_2, u_2, f_1^*(x_2 - C_2, u_2)$			Optimal Solution	
	$u_2 = 1$ $R = 0.5, C = 20$	$u_2 = 2$ $R = 0.7, C = 40$	$u_2 = 3$ $R = 0.8, C = 50$	$f_2(x_2)$	u_2^*
30	0.5×0.7	—	—	0.35	1
40	0.5×0.8	—	—	0.40	1
50	0.5×0.9	0.7×0.7	—	0.49	2
60	0.5×0.9	0.7×0.8	0.8×0.7	0.56	2,3
70	0.5×0.9	0.7×0.9	0.8×0.8	0.64	3

Stage 3

x_i	$f_3(u_3/x_3) = R_3, u_3 \times f_2^*(x_3 - C_3, u_3)$			Optimal Solution	
	$u_3 = 1$ $R = 0.7, C = 10$	$u_3 = 2$ $R = 0.9, C = 30$	$u_3 = 3$ $R = 0.95, C = 40$	$f_3(x_3)$	u_3^*
40	$.7 \times .35 = .245$	—	—	.245	1
50	$.7 \times .40 = .280$	—	—	.280	1
60	$.7 \times .49 = .343$	$.9 \times .35 = .315$	—	.343	1
70	$.7 \times .56 = .392$	$.9 \times .40 = .360$	$.95 \times .35 = .3325$.392	1
80	$.7 \times .64 = .448$	$.9 \times .49 = .441$	$.95 \times .40 = .380$.448	1

Stage 4

x_i	$f_4(u_4/x_4) = R_4, u_4 \times f_3^*(x_4 - C_4, u_4)$			Optimal Solution	
	$u_4 = 1$ $R = 0.6, C = 20$	$u_4 = 2$ $R = 0.7, C = 30$	$u_4 = 3$ $R = 0.9, C = 40$	$f_4(x_4)$	u_4^*
60	$.6 \times .245 = .147$	—	—	.147	1
70	$.6 \times .280 = .168$	$.7 \times .245 = .1715$	—	.1715	2
80	$.6 \times .343 = .2058$	$.7 \times .280 = .196$	$.9 \times .245 = .2205$.2205	3
90	$.6 \times .392 = .2352$	$.7 \times .343 = .2401$	$.9 \times .280 = .252$.252	1
100	$.6 \times .448 = .2688$	$.7 \times .392 = .2744$	$.9 \times .343 = .3087$.3087	3

Optimal value of $f_4(x_4) = 0.3087$ with $u_4^* = 3$ and $x_4 = 100$, is obtained from $f_3(x_3) = 0.343$ which has $u_3^* = 1$ and $f_2(x_2) = 0.49$, which is for $u_2^* = 2$ and then $u_1^* = 1$. Thus the optimal allocation is: 3 units in parallel should be installed on component four, 1 unit on component three, 2 units on component two and 1 unit on component one.

1.7 SOLUTION OF L.P.P BY DYNAMIC PROGRAMMING

The linear programming problem in the general form is

$$\begin{aligned} \text{maximize} \quad & Z = c_1x_1 + c_2x_2 + \dots + c_nx_n, \\ \text{subject to} \quad & a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \leq b_1, \\ & a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \leq b_2, \\ & \vdots \\ & a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \leq b_m, \\ & x_1, x_2, \dots, x_n \geq 0. \end{aligned}$$

This problem involving m resources and n decision variables can be formulated as a dynamic programming problem as follows:

Each activity j ($j = 1, 2, \dots, n$) is considered a stage. Then the problem can be regarded as n -stage problem and decision variables (alternatives) are the levels of activities x_j (≥ 0) at stage j . Since x_j is continuous, each activity has infinite number of alternatives (values of x_j) within the feasible space.

Allocation problems are a particular type of L.P. problems that require allocation of available resources to the activities. The constants b_1, b_2, \dots, b_m are the amounts of the available resources. The state of the system at any stage is given by the amount allocated at that stage and left for the remaining stages. The state would be thus an m -dimensional vector (b_1, b_2, \dots, b_m) .

Let $f_n(b_1, b_2, \dots, b_m)$ be the optimal value of the objective function defined above for stages x_1, x_2, \dots, x_n for states b_1, b_2, \dots, b_m . Using forward computational procedure, the recursive equation can be written as

$$f_j(b_1, b_2, \dots, b_m) = \max_{0 \leq x_j \leq b} [c_jx_j + f_{j-1}(b_1 - a_{1j}x_j, b_2 - a_{2j}x_j, \dots, b_m - a_{mj}x_j)].$$

The maximum value of b that x_j can assume is

$$b = \min \left[\frac{b_1}{a_{1j}}, \frac{b_2}{a_{2j}}, \dots, \frac{b_m}{a_{mj}} \right].$$

EXAMPLE 1.7 - 1

Using dynamic programming to solve the following L.P.P. :

$$\begin{aligned} \text{Maximize } Z &= 3x_1 + 5x_2 \\ \text{subject to } x_1 &\leq 4, \\ x_2 &\leq 6, \\ 3x_1 + 2x_2 &\leq 18 \\ x_1, x_2 &\geq 0. \end{aligned}$$

Solution.

There are two variables and hence the problem can be treated as a two-stage dynamic programming problem. Both x_1 and x_2 being continuous, represent the infinite number of alternatives within the feasible space. The three constraints can be regarded as three resources, say b_1 , b_2 , and b_3 which are to be allocated to x_1 and x_2 at different stages. The state of the system at a stage would be given by the amount of resources allocated at that stage and left for the remaining stages. Therefore, the states of the equivalent D.P. problem are $b_1 = 4$, $b_2 = 6$ and $b_3 = 18$.

Stage 1

The optimal value $f_1(b_1, b_2, b_3)$ at stage 1 is given by

$$f_1(b_1, b_2, b_3) = \max_{0 \leq x_1 \leq b} [3x_1],$$

where $b_1 = 4$, $b_2 = 6$, $b_3 = 18$.

The feasible value of x_1 is non-negative and satisfies all the three constraints.

But the maximum value of that x_1 can assume is $= \min\left(\frac{4}{1}, \frac{6}{0}, \frac{18}{3}\right) = 4$.

$$\therefore f_1(4, 6, 18) = \max_{0 \leq x_1 \leq 4} [3x_1] = 3 \min\left[4, \frac{18 - 2x_2}{3}\right],$$

where $x_1^0 = \min\left[4, \frac{18 - 2x_2}{3}\right] = 4$.

Stage 2

The recursive equation for optimization of this two-stage problem is

$$\begin{aligned} f_2(4, 6, 18) &= \max_{0 \leq x_2 \leq b} [5x_2 + 3x_1] \\ &= \max_{0 \leq x_2 \leq b} \left[5x_2 + 3 \min\left(4, \frac{18 - 2x_2}{3}\right) \right]. \end{aligned}$$

But the maximum value of b that x_2 can assume is $= \min\left(\frac{4}{0}, \frac{6}{1}, \frac{18}{2}\right) = 6$.

$$\therefore f_2(4, 6, 18) = \max_{0 \leq x_2 \leq 6} \left[5x_2 + 3 \min\left(4, \frac{18 - 2x_2}{3}\right) \right].$$

Now
$$\min\left(4, \frac{18-2x_2}{3}\right) = \begin{cases} 4 & , \text{if } 0 \leq x_2 \leq 3, \\ \frac{18-2x_2}{3} & , \text{if } 3 < x_2 \leq 6. \end{cases}$$

$$\therefore 5x_2 + 3 \min\left(4, \frac{18-2x_2}{3}\right) = \begin{cases} 5x_2 + 12 & , \text{if } 0 \leq x_2 \leq 3, \\ 3x_2 + 18 & , \text{if } 3 < x_2 \leq 6. \end{cases}$$

$$\therefore f_2(4, 6, 18) = \max \begin{cases} 27 & , \text{at } x_2 = 3, \\ 36 & , \text{at } x_2 = 6, \end{cases}$$

$$= 36.$$

$$\therefore x_2^* = 6, Z_{\max} = 36 \text{ and } x_1^* = \min\left[4, \frac{18-2x_2}{3}\right] = \min\left[4, \frac{18-12}{3}\right] = 2$$

EXAMPLE 1.7 - 2

Solve the following L.P.P. by the method of dynamic programming:

$$\begin{aligned} \text{Maximize } Z &= 2x_1 + 5x_2, \\ \text{Subject to } 2x_1 + x_2 &\leq 430, \\ 2x_2 &\leq 460, \\ x_1, x_2 &\geq 0. \end{aligned}$$

Solution:

The problem involves two decision variables and two resources b_1 and b_2 . Therefore, the states of the equivalent D.P. problem are $b_1 = 430, b_2 = 460$.

Stage 1

$$f_1(b_1, b_2) = \max_{0 \leq x_1 \leq b} [2x_1].$$

$$\begin{aligned} \text{The maximum value of } b \text{ that } x_1 \text{ can assume is } &= \min\left(\frac{430}{2}, \frac{460}{0}\right) \\ &= \min(215, \infty) = 215. \end{aligned}$$

$$f_1(430, 460) = \max_{0 \leq x_1 \leq 215} [2x_1] = 2 \min\left[\frac{430-x_2}{2}, \infty\right],$$

and $x_1^0 = 215.$

Stage 2

The recursive equation is

$$f_2(b_1, b_2) = \max_{0 \leq x_2 \leq b} [5x_2 + 2x_1]$$

or
$$f_2(430, 460) = \max_{0 \leq x_2 \leq b} \left[5x_2 + 2 \min \left(\frac{430 - x_2}{2}, \infty \right) \right].$$

But the maximum value of b that x_2 can assume is $\min \left(\frac{430}{1}, \frac{460}{2} \right) = 230$

$$\therefore f_2(430, 460) = \max_{0 \leq x_2 \leq 230} \left[5x_2 + 2 \min \left(\frac{430 - x_2}{2}, \infty \right) \right].$$

Now
$$\min \left(\frac{430 - x_2}{2}, \infty \right) = \frac{430 - x_2}{2} = 100, \quad 0 \leq x_2 \leq 230.$$

$$\therefore 5x_2 + 2 \min \left(\frac{430 - x_2}{2}, \infty \right) = 5x_2 + 200, \quad 0 \leq x_2 \leq 230.$$

$$\therefore f_2(430, 460) = 5x_2 + 200 = 5 \times 230 + 200 = 1,150 + 200 = 1,350.$$

$$\therefore x_2^* = 230, \quad Z_{\max} = 1,350 \text{ and } x_1^* = \frac{430 - 230}{2} = 100.$$

EXAMPLE 1.7 – 3

Solve the following L.P.P. by dynamic programming:

$$\begin{aligned} &\text{Maximize } Z = 50x_1 + 100x_2, \\ &\text{subject to } \begin{aligned} 10x_1 + 5x_2 &\leq 2,500, \\ 4x_1 + 10x_2 &\leq 2,000, \\ x_1 + \frac{3}{2}x_2 &\leq 450, \\ x_1, x_2 &\geq 0. \end{aligned} \end{aligned}$$

Solution.

The problem involves two decision variables and three resources b_1 , b_2 and b_3 . Therefore, the states of the equivalent D.P problem are $b_1 = 2,500$, $b_2 = 2,000$ and $b_3 = 450$.

Stage 1

$$f_1(b_1, b_2, b_3) = \max_{0 \leq x_1 \leq b} [50x_1].$$

where the maximum is to be taken over $0 \leq 10x_1 \leq 2,500$; $0 \leq 4x_1 \leq 2,000$ and $0 \leq x_1 \leq 450$.

The maximum value of b that x_1 can assume is $\min\left(\frac{2,500}{10}, \frac{2,000}{4}, \frac{450}{1}\right) = 250$.

$$\begin{aligned}\therefore f_1(2,500, 2,000, 450) &= \max_{0 \leq x_1 \leq 250} [50x_1] \\ &= 50 \min\left[\frac{2,500-5x_2}{10}, \frac{2,000-10x_2}{4}, \frac{450-3/2x_2}{1}\right]\end{aligned}$$

and $x_1^0 = 250$.

Stage 2

The recursive equation is

$$f_2(b_1, b_2, b_3) = \max_{0 \leq x_2 \leq b} [100x_2 + 50x_1]$$

$$\begin{aligned}\text{or } f_2(2,500, 2,000, 450) &= \max_{0 \leq x_2 \leq b} \left[100x_2 + 50 \min\left(\frac{2,500-5x_2}{10}, \frac{2,000-10x_2}{4}, \frac{450-3/2x_2}{1}\right) \right]\end{aligned}$$

Now the maximum value of b that x_2 can assume is

$$= \min\left(\frac{2,500}{5}, \frac{2,000}{10}, \frac{450}{3/2}\right) = \min(500, 200, 300) = 200.$$

$$\begin{aligned}\therefore f_2(2,500, 2,000, 450) &= \max_{0 \leq x_2 \leq 200} \left[100x_2 + 50 \min\left(\frac{2,500-5x_2}{10}, \frac{2,000-10x_2}{4}, \frac{450-3/2x_2}{1}\right) \right]\end{aligned}$$

Now

$$\begin{aligned}\min\left[\frac{2,500-5x_2}{10}, \frac{2,000-10x_2}{4}, \frac{450-3/2x_2}{1}\right] &= \begin{cases} \frac{2,500-5x_2}{10}, & \text{if } 0 \leq x_2 \leq 125, \\ \frac{2,000-10x_2}{4}, & \text{if } 125 < x_2 \leq 200. \end{cases} \\ \therefore f_2(2,500, 2,000, 450) &= \max \begin{cases} 100x_2 + 50 \cdot \frac{2,500-5x_2}{10}, & \text{if } 0 \leq x_2 \leq 125, \\ 100x_2 + 50 \cdot \frac{2,000-10x_2}{4}, & \text{if } 125 < x_2 \leq 200. \end{cases} \\ &= \max \begin{cases} 75x_2 + 12,500, & \text{if } 0 \leq x_2 \leq 125, \\ 25,000 - 25x_2, & \text{if } 125 < x_2 \leq 200. \end{cases} \\ &= \max \begin{cases} 21,875, & \text{at } x_2 = 125, \\ 21,875, & \text{at } x_2 = 125. \end{cases} \\ &= 21,875.\end{aligned}$$

$$\therefore Z_{\max} = 21,875; x_2^* = 125, x_1^* = \min\left[\frac{2,500-5x_2^*}{10}, \frac{2,000-10x_2^*}{4}, \frac{450-3/2x_2^*}{1}\right]$$

$$\begin{aligned}
 \text{or } x_1^* &= \min \left[\frac{2,500 - 5 \times 125}{10}, \frac{2,000 - 10 \times 125}{4}, 450 - \frac{3}{2} \times 125 \right] \\
 &= \min[187.5, 187.5, 262.5] \\
 &= 187.5.
 \end{aligned}$$

1.8 APPLICATIONS OF DYNAMIC PROGRAMMING

We have discussed some over-simplified examples from the various fields of applications of dynamic programming. Many more applications are found for this decision-making technique. Whereas linear programming has found its applications in large-scale complex situations, dynamic programming has more applications in smaller-scale systems.

Following are a few of the large number of fields in which dynamic programming has been successfully applied:

1. **Production.** In the production area, this technique has been employed for *production, scheduling and employment smoothening*, in the face of widely fluctuating demand requirements.
2. **Inventory Control.** This technique has been used to determine the optimum inventory level and for formulating the inventory reordering rules, indicating when to replenish an item and by what amount.
3. **Allocation of Resources.** It has been employed for allocating the scarce resources to different alternative uses, such as allocating salesmen to different sales zones and *capital budgeting procedures*.
4. **Selection of Advertising Media.** (See example 4.10)
5. **Spare Part Level Determination** to guarantee high efficiency utilization of expensive equipment.
6. **Equipment Replacement Policies.** To determine at which stage equipment is to be replaced for optimal return from the facilities.
7. Scheduling methods for routine and major overhauls on complex machinery.
8. Systematic plan or search to discover the whereabouts of a valuable resource.

These are only a few of the wide range of situations to which dynamic programming has been successfully applied. Many real operating systems call for thousands of such decisions. The dynamic programming models make it possible to make all these decisions, of course with the help of computers. These decisions individually

may not appear to be of much economic benefit, but in aggregate they exert a major influence on the economy of a firm.

1.9 DETERMINISTIC DYNAMIC PROGRAMMING

In deterministic dynamic programming the decisions are under the control of the decision-maker and outcomes of decisions have values which are fixed and certain. The number of decisions to be made may be finite or infinite. Example 1.4-1 on allocation of nine salesmen to three zones involves revenues which are fixed and known with certainty and hence forms a deterministic D.P. problem. The problem involves three stages and hence 3 decisions. However, in many situations the number of decisions to be taken could be infinite. For instance, the intention of the management of an organization quite often is to remain in business indefinitely. Likewise, the objective of a production manager is to minimize the production costs over an indefinite future period.

In such situations, since each decision stage will involve some cost, the cost of all the decision policies will always be unlimited and it is futile to talk of choosing a policy that minimizes the total cost of all the decisions. Two approaches are followed in such cases:

The first approach discounts the costs to be incurred in future. This approach is justified when the decisions to be made are not very frequent, say not more than once or twice a year. Here, the costs incurred in the current year will have more influence on the decisions than the costs incurred next year and so on.

The second approach is used when the decisions are to be made frequently, say once a week or fortnight, so that discounting the costs each week over long indefinite periods will be too long and cumbersome. In such cases a policy that minimizes the average cost per decision rather than the total cost is selected.

1.10 PROBABILISTIC DYNAMIC PROGRAMMING

In probabilistic dynamic programming the outcomes of the decisions are not certain and are associated with probabilities. These situations also may involve finite or infinite number of decisions.

(i) **Finite number of decisions** : In such cases decision tree approach is used. Decision tree diagram is a graphical representation of various decisions, alternatives and their outcomes in a decision - making problem. In constructing a decision tree there are certain conventions to be followed. The tree is constructed by starting from left and moving towards right. The square box \square denotes a decision point at which the available strategies are considered. The circle \circ represents the chance node or event. The various outcomes or states of nature emanate from this chance node. These states of nature are represented by straight lines. Probabilities are associated with these outcomes.

Suppose a company manufacturing toys is to decide whether to introduce the deluxe model or popular model. From market survey, the probabilities of market demand along with the associated profit or loss from sales for both the models is known. To analyze the problem and to decide the optimum strategy, the problem can be represented as a decision tree shown in Fig. 1.2. There are two branches at the decision node 1, each branch representing a strategy. Similarly, at each of the two chance nodes A and B, there are three branches representing outcomes.

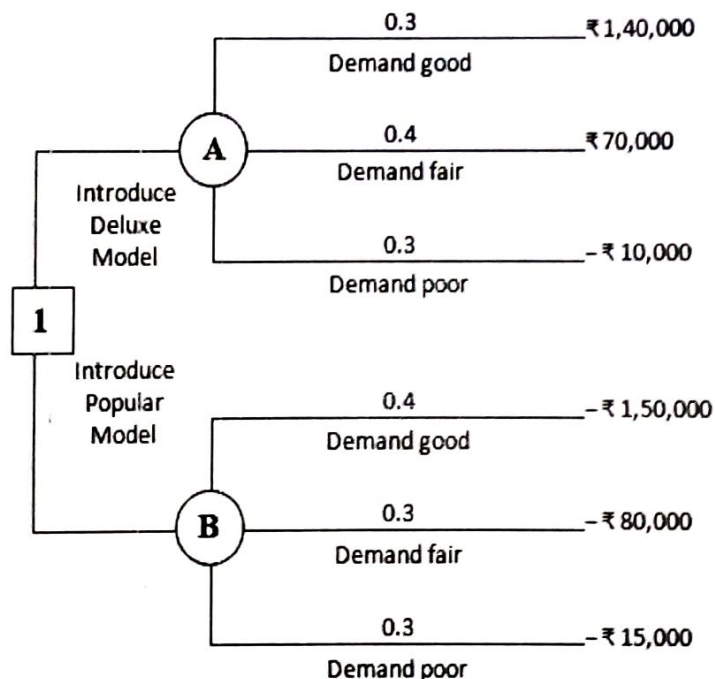


FIGURE 1.1

Thus the decision tree shows the structure of the decision problem. To carry out the decision analysis, conditional payoffs are estimated for every combination of actions

and events. The payoffs can be either positive or negative. Also probabilities for each event must be assessed by the decision-maker. Analysis of the decision tree is done by the following roll back method or backward pass method as shown below.

$$\begin{aligned}\text{Expected payoff at chance node A} &= ₹ [0.3 \times 1,40,000 + 0.4 \times 70,000 - \\ &\quad 0.3 \times 10,000] \\ &= ₹ (42,000 + 28,000 - 3,000) \\ &= ₹ 67,000.\end{aligned}$$

$$\begin{aligned}\text{Expected payoff at chance node B} &= ₹ [0.4 \times 1,50,000 + 0.3 \times 80,000 - \\ &\quad 0.3 \times 15,000] \\ &= ₹ [60,000 + 24,000 - 4,500] \\ &= ₹ 79,500.\end{aligned}$$

$$\begin{aligned}\therefore \text{Expected payoff at decision node 1} &= \text{Max ₹ [67,000; 79,500]} \\ &= ₹ 79,500.\end{aligned}$$

Therefore, the company should produce the popular model.

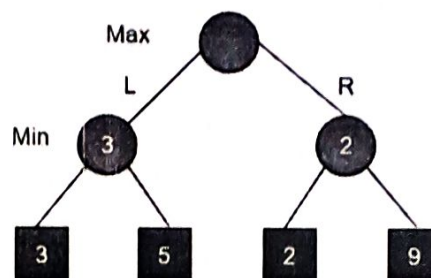
(ii) Infinite number of decisions

There are situations in which the outcomes are probabilistic and the same decision is to be made at regular intervals and this process continues indefinitely e.g., problems involving maintenance, replacement of equipment, etc. The approach in such situations is to maximize the outcome per decision rather than the total outcome over an indefinite period.

APPLICATIONS

1. Dynamic Programming on Graph Theory:

Dynamic programming is “an algorithmic technique which is usually based on a recurrent formula and one (or some) starting states.” When it’s applied to graphs, we can solve for the shortest paths with one source or shortest paths for every pair. Graph Theory consists of problems and ways to model things you may encounter in game theory, or have problems that can be solved using dynamic programming. There are many aspects of Graph Theory that have little to do with game theory and are inapplicable to employing dynamic programming. Some game theory utilizes graphs, and some graph theoretic problems can be solved using dynamic programming.



2. Dynamic Programming on Fibonacci Sequence:

Using dynamic programming in the calculation of the n th member of the Fibonacci sequence improves its performance greatly. Here is a naïve implementation, based directly on the mathematical definition:

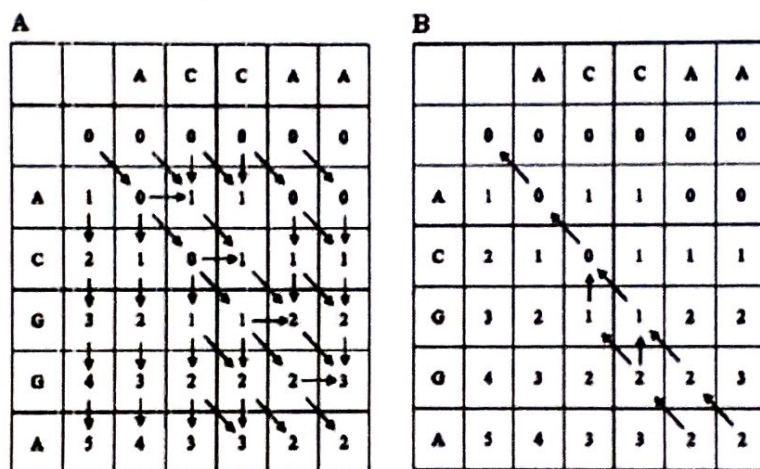
```
function fib(n)
if n <= 1 return n
return fib(n - 1) + fib(n - 2)
```

3. Dynamic Programming on Bioinformatics:

Dynamic programming (DP) is a most fundamental programming technique in bioinformatics. Sequence comparison, gene recognition, RNA structure prediction and hundreds of other problems are solved by ever new variants of DP. The first dynamic programming algorithms for protein-DNA binding were developed in the 1970s independently by Charles DeLisi in USA and Georgii Gurskii and Alexander Zasedatelev in USSR. Recently these algorithms have become very popular in bioinformatics

and computational biology, particularly in the studies of nucleosome positioning and transcription factor binding.

Despite of all available experience, the development of the typical DP recurrences is nontrivial, and their implementation presents quite a few pitfalls. To quote a recent comment by an expert: Presently, developing efficient DP algorithms is a matter of experience, talent and luck. A more systematic or mechanical way to develop efficient DP algorithms would be very valuable in many fields in addition to bioinformatics. In my own experience, developing successful DP occurrences is hard, and I would like to learn a mechanistic way to do it on problems that haven't already been worked out.



CONCLUSION

Dynamic programming is a mathematical modeling theory that is useful for solving a select set of problems involving a sequence of interrelated decisions. Dynamic programming provides a systematic means of solving multistage problems over a planning horizon or a sequence of probabilities.

As an example, a stock investment problem can be analyzed through a dynamic programming model to determine the allocation of funds that will maximize total profit over a number of years. Decision making in this case requires a set of decisions separated by time. Dynamic programming is a powerful tool that allows segmentation or decomposition of complex multistage problems into a number of simpler subproblems.

Most of the problems you'll encounter within Dynamic Programming already exist in one shape or another. Often, your problem will build on from the answers for previous problems. Among all its tremendous applications, the most important contribution of Dynamic Programming is RNA structure prediction and protein - DNA binding. So, by making effective researches on Dynamic Programming, mankind will be much benefited.

REFERENCES

- [1] Bellman, Richard (1954), "**The theory of dynamic programming**", Bulletin of the American Mathematical Society, DOI:10.1090/S0002-9904-1954-09848-8, MR 0067459. Includes an extensive bibliography of the literature in the area, up to the year 1954.
- [2] Bellman, Richard (1957), **Dynamic Programming**, Princeton University Press. Dover paperback edition (2003), ISBN 0-486-42809-5.
- [3] Dreyfus, Stuart E.; Law, Averill M. (1977), **The Art and Theory of Dynamic Programming**, Academic Press, ISBN 978-0-12-221860-6.
- [4] Giegerich. R.; Meyer. C.; Steffen. P.; (2004), "**A Discipline of Dynamic Programming over Sequence Data**", Science of Computer Programming.

FOURIER TRANSFORMS : STUDY, APPLICATIONS IN HEALTH AND DATA SCIENCES AND CODING THROUGH SOFTWARES

Project Report submitted to

ST.MARY'S COLLEGE (AUTONOMOUS), THOOTHUKUDI

Affiliated to

MANONMANIAM SUNDARANAR UNIVERSITY, TIRUNELVELI

In partial fulfillment of the requirement for the award of degree of

Bachelor of Science in Mathematics

Submitted by

NAME

REG.NO.

JOAN MORAIS. R

19AUMT17

MARIA ABITHAA VICTORIA. C

19AUMT21

MARIA BENJAMINI. A

19AUMT23

MONISHA ROSE. G

19AUMT28

YAMINI. S

19AUMT50

Under the Guidance of

Dr. Sr. S. KULANDAI THERESE M.Sc., B.Ed., M.Phil., Ph.D.

Assistant Professor of Mathematics

St. Mary's College (Autonomous), Thoothukudi.



Department of Mathematics

St. Mary's College (Autonomous), Thoothukudi

(2021 - 2022)

CERTIFICATE

We hereby declare that the project report entitled "**FOURIER TRANSFORMS : STUDY, APPLICATIONS IN HEALTH AND DATA SCIENCES AND CODING THROUGH SOFTWARES**" being submitted to **St. Mary's College (Autonomous), Thoothukudi** affiliated to **Manonmaniam Sundaranar University, Tirunelveli** in partial fulfillment for the award of degree of **Bachelor of Science in Mathematics** and it is a record of work done during the year 2021 - 2022 by the following students :


NAME	REG.NO.
JOAN MORAIS. R	19AUMT17
MARIA ABITHAA VICTORIA. C	19AUMT21
MARIA BENJAMINI. A	19AUMT23
MONISHA ROSE. G	19AUMT28
YAMINI. S	19AUMT50



Signature of the Guide
Dr. S. KULANDAI THERESE
M.Sc., B.Ed., M.Phil., Ph.D.,
Assistant Professor,
Department of Mathematics,
St. Mary's College (Autonomous),
Thoothukudi - 628 001.



Signature of the Examiner


Dr. Signature of the HOD Mary
M.Sc., M.Phil., B.Ed., Ph.D.,
Head & Asst Professor of Mathematics
St. Mary's College (Autonomous)
Thoothukudi-628 001.



Signature of the Principal

Principal
St. Mary's College (Autonomous)
Thoothukudi-628 001.

DECLARATION

We hereby declare that the project reported entitled "**FOURIER TRANSFORMS : STUDY, APPLICATIONS IN HEALTH AND DATA SCIENCES AND CODING THROUGH SOFTWARES**", is our original work. It has not been submitted to any university for any degree or diploma.



(JOAN MORAIS. R)



(MARIA BENJAMINI. A)



(MONISHA ROSE. G)



(YAMINI. S)



(MARIA ABITHAA VICTORIA. C)

ACKNOWLEDGEMENT

First of all, we thank Lord Almighty for showering his blessings to undergo this project.

With immense pleasure, we register our deep sense of gratitude to our guide **Dr. Sr. S. Kulandai Therese M.Sc., B.Ed., M.Phil., Ph.D.** and the Head of the Department, **Dr. V. L. Stella Arputha Mary M.Sc., M.Phil., B.Ed., Ph.D.** for having imparted necessary guidelines throughout the period of our studies.

We thank our beloved Principal, **Rev. Dr. Sr. A.S.J. Lucia Rose M.Sc., M.Phil., Ph.D., PGDCA** for providing us the help to carry out our project work successfully.

Finally, we thank all those who extended their helping hands regarding this project.

FOURIER TRANSFORMS

PREFACE

The topic of our Project "FOURIER TRANSFORMS : STUDY, APPLICATIONS IN HEALTH AND DATA SCIENCES AND CODING THROUGH SOFTWARES", focuses on underlying concepts of the discipline and behavioural aspects of signals in time domain and frequency domain. Fourier Transform named after Joseph Fourier, is a mathematical transformation employed to transform signals between time (or spatial) domain and frequency domain. The Fourier Transform allows us to perform tasks that would be impossible to perform any other way.

Important properties, standard formulae, definitions, relevant examples and references have also been discussed to scaffold the readers on the necessary concepts. After brief learning, it was a natural progression to apply the learned concepts and practices in real life. Fourier Transform has multitude of applications in almost all areas of life which have been discussed in our project. The Project is structured into five chapters :

Chapter 1 presents briefly the idea of What is a Fourier Transform and it's Types.

Chapter 2 deals with the most important Applications of Fourier Transform and the softwares which can be used to calculate all types of Fourier Transforms.

Chapter 3 introduces the definitions of Fourier Transform, it's Inverse, Fourier Cosine and Sine Transform and it's Inverses respectively.

Chapter 4 focuses on Properties, Fourier Integral Theorem, Alternative Form of Fourier Complex Integral Formula, Standard Fourier Transform Pairs, Deriving Fourier Transform from Fourier Series and Relationship between Fourier Transform and Laplace Transform.

Chapter 5 deals with Finite Fourier Sine and Cosine Transform, it's Inverses and Finite Fourier Transforms of Derivatives.

CONTENT

1 Chapter 1

1.1 Introduction	9
1.2 Types of Fourier Transform	10

2 Chapter 2

2.1 Applications of Fourier Transform	11
2.2 Softwares that can be used to calculate all types of Fourier Transforms	18

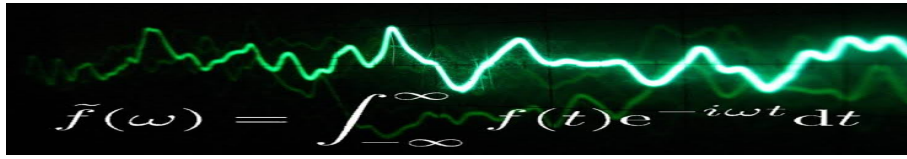
3 Chapter 3

3.1 Fourier Transform	22
3.2 Inverse Fourier Transform	24
3.3 Fourier Cosine Transform	26
3.4 Inverse Fourier Cosine Transform	27
3.5 Fourier Sine Transform	28
3.6 Inverse Fourier Sine Transform	29

4 Chapter 4

4.1 Fourier Integral Theorem	30
4.2 Alternative Form of Fourier Complex Integral Formula	34
4.3 Standard Fourier Transform Pairs	35

4.4	Deriving Fourier Transform from Fourier Series	36
4.5	Relationship between Fourier Transform and Laplace Transform	37
4.6	Properties of Fourier Transform	38
5	Chapter 5	
5.1	Finite Fourier Transforms of Derivatives	48
5.2	Finite Fourier Sine Transform	49
5.3	Finite Fourier Cosine Transform	50
5.4	Inverse Finite Fourier Sine Transform	50
5.5	Inverse Finite Fourier Cosine Transform	51
6	Conclusion	53
7	References	54



$$\tilde{f}(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt$$

FOURIER TRANSFORMS : STUDY, APPLICATIONS IN HEALTH AND DATA SCIENCES AND CODING THROUGH SOFTWARES

1.1 Introduction

“Profound study of nature is the most fertile source of mathematical discoveries.”

- Joseph Fourier

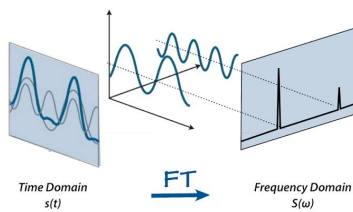
Jean Baptiste Joseph Fourier (1768 – 1830) was a French mathematician and Physicist, best known for initiating the investigation of Fourier series, which eventually developed into Fourier analysis and Harmonic analysis, and their applications to problems of Heat transfer and Vibrations. The Fourier Transform and Fourier’s law of conduction are also named in his honour. Fourier is also generally credited with the discovery of the Greenhouse effect.



What is Fourier Transform?

In mathematics, Fourier Transform is a mathematical technique that transforms a function of time, $x(t)$, to a function of frequency, $X(\omega)$

It is a mathematical transform that decomposes functions depending on space or time into functions depending on spatial or temporal frequency. The term Fourier transform refers to both the frequency domain representation and the mathematical operation that associates the frequency domain representation to a function of space or time. The Fourier transform can be formally defined as an improper Riemann integral, making it an integral transform.



The representation of periodic signals as a linear combination of harmonically related complex exponentials can be extended to develop a representation of periodic signals as linear combination of complex exponentials. This leads to Fourier Transforms. Also, Fourier Transform is a tool that breaks a waveform (a function or signal) into an alternate representation, characterized by sine and cosines. The Fourier Transform shows that any waveform can be re-written as the sum of sinusoidal functions. The more concentrated $f(x)$ is, the more spread out its Fourier transform $f(e)$ must be.

It is closely related to the Fourier Series and it is an extended form of Fourier Analysis. Fourier Series is mainly used for periodic signals whereas Fourier Transform is used for non-periodic signals.

Dirichlet's Conditions (Conditions for existence of Fourier transform)

1. $f(t)$ should be absolutely integrable (i.e.) $\int_{-\infty}^{\infty} |f(t)| dt < \infty$.
2. The function must have finite number of maxima and minima.
3. The function must have finite number of discontinuities.

The choice of a particular transform is decided by the nature of the boundary conditions and the convenience of inverting the transform function $\tilde{f}(s)$ to give $f(x)$.

1.2 TYPES OF FOURIER TRANSFORM

- Continuous Fourier Transform
- Discrete - Time Fourier Transform
- Discrete Fourier Transform
- Discrete Fourier Transform over a Ring
- Fourier Transform on Finite Groups
- Fourier Analysis
- Fast Fourier Transform

2.1 APPLICATIONS OF FOURIER TRANSFORM

The Fourier Transform

$$\mathcal{F}\{g(t)\} = G(f) = \int_{-\infty}^{\infty} g(t)e^{-i2\pi ft} dt$$
$$\mathcal{F}^{-1}\{G(f)\} = g(t) = \int_{-\infty}^{\infty} G(f)e^{i2\pi ft} df$$



1. Rapid diagnosis of COVID-19 using FT-IR ATR spectroscopy and machine learning

Attenuated Total Reflection - Fourier Transform InfraRed (ATR-FTIR) Spectroscopy associated with machine learning in oropharyngeal swab suspension fluid is applicable to predict COVID-19 positive samples.

The study included samples of 243 patients from two Brazilian States. Samples were transported by using different viral transport mediums (liquid 1 or 2). Clinical COVID-19 diagnosis was performed by the Reverse Transcription - Polymerase Chain Reaction (RT-PCR). Researchers built a classification model based on Partial Least Squares (PLS) associated with cosine k-Nearest Neighbours (KNN). Their analysis led to 84% and 87% sensitivity, 66% and 64% specificity, and 76.9% and 78.4% accuracy for samples of liquids 1 and 2, respectively. Based on this proof-of-concept study, they believe this method could offer a **simple, label-free, cost-effective solution** for high-throughput screening of suspect patients for COVID-19 in health care centres and emergency departments.



This technique has shown promise as a diagnostic or screening tool in several diseases such as cancer, diabetes, hypertension, and physiological stress.

Recently, **ATR-FTIR** has already been investigated as a screening/diagnostic tool in medicine. In 2019, the use of this technique was reported in the screening of patients with brain

cancer, achieving sensitivity of 93.2% and specificity of 92.8% in the identification of high-risk patients indicated for Definitive Diagnostic Tests (more expensive), thus saving time and cost. In infectious diseases, a similar study was done to discriminate patients with Human immunodeficiency virus (HIV) infection by **ATR-FTIR** also associated with Linear Discriminant Analysis (LDA) in plasma samples. Interestingly, this analysis proved to be a possible strategy for discrimination against different spectra of HIV infection and co-infection with the hepatitis C virus (AIDS, HIV + HCV or AIDS + HCV).

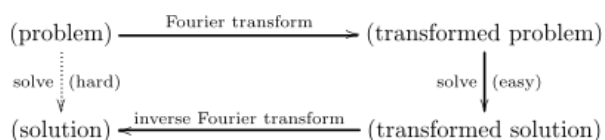
2. Fourier Transform on Data Science

- Animated Visualization using Fourier Transform.
- Clean Up Data Noise with Fourier Transform in Python.
- Image Processing and Removal of Image Elements with Python - Application of Fourier Transformation.

One of the more advanced topics in image processing has to do with the concept of Fourier Transformation. Put very briefly, some images contain systematic noise that users may want to remove. If such noise is regular enough, employing Fourier Transformation adjustments may aid in image processing.



3. Fourier Transform on Analysis of Differential Equations



Some problems, such as certain differential equations, become easier to solve when the Fourier transform is applied. In that case the solution to the original problem is recovered

using the inverse Fourier transform. The operation of differentiation in the time domain corresponds to multiplication by the frequency, so some differential equations are easier to analyze in the frequency domain. Perhaps the most important use of the Fourier transformation is to solve partial differential equations.

4. Fourier Transform Spectroscopy

Fourier-transform spectroscopy is a measurement technique whereby spectra are collected based on measurements of the coherence of a radiative source, using time-domain or space-domain measurements of the radiation and electromagnetic.



The Fourier transform is also used in **Nuclear Magnetic Resonance (NMR)** and in other kinds of spectroscopy, e.g. **Infrared (FTIR)**. In NMR an exponentially shaped Free Induction Decay (FID) signal is acquired in the time domain and Fourier-transformed to a Lorentzian line-shape in the frequency domain. The Fourier transform is also used in **Magnetic Resonance Imaging (MRI)** and **Mass Spectrometry**.

5. Fourier Transform on Signal Processing

The Fourier transform is used for the spectral analysis of time-series. The subject of statistical signal processing does not, however, usually apply the Fourier transformation to the signal itself. Even if a real signal is indeed transient (Lasting only for a short time; Impermanent), it has been found in practice advisable to model a signal by a function which is stationary in the sense that its characteristic properties are constant over all time. The Fourier transform of such a function does not exist in the usual sense, and it has been found more useful for the analysis of signals to instead take the Fourier transform of its auto correlation function. For video signals other types of spectral analysis must also be employed, still using the Fourier transform as a tool.



6. Fourier Transform on Quantum mechanics

The Fourier transform is useful in quantum mechanics. The Fourier transform can be used to pass from one way of representing the state of the particle, by a wave function of position, to another way of representing the state of the particle: by a wave function of momentum. The other use of the Fourier transform

in both quantum mechanics and quantum field theory is to solve the applicable wave equation. Fourier methods have been adapted to also deal with non-trivial interactions.

7. Fourier Transform on Circuit Analysis

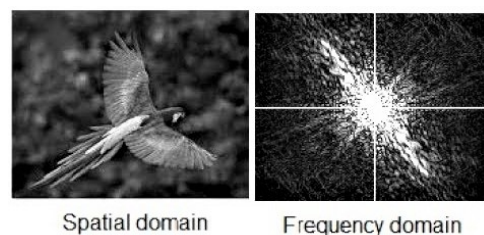
There are many linear circuits used in Electronic engineering field. These circuits include various components like capacitor, inductor, resistor etc. Every Electronic circuit can be modelled using mathematical equations. To perform frequency analysis of the circuit Fourier Transform is used. Fourier Transform helps us to analyse the behavior of circuit when different inputs are applied.

8. Fourier Transform on Cell phones

Communication is all based on Mathematics. The communication includes automatic transmission of data over wires and radio circuits through signals. Cell phones are one of the most prominent communication device. The principle of Fourier Transform is used in signal, which can be represented as the sum of a collection of sine and cosine waves with various frequencies and amplitudes. This collection of waves can then be manipulated with relative ease. Our mobile phone has performing Fourier Transform. Every mobile device - such as netbook, tablet and phone have been built in high speed cellular connection, just like Fourier Transform. Humans very easily perform it mechanically everyday. For example, when you are in a room with a great deal of noise and you selectively hear your name above the noise, then you just performed Fourier transform.

9. Fourier Transform on Image Processing

The Fourier Transform is used in a wide range of applications such as image analysis, image filtering, image reconstruction and image compression. The Fourier Transform is an important image processing tool which is used to decompose an image into its sine and cosine components. The output of the transformation represents the image in



the Fourier or frequency domain, while the input image is the spatial domain equivalent. In the Fourier domain image, each point represents a particular frequency contained in the spatial domain image.

10. Fourier Transform on Analysis of Linear Time Invariant (LTI) Systems

A signal is any waveform (function of time). This could be anything in the real world - an electromagnetic wave, the voltage across a resistor versus time, the air pressure variance due to your speech (i.e. a sound wave), or the value of Apple Stock versus time. The family of Fourier Transforms are specifically developed for analysing frequency contents of the signals for which there is no definition of linearity or time invariance. Hence we can define the Fourier transform of any signal, as long as it's integrable (i.e. stable).

11. Fourier Transform on Radio Astronomy

Radio astronomers are particularly avid users of Fourier transforms because Fourier transforms are key components in data processing (e.g., periodicity searches) and instruments (e.g., antennas, receivers, spectrometers) and they are the cornerstones of interferometry and aperture synthesis.



Radio Frequency Interference (RFI) makes the process of detecting and analyzing pulsars extremely difficult. This has forced astronomers to be creative in identifying and determining the specific characteristics of these unique rotating neutron stars. Astrophysicists have utilized algorithms such as the Fast Fourier Transform (FFT) to predict the spin period and harmonic frequencies of pulsars. Dedispersion and the pulsar frequency are critical for predict-

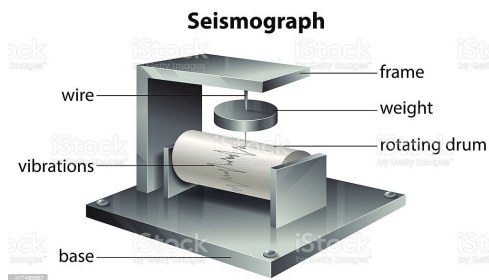
ing multiple characteristics of pulsars and correcting the influence of the Interstellar Medium (ISM). Hence, Discrete Fourier Transform is a useful technique for detecting radio signals and determining the pulsar frequency.

12. Fourier Transform on Astronomy

Fourier transforms are performed to learn about the spectral characteristics of a data set. Thus in astronomy, when looking for periodicities in a time series, we Fourier transform the data and look for peaks in the spectrum. If the data are regularly sampled we can make use of the Fast Fourier Transform to decrease the computation time.



13. Fourier Transform on Seismology



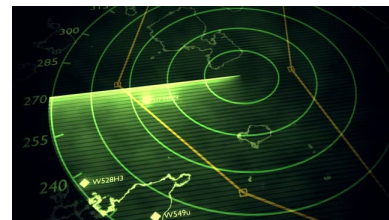
Fourier transform is fundamental to seismic data analysis. It applies to almost all stages of processing. A seismic trace represents a seismic wavefield recorded at a receiver location. The digital form of a seismic trace is a time series which can be completely described as a discrete sum of a number of sinusoids — each with a unique peak ampli-

tude, frequency, and a phase-lag (relative alignment). The analysis of a seismic trace into its sinusoidal components is achieved by the Forward Fourier Transform. Conversely, the synthesis of a seismic trace from the individual sinusoidal components is achieved by the Inverse Fourier Transform.

Seismic research has always been a common user for the Discrete Fourier Transform (and the FFT). If you look at the history of the FFT you will find that one of the original uses for the FFT was to distinguish between natural seismic events and nuclear test explosions because they generate different frequency spectra.

14. Fourier Transform on Radio Detection And Ranging (RADAR)

The Fourier- transformation has become a fundamental method in the signal processing procedures, since the radar echo contains a variety of informations in the signal form. This information is converted by the Fourier- transformation into a data format which can be used by the computer-aided signal processing. With help of the Fast Fourier analysis whole signal forms of radar echoes can be stored as only few data by the digital signal processing. These data can be used by the process of the identification of radar targets like fingerprints.



The signal received by a pulsed radar is a time sequence of pulses for which the amplitude and phase are measured. Doppler processing techniques are based on measuring the spectral (frequency) content of this signal. The frequency content of this time-domain signal is obtained by taking its Fourier transformation, thus turning it into a frequency-domain signal or spectrum of the time-domain signal.

15. Fourier Transform on Music



Fourier Transform helps in determining the constituent pitches in a musical waveform. While applying a Constant-Q transform (a Fourier-related transform) to the waveform of a C major piano chord, the first three peaks on the left correspond to the frequencies of the fundamental frequency of the chord (C, E, G). The remaining smaller peaks are higher-frequency overtones of the fundamental pitches. A pitch detection algorithm could use the relative intensity of these peaks to infer which notes the pianist pressed.

2.2 SOFTWARES THAT CAN BE USED TO CALCULATE ALL TYPES OF FOURIER TRANSFORMS

- Python
- MATLAB
- "WolframAlpha - Computational Intelligence" - Free Online Fourier Transform Calculator
- CoCalc

1. Python

Different types of Fourier Transform can be calculated through Python Coding within some minutes. Let us try to understand this through simple coding for calculating Fourier Transform (Continuous time and frequency).

Fourier Transform (Continuous time and frequency)

This occurs when the functional form of your time series is known analytically (i.e. you have a formula $x(t) = \dots$ for it) and goes from $-\infty$ to ∞ .

$$x(f) = \int_{-\infty}^{\infty} x(t) e^{-2\pi i f t} dt$$

Program

Write a Program Coding to find the Fourier Transform of the function kte^{-kt^2} using Python in terms of the final variable f .

Coding

```
In [1] : import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
plt.style.use( ['science', 'notebook'] )
import sympy as smp
from skimage import color
from skimage import io
from scipy.fft import fftfreq
from scipy.fft import fft, ifft, fft2, ifft2
In [2] : t, f = smp.symbols('t, f', real=True)
In [3] : t, f = smp.symbols('t, f', real=True)
```

```

k = symp.symbols('k', real=True, positive=True)
x = symp.exp(-k * t**2) * k * t
x
Out [3] :  $kte^{-kt^2}$ 
In [4] : from sympy.integrals.transforms import fourier_transform
In [5] : x_FT = fourier_transform(x, t, f)
x_FT
Out [5] :

```

$$-\frac{i\pi^{\frac{3}{2}}fe^{-\frac{\pi^2f^2}{k}}}{\sqrt{k}}$$

Like the above example, we can calculate any type of Fourier Transform using Python. Plots can also be plotted for Fourier Transforms.

2. MATLAB

Different types of Fourier Transform can be calculated through MATLAB Coding. Let us try to understand this through simple coding for calculating Fourier Transform of Unit impulse (Dirac delta) Function in MATLAB.

Note that **fourier(f)** returns the Fourier Transform of **f**. By default, the function **symvar** determines the independent variable, and **w** is the transformation variable.

Program

Write a Program to calculate the Fourier Transform of Unit impulse (Dirac delta) Function using MATLAB.

Coding

```

> f = dirac(t);
> f_FT = fourier(f)

```

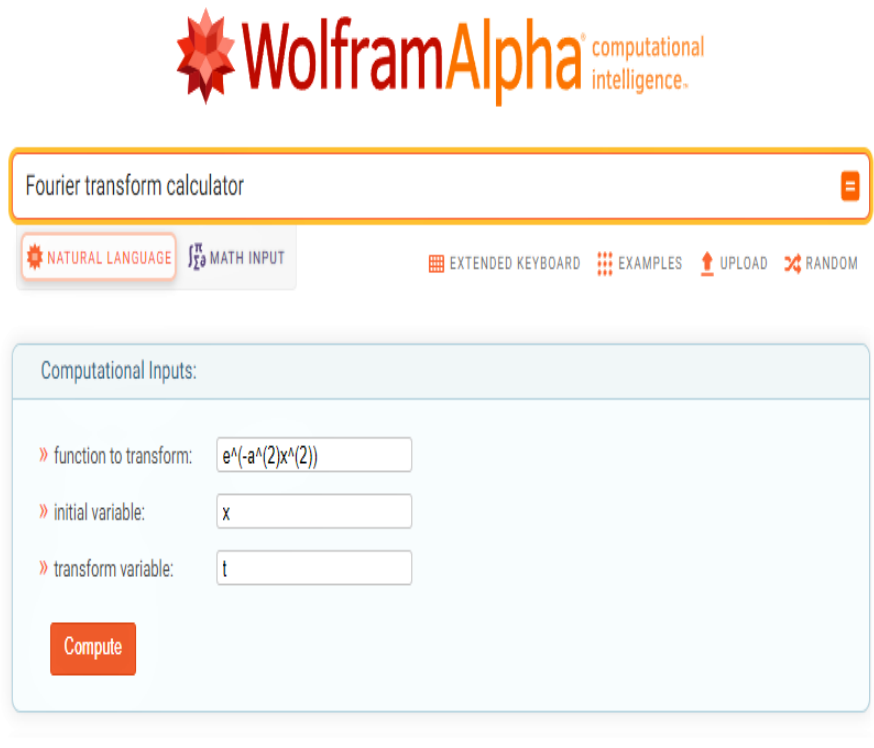
```

f_FT =
1

```

3. "WolframAlpha - Computational Intelligence" - Free Online Fourier Transform Calculator

"WolframAlpha" is an Free Online Fourier Transform Calculator through which all types of Fourier Transforms can be calculated instantly. All you need to know is which function to transform, initial variable and transform variable.



The screenshot shows the WolframAlpha website's Fourier transform calculator. At the top is the WolframAlpha logo with the tagline "computational intelligence". Below the logo is a search bar containing the text "Fourier transform calculator". Under the search bar are two tabs: "NATURAL LANGUAGE" and "MATH INPUT". To the right of these tabs are links for "EXTENDED KEYBOARD", "EXAMPLES", "UPLOAD", and "RANDOM". Below the tabs is a section titled "Computational Inputs:" which contains three input fields: "function to transform:" with the value $e^{-a^2(2)x^2(2)}$, "initial variable:" with the value x , and "transform variable:" with the value t . At the bottom of this section is a red "Compute" button.

4. CoCalc

Different types of Fourier Transform can be calculated through CoCalc Coding also.

Program

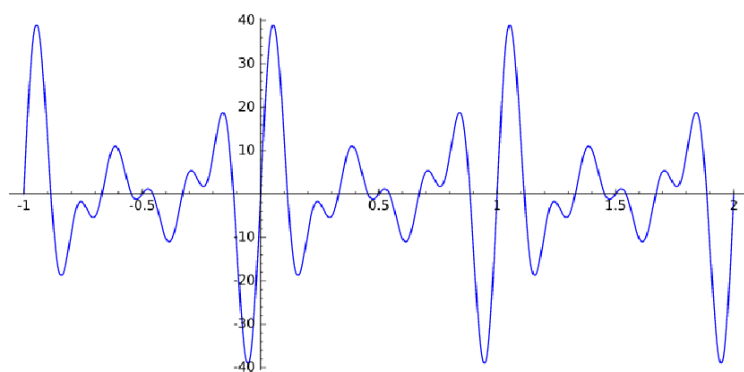
Write a Program to obtain Fourier Transformation Plot for the function

$$f_1(t) = 12 \sin(2\pi(3t)) + 11 \sin(2\pi(4t)) + 10 \sin(2\pi(5t)) + 9 \sin(2\pi(6t))$$

Coding

```
from sage.plot.bar _chart import BarChart
var('t')
f1(t) = 12*sin(2*pi*(3*t)) + 11*sin(2*pi*(4*t)) + 10*sin(2*pi*(5*t)) + 9*sin(2*pi*(6*t))
plot(f1, (t, -1, 2))
```

Output



3.1 Fourier Transform

Fourier transform of $f(x)$ is denoted by $\bar{f}(s)$ or $F\{f(x)\}$.

F is the Fourier transform operator.

Fourier transform of $f(x)$

$$\bar{f}(s) = F\{f(x)\} = \int_{-\infty}^{\infty} f(x) e^{-isx} dx$$

Example on how to calculate Fourier Transform for a given function

Example 1

Find the Fourier transform of $f(x)$, defined as $f(x) = \begin{cases} 1, & \text{for } |x| < a \\ 0, & \text{for } |x| > a \end{cases}$

and hence find the value of $\int_0^{\infty} \frac{\sin x}{x} dx$.

Solution:

$$\begin{aligned} F\{f(x)\} &= \int_{-a}^a e^{-isx} dx \\ &= \int_{-a}^a (\cos sx - i \sin sx) dx \\ &= 2 \int_0^a \cos sx dx, \quad [\text{by the property of definite integrals}] \\ &= \frac{2}{s} \sin as \end{aligned}$$

Taking Fourier inverse transforms,

$$F^{-1}\left\{\frac{2}{s} \sin as\right\} = f(x)$$

i.e.,

$$\frac{1}{2\pi} \int_{-\infty}^{\infty} \frac{2}{s} \sin ase^{ixs} ds = f(x)$$

i.e.,

$$\frac{1}{\pi} \int_{-\infty}^{\infty} \frac{1}{s} \sin as (\cos xs + i \sin xs) ds = f(x)$$

i.e.,

$$\frac{2}{\pi} \int_0^{\infty} \frac{1}{s} \sin as \cos xs ds = f(x) \quad \left[\text{Since, } \frac{1}{s} \sin as \sin xs \text{ is odd} \right]$$

i.e.,

$$\int_0^{\infty} \frac{1}{s} \sin as \cos xs ds = \begin{cases} \frac{\pi}{2}, & \text{for } |x| < a \\ 0, & \text{for } |x| > a \end{cases}$$

Substituting $a = 1$ and $x = 0$, so that $|0| < 1$, we get

$$\int_0^{\infty} \frac{\sin s}{s} ds = \frac{\pi}{2}$$

Changing the dummy variable s into x , we get

$$\int_0^{\infty} \frac{\sin x}{x} dx = \frac{\pi}{2}$$

Example on how to calculate Fourier Transform for Unit Step Function and Unit Impulse Function

Example 2

Find the Fourier transform of the unit step function and unit impulse function.

Solution:

(i) The unit step function is defined as

$$u_a(x) = \begin{cases} 0, & \text{for } x < a \\ 1, & \text{for } x \geq a \end{cases}$$

$$\therefore F\{u_a(x)\} = \int_a^{\infty} e^{-isx} dx = \left[\frac{e^{-isx}}{-is} \right]_a^{\infty} = \frac{1}{is} e^{-ias}$$

In particular

$$F\{u_0(x)\} = \frac{1}{is} \text{ or } \frac{-i}{s}$$

(ii) The unit impulse function or Dirac Delta function $\delta_a(x)$ is defined as $\lim_{\epsilon \rightarrow 0}[f(x)]$, where

$$\begin{aligned}
 f(x) &= \begin{cases} \frac{1}{\epsilon}, & \text{for } a - \frac{\epsilon}{2} \leq x \leq a + \frac{\epsilon}{2} \\ 0, & \text{elsewhere} \end{cases} \\
 F\{f(x)\} &= \int_{a - \frac{\epsilon}{2}}^{a + \frac{\epsilon}{2}} \frac{1}{\epsilon} e^{-isx} dx \\
 &= \frac{1}{\epsilon} \left[\frac{e^{-isx}}{-is} \right]_{a - \frac{\epsilon}{2}}^{a + \frac{\epsilon}{2}} \\
 &= \frac{1}{i\epsilon s} \left\{ e^{-is(a - \frac{\epsilon}{2})} - e^{-is(a + \frac{\epsilon}{2})} \right\} \\
 &= e^{-ias} \left[\frac{\sin \left[\frac{\epsilon s}{2} \right]}{\left[\frac{\epsilon s}{2} \right]} \right] \\
 \therefore F\{\delta_a(x)\} &= \lim_{\epsilon \rightarrow 0} \left[e^{-ias} \cdot \left\{ \frac{\sin \left[\frac{\epsilon s}{2} \right]}{\frac{\epsilon s}{2}} \right\} \right]
 \end{aligned}$$

In particular

$$F\{\delta_a(x)\} = 1$$

3.2 Inverse Fourier Transform

Inverse Fourier transform of $\tilde{f}(s)$

$$F^{-1}\{\tilde{f}(s)\} = \frac{1}{2\pi} \int_{-\infty}^{\infty} \tilde{f}(s) e^{ixs} ds$$

Some authors define the Fourier transform pair as:

$$F\{f(x)\} = \tilde{f}(s) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) e^{-isx} dx \quad \text{and}$$

$$F^{-1}\{\tilde{f}(s)\} = f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \tilde{f}(s) \cdot e^{ixs} ds$$

Example on how to calculate Inverse Fourier Transform for a given function

Example 3

Find the inverse Fourier transform of $\tilde{f}(s)$ given by

$$\tilde{f}(s) = \begin{cases} a - |s|, & \text{for } |s| \leq a \\ 0, & \text{for } |s| > a. \end{cases}$$

Hence show that $\int_0^\infty \frac{\sin^2 x}{x^2} dx = \frac{\pi}{2}$.

Solution:

$$\begin{aligned} F^{-1}\{\tilde{f}(s)\} &= \frac{1}{2\pi} \int_{-\infty}^{\infty} \tilde{f}(s) e^{ixs} ds \\ &= \frac{1}{2\pi} \int_{-a}^a \{a - |s|\} (\cos xs + i \sin xs) ds \\ &= \frac{1}{\pi} \int_0^a (a - s) \cos xs ds \\ &\quad \text{[Since, } \{a - |s|\} \sin xs \text{ is odd]} \\ &= \frac{1}{\pi} \left[(a - s) \frac{\sin xs}{x} - \frac{\cos xs}{x^2} \right]_0^a \\ &= \frac{1}{\pi x^2} (1 - \cos ax) \\ &= \frac{a^2}{2\pi} \left[\frac{\sin \frac{ax}{2}}{\frac{ax}{2}} \right]^2 \end{aligned}$$

$$\therefore F \left[\frac{a^2}{2\pi} \left[\frac{\sin \frac{ax}{2}}{\frac{ax}{2}} \right]^2 \right] = \tilde{f}(s)$$

i.e.,

$$\frac{a^2}{2\pi} \int_{-\infty}^{\infty} \left[\frac{\sin \frac{ax}{2}}{\frac{ax}{2}} \right]^2 e^{-isx} dx = \begin{cases} a - |s|, & \text{for } |s| \leq a \\ 0, & \text{for } |s| > a \end{cases}$$

Taking $a = 2$ and letting $s \rightarrow 0$, we get

$$\int_{-\infty}^{\infty} \left[\frac{\sin x}{x} \right]^2 dx = \pi$$

Since the integrand is an even function of x , we get

$$\int_0^{\infty} \left[\frac{\sin x}{x} \right]^2 dx = \frac{\pi}{2}$$

3.3 Fourier Cosine Transform

Fourier cosine transform of $f(x)$ is denoted by $\bar{f}_C(s)$ or $F_C\{f(x)\}$.

F_C is the Fourier cosine transform operator.

Fourier cosine transform of $f(x)$

$$\bar{f}_C(s) = F_C\{f(x)\} = \int_0^{\infty} f(x) \cos sx \, dx$$

Example on how to calculate Fourier Cosine Transform for a given function

Example 4

Find the Fourier transform of $e^{-a^2 x^2}$. Hence

- (i) Prove that $e^{-\frac{x^2}{2}}$ is self-reciprocal with respect to Fourier Transforms; and
- (ii) Find the Fourier cosine transform of e^{-x^2}

Solution:

$$\begin{aligned} F\{e^{-a^2 x^2}\} &= \int_{-\infty}^{\infty} e^{-a^2 x^2} \cdot e^{-isx} \, dx \\ &= \int_{-\infty}^{\infty} e^{-\left[ax + \left(\frac{is}{2a}\right)^2\right]} \cdot e^{\frac{-s^2}{4a^2}} \, dx \\ &= e^{\frac{-s^2}{4a^2}} \cdot \frac{1}{a} \int_{-\infty}^{\infty} e^{-t^2} \, dt, \\ &\quad \text{[on substituting } sx + \frac{is}{2a} = t] \\ &= \frac{\sqrt{\pi}}{a} e^{\frac{-s^2}{4a^2}} \end{aligned} \tag{1}$$

(i) Had we assumed the definition of the Fourier transform as

$$F\{f(x)\} = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) e^{isx} dx$$

(1) would have become

$$F\{e^{-a^2 x^2}\} = \frac{1}{a\sqrt{2}} e^{\frac{-s^2}{4a^2}}$$

Substituting $a = \frac{1}{\sqrt{2}}$ in (2), we get

$$F\{e^{\frac{-x^2}{2}}\} = e^{\frac{-s^2}{2}}$$

and so

$$F^{-1}\{e^{\frac{-s^2}{2}}\} = e^{\frac{-x^2}{2}}$$

i.e., $e^{\frac{-x^2}{2}}$ is reciprocal with respect to Fourier transforms.

(ii) From (1), we have

$$\int_{-\infty}^{\infty} e^{-a^2 x^2} (\cos sx - i \sin sx) dx = \frac{\sqrt{\pi}}{2a} e^{\frac{-s^2}{4a^2}}$$

Equating the real parts on both sides, we get

$$\int_0^{\infty} e^{-a^2 x^2} \cos sx dx = \frac{\sqrt{\pi}}{2a} e^{\frac{-s^2}{4a^2}}$$

or

$$F_C\{e^{-a^2 x^2}\} = \frac{\sqrt{\pi}}{2a} e^{\frac{-s^2}{4a^2}}$$

3.4 Inverse Fourier Cosine Transform

Inverse Fourier cosine transform of $\bar{f}_C(s)$

$$F_C^{-1}\{\bar{f}_C(s)\} = \frac{2}{\pi} \int_0^{\infty} \bar{f}_C(s) \cos xs ds$$

Some authors define the Fourier cosine transform pair as:

$$F_C\{f(x)\} = \bar{f}_C(s) = \sqrt{\frac{2}{\pi}} \int_0^{\infty} f(x) \cos sx \, dx \quad \text{and}$$

$$F_C^{-1}\{\bar{f}_C(s)\} = f(x) = \sqrt{\frac{2}{\pi}} \int_0^{\infty} \bar{f}_C(s) \cos xs \, ds$$

3.5 Fourier Sine Transform

Fourier sine transform of $f(x)$ is denoted by $\bar{f}_S(s)$ or $F_S\{f(x)\}$.

F_S is the Fourier sine transform operator.

Fourier sine transform of $f(x)$

$$\bar{f}_S(s) = F_S\{f(x)\} = \int_0^{\infty} f(x) \sin sx \, dx$$

Example on how to calculate Fourier Sine Transform for a given function

Example 5

Find the Fourier Sine Transform of e^{-ax} ($a > 0$). Hence find $F_S\{xe^{-ax}\}$ and $F_S\{\frac{e^{-ax}}{x}\}$. Deduce the value of

$$\int_0^{\infty} \frac{\sin sx}{x} \, dx.$$

Solution:

$$\begin{aligned} F_S(e^{-ax}) &= \int_0^{\infty} e^{-ax} \sin sx \, dx \\ &= \left[\frac{e^{-ax}}{s^2 + a^2} (-a \sin sx - s \cos sx) \right]_0^{\infty} \\ &= \frac{s}{s^2 + a^2} \end{aligned}$$

i.e.,

$$\int_0^{\infty} e^{-ax} \sin sx \, dx = \frac{s}{s^2 + a^2} \quad (1)$$

Differentiating both sides of (1) with respect to a , we get

$$\int_0^{\infty} -x e^{-ax} \sin sx \, dx = -\frac{2as}{(s^2 + a^2)^2}$$

i.e.,

$$F_S(x e^{-ax}) = \frac{2as}{(s^2 + a^2)^2}$$

Integrating both sides of (1) with respect to a between a and ∞ ,

$$\int_0^{\infty} \left(\frac{e^{-ax}}{-x} \right)_0^{\infty} \sin sx \, dx = \left[-\cot^{-1} \left(\frac{a}{s} \right) \right]_a^{\infty}$$

i.e.,

$$\int_0^{\infty} \left(\frac{e^{-ax}}{x} \right) \sin sx \, dx = \cot^{-1} \left(\frac{a}{s} \right)$$

i.e.,

$$F_S \left(\frac{e^{-ax}}{x} \right) = \cot^{-1} \left(\frac{a}{s} \right), \quad a > 0 \quad (2)$$

Taking limits on both sides of (2) as $a \rightarrow 0$, we get

$$F_S \left(\frac{1}{x} \right) = \cot^{-1}(0) = \frac{\pi}{2}$$

Thus

$$\int_0^{\infty} \frac{\sin sx}{x} = \frac{\pi}{2}, \quad s > 0.$$

3.6 Inverse Fourier Sine Transform

Inverse Fourier sine transform of $\bar{f}_S(s)$

$$F_S^{-1}\{\bar{f}_S(s)\} = \frac{2}{\pi} \int_0^{\infty} \bar{f}_S(s) \sin xs \, ds$$

Some authors define the Fourier sine transform pair as:

$$F_S\{f(x)\} = \bar{f}_S(s) = \sqrt{\frac{2}{\pi}} \int_0^{\infty} f(x) \sin sx \, dx \quad \text{and}$$

$$F_S^{-1}\{\bar{f}_S(s)\} = f(x) = \sqrt{\frac{2}{\pi}} \int_0^{\infty} \bar{f}_S(s) \sin xs \, ds$$

4.1 Fourier Integral Theorem

If $f(x)$ is piecewise continuous, has piecewise continuous derivatives in every finite interval in $(-\infty, \infty)$ and absolutely integrable in $(-\infty, \infty)$, then

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(t) \cdot e^{is(x-t)} \, dt \, ds \quad \text{or equivalently}$$

$$f(x) = \frac{1}{\pi} \int_0^{\infty} \int_{-\infty}^{\infty} f(t) \cos\{s(x-t)\} \, dt \, ds.$$

Proof:

When $f(x)$ satisfies the conditions given in the theorem, we can prove that $f(x)$ can be expanded as a infinite series of the form.

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{\frac{in\pi x}{l}} \quad (1)$$

in $(-l, l)$ however large l may be, where

$$c_n = \frac{1}{2l} \int_{-l}^l f(t) e^{\frac{-in\pi t}{l}} \, dt \quad (2)$$

Substituting $s_n = \frac{n\pi}{l}$ and inserting (2) in (1), we have

$$\begin{aligned} f(x) &= \sum_{n=-\infty}^{\infty} \frac{1}{2l} \int_{-l}^l f(t) e^{is_n(x-t)} \, dt \\ &= \frac{1}{2\pi} \int_{-l}^l \left[\sum_{n=-\infty}^{\infty} f(t) e^{is_n(x-t)} \cdot \frac{\pi}{l} \right] \, dt \end{aligned}$$

on interchanging summation and integration.

$$= \frac{1}{2\pi} \int_{-l}^l \left[\sum_{n=-\infty}^{\infty} f(t) e^{is_n(x-t)} \Delta S_n \right] dt$$

Since

$$\Delta S_n = s_{n+1} - s_n = \frac{(n+1)\pi}{l} - \frac{n\pi}{l} = \frac{\pi}{l}$$

Taking limits as $\Delta S_n \rightarrow 0$ or equivalently $l \rightarrow \infty$ we get

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} f(t) e^{is(x-t)} ds \right] dt \quad (3)$$

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(t) e^{is(x-t)} dt ds \quad (4)$$

[Since, the limits of integration are constants]

From (3)

$$\begin{aligned} f(x) &= \frac{1}{2\pi} \int_{-\infty}^{\infty} f(t) \int_{-\infty}^{\infty} [\cos s(x-t) + i \sin s(x-t)] ds dt \\ &= \frac{1}{2\pi} \int_{-\infty}^{\infty} f(t) \cdot 2 \int_0^{\infty} \cos s(x-t) ds dt \end{aligned}$$

[Since, $\cos s(x-t)$ is an even function and $\sin s(x-t)$ is an odd function of s in $(-\infty, \infty)$]

$$f(x) = \frac{1}{\pi} \int_0^{\infty} \int_{-\infty}^{\infty} f(t) \cos s(x-t) dt ds \quad (5)$$

[Since, the limits of the integration are constants]

From (5), we have

$$\begin{aligned} f(x) &= \frac{1}{\pi} \int_0^{\infty} \int_{-\infty}^{\infty} f(t) [\cos sx \cos st + \sin sx \cdot \sin st] dt ds \\ &= \frac{1}{\pi} \int_0^{\infty} \cos sx \left(\int_{-\infty}^{\infty} f(t) \cos st dt \right) ds + \frac{1}{\pi} \int_0^{\infty} \sin sx \left(\int_{-\infty}^{\infty} f(t) \sin st dt \right) ds \quad (6) \end{aligned}$$

If $f(x)$ [or $f(t)$] is even,

$f(t) \cos st$ is an even function of t and $f(t) \sin st$ is an odd function of t . Hence, by the property of definite integrals, we get the following from (6)

$$f(x) = \frac{2}{\pi} \int_0^{\infty} \int_0^{\infty} f(t) \cos sx \cos st dt ds \quad (7)$$

The R.H.S of (7) is called the Fourier Cosine Integral of $f(x)$, provided $f(x)$ is even.

If $f(x)$ [or $f(t)$] is odd, $f(t) \cos st$ is an odd function of t and $f(t) \sin st$ is an even function of t .

Hence, by the property of definite integrals, we get the following from (6)

$$f(x) = \frac{2}{\pi} \int_0^{\infty} \int_0^{\infty} f(t) \sin sx \sin st dt ds \quad (8)$$

The R.H.S of (8) is called the Fourier Sine Integral of $f(x)$, provided $f(x)$ is odd.

Example on how to find Fourier Integral Representation for a given function using Fourier Integral Theorem

Example 6

Find the Fourier integral representation of $f(x)$ defined as

$$f(x) = \begin{cases} 0, & \text{for } x < 0 \\ \frac{1}{2}, & \text{for } x = 0 \\ e^{-x}, & \text{for } x > 0 \end{cases}$$

Verify the representation at $x = 0$.

Solution:

Fourier (complex) integral representation is given by

$$\begin{aligned}
 f(x) &= \frac{1}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(t) e^{-ist} e^{isx} dt ds \\
 &= \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{isx} \left[\int_{-\infty}^0 + \int_0^{\infty} f(t) e^{-ist} dt \right] ds \\
 &= \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{isx} \left[\int_0^{\infty} e^{-(1+is)t} dt \right] ds \quad [\text{on using the given values of } f(t)] \\
 &= \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{isx} \left\{ \frac{e^{-(1+is)t}}{-(1+is)} \right\}_{t=0}^{t=\infty} ds \\
 &= \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{isx} \cdot \frac{1}{1+is} ds \\
 &= \frac{1}{2\pi} \int_{-\infty}^{\infty} \frac{(1-is)}{1+s^2} (\cos xs + i \sin xs) ds \\
 &= \frac{1}{2\pi} \int_{-\infty}^{\infty} \frac{1}{1+s^2} [\{\cos xs + s \sin xs\} + i\{\sin xs - s \cos xs\}] \\
 &= \frac{1}{\pi} \int_0^{\infty} \left(\frac{\cos xs + s \sin xs}{1+s^2} \right) ds \tag{1}
 \end{aligned}$$

by property of definite integrals, as the real part is even and the imaginary part is odd.

Substituting $x = 0$ in the integral representation (1), we get

$$f(0) = \frac{1}{\pi} \int_0^{\infty} \frac{ds}{1+s^2} = \frac{1}{\pi} \left[\tan^{-1} s \right]_0^{\infty} = \frac{1}{2}$$

Thus the integral representation (1) holds good for $x = 0$ also.

4.2 Alternative Form of Fourier Complex Integral Formula

The Fourier integral formula for $f(x)$ is

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(t) e^{is(t-x)} dt ds$$

Proof:

$$\begin{aligned} f(x) &= \frac{1}{\pi} \int_0^{\infty} \int_{-\infty}^{\infty} f(t) \cos s(x-t) dt ds \\ &= \frac{1}{\pi} \int_0^{\infty} \int_{-\infty}^{\infty} f(t) \cos s(t-x) dt ds \\ &= \frac{1}{2\pi} \int_0^{\infty} \int_{-\infty}^{\infty} f(t) \left[e^{is(t-x)} + e^{-is(t-x)} \right] dt ds \\ &= \frac{1}{2\pi} \int_0^{\infty} \int_{-\infty}^{\infty} f(t) e^{is(t-x)} dt ds + \frac{1}{2\pi} \int_0^{\infty} \int_{-\infty}^{\infty} f(t) e^{-is(t-x)} dt ds \end{aligned}$$

Substituting $s = -s'$ in the second integral, we get

$$\begin{aligned} f(x) &= \frac{1}{2\pi} \int_0^{\infty} \int_{-\infty}^{\infty} f(t) e^{is(t-x)} dt ds + \frac{1}{2\pi} \int_0^0 \int_{-\infty}^{\infty} f(t) e^{is'(t-x)} dt ds' \\ &= \frac{1}{2\pi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(t) e^{is(t-x)} dt ds \end{aligned}$$

[on changing s' into s and combining the two integrals] (1)

(1) provides an alternative formula for $f(x)$. Comparing this with the Fourier Complex integral formula derived in (4) of Fourier integral theorem, we note that x and t can be interchanged in the exponential function.

4.3 Standard Fourier Transform Pairs

S.No.	$x(t)$	$X(f)$	$X(\omega)$
1.	$\delta(t)$	1	1
2.	$rect(t)$	$sinc(f)$	$sinc\left(\frac{\omega}{2\pi}\right)$
3.	$tri(t)$	$sinc^2(f)$	$sinc^2\left(\frac{\omega}{2\pi}\right)$
4.	$sinc(t)$	$rect(f)$	$rect\left(\frac{\omega}{2\pi}\right)$
5.	$\cos(2\pi at)$	$\frac{1}{2} [\delta(f+a) + \delta(f-a)]$	$\pi [\delta(\omega+2\pi a) + \delta(\omega-2\pi a)]$
6.	$\sin(2\pi at)$	$\frac{j}{2} [\delta(f+a) - \delta(f-a)]$	$j\pi [\delta(\omega+2\pi a) - \delta(\omega-2\pi a)]$
7.	$e^{-at}u(t)$	$\frac{1}{a+j2\pi f}$	$\frac{1}{a+j\omega}$
8.	$t^n e^{-at}u(t)$	$\frac{1}{(a+j2\pi f)^{n+1}}$	$\frac{1}{(a+j\omega)^{n+1}}$
9.	$e^{-a t }$	$\frac{2a}{a^2+4\pi^2 f^2}$	$\frac{2a}{a^2+\omega^2}$
10.	$e^{-\pi t^2}$	$e^{-\pi f^2}$	$e^{-\frac{\omega^2}{4\pi}}$
11.	$sgn(t)$	$\frac{1}{j\pi f}$	$\frac{2}{j\omega}$
12.	$u(t)$	$\frac{1}{2}\delta(f) + \frac{1}{j2\pi f}$	$\pi\delta(\omega) + \frac{1}{j\omega}$
13.	$e^{-at}\cos 2\pi btu(t)$	$\frac{a+j2\pi f}{(a+j2\pi f)^2+(2\pi b)^2}$	$\frac{a+j\omega}{(a+j\omega)^2+(2\pi b)^2}$
14.	$e^{-at}\sin 2\pi btu(t)$	$\frac{2\pi b}{(a+j2\pi f)^2+(2\pi b)^2}$	$\frac{2\pi b}{(a+j\omega)^2+(2\pi b)^2}$
15.	$e^{at}u(-t)$	$\frac{1}{a-j2\pi f}$	$\frac{1}{a-j\omega}$

4.4 Deriving Fourier Transform from Fourier Series

$$x(t) = \sum_{k=-\infty}^{\infty} X[k] e^{jk\omega t}$$

$$x(t) = \sum_{k=-\infty}^{\infty} X[k] e^{jk\frac{2\pi}{T}t} \quad \left[\text{Since, } \omega = \frac{2\pi}{T} \right] \quad (1)$$

$$\text{Let } \Delta f = \frac{1}{T}$$

$$(1) \Rightarrow x(t) = \sum_{k=-\infty}^{\infty} X[k] e^{jk2\pi\Delta f t}$$

$$X[k] = \frac{1}{T} \int_{t_0}^{t_0+T} x(t) e^{-jk\omega t} dt$$

$$X[k] = \frac{1}{T} \int_{t_0}^{t_0+T} x(t) e^{jk2\pi\Delta f t} dt \quad (2)$$

where,
$$X(\omega) = \int_{-\infty}^{\infty} x(t) e^{-j\omega t} dt$$

The above equation is known as Fourier Transform Equation.
Substituting equation (2) in equation (1), we get,

$$x(t) = \sum_{k=-\infty}^{\infty} \left[\Delta f \int_{t_0}^{t_0+T} x(t) e^{-j2\pi k\Delta f t} dt \right] \cdot e^{jk2\pi\Delta f t}$$

$$\text{Let } t_0 = -\frac{T}{2}$$

$$x(t) = \lim_{T \rightarrow \infty} \sum_{k=-\infty}^{\infty} \left[\int_{-\frac{T}{2}}^{\frac{T}{2}} x(t) e^{-j2\pi k\Delta f t} dt \right] \cdot e^{jk2\pi\Delta f t}$$

When $T \rightarrow \infty$, $\sum \rightarrow \int$, $\Delta f \rightarrow df$, $k\Delta f \rightarrow f$, continuous variable function.

$$x(t) = \int_{-\infty}^{\infty} df \int_{-\infty}^{\infty} x(t) e^{-j2\pi f t} dt \cdot e^{j2\pi f t}$$

$$x(t) = \int_{-\infty}^{\infty} [X(\omega)] e^{j\omega t} d\omega$$

The above equation is known as Fourier Inverse Transform Equation.

4.5 Relationship between Fourier Transform and Laplace Transform

$$\text{Let } f(t) \text{ be defined as } f(t) = \begin{cases} e^{-xt}\phi(t), & t \geq 0 \\ 0, & t < 0 \end{cases}$$

Then

$$F\{f(t)\} = \int_{-\infty}^{\infty} f(t)e^{-iyt} dt$$

where y is the Fourier transform variable.

$$\begin{aligned} &= \int_{-\infty}^0 0 \cdot e^{-iyt} dt + \int_0^{\infty} e^{-xt}\phi(t)e^{-iyt} dt \\ &= \int_0^{\infty} e^{-st}\phi(t) dt \quad [\text{where } s = x + iy] \end{aligned}$$

$$\text{i.e., } \boxed{F\{f(t)\} = L\{\phi(t)\}}$$

4.6 PROPERTIES OF FOURIER TRANSFORM

Property 1 - Linearity Property

F is a linear operator, i.e. $F[(c_1 f_1(x) + c_2 f_2(x))] = c_1 F\{f_1(x)\} + c_2 F\{f_2(x)\}$, where c_1 and c_2 are constants.

Proof:

$$\begin{aligned} F[c_1 f_1(x) + c_2 f_2(x)] &= \int_{-\infty}^{\infty} [c_1 f_1(x) + c_2 f_2(x)] e^{-isx} dx \\ &= c_1 \int_{-\infty}^{\infty} f_1(x) e^{-isx} dx + c_2 \int_{-\infty}^{\infty} f_2(x) e^{-isx} dx \\ &= c_1 F\{f_1\}(x) + c_2 F\{f_2(x)\} \end{aligned}$$

Property 2 - Change of Scale Property

If $F\{f(x)\} = \bar{f}(s)$, then $F\{f(ax)\} = \frac{1}{|a|} \bar{f}\left[\frac{s}{a}\right]$

Proof:

$$\begin{aligned} F\{f(ax)\} &= \int_{-\infty}^{\infty} f(ax) e^{-isx} dx \\ &= \int_{-\infty}^{\infty} f(t) e^{\frac{-ist}{a}} \cdot \frac{dt}{a}, \quad [\text{on substituting } ax = t \text{ and assuming that } a > 0.] \\ &= \frac{1}{a} \bar{f}\left[\frac{s}{a}\right] \end{aligned}$$

But

$$\begin{aligned} F\{f(ax)\} &= \int_{\infty}^{-\infty} f(t) e^{\frac{-ist}{a}} \cdot \frac{dt}{a}, \quad \text{if } a < 0 \\ &= -\frac{1}{a} \bar{f}\left[\frac{s}{a}\right] \\ \therefore F\{f(ax)\} &= \frac{1}{|a|} \bar{f}\left[\frac{s}{a}\right] \end{aligned}$$

Similarly,

$$F_C\{f(ax)\} = \frac{1}{a} \cdot \bar{f}_C\left[\frac{s}{a}\right]$$

and

$$F_S\{f(ax)\} = \frac{1}{a} \cdot \bar{f}_S\left[\frac{s}{a}\right]$$

Property 3 - Shifting Property (Shifting in x)

If $F\{f(x)\} = \bar{f}(s)$, then $F\{f(x-a)\} = e^{-ias} \bar{f}(s)$

Proof:

$$\begin{aligned} F\{f(x-a)\} &= \int_{-\infty}^{\infty} f(x-a) e^{-isx} dx \\ &= \int_{-\infty}^{\infty} f(t) e^{-is(t+a)} dt, \quad [\text{on substituting } t = x-a] \\ &= e^{-ias} \bar{f}(s) \end{aligned}$$

Property 4 - Shifting in Respect of s

If $F\{f(x)\} = \bar{f}(s)$, then $F\{e^{-iax} f(x)\} = \bar{f}(s+a)$

Proof:

$$\begin{aligned} F\{e^{-iax} f(x)\} &= \int_{-\infty}^{\infty} e^{-iax} f(x) e^{-isx} dx \\ &= \int_{-\infty}^{\infty} f(x) e^{-i(s+a)x} dx \\ &= \bar{f}(s+a) \end{aligned}$$

$$\begin{aligned} F\{e^{iax} f(x)\} &= \int_{-\infty}^{\infty} e^{iax} f(x) e^{-isx} dx \\ &= \int_{-\infty}^{\infty} f(x) e^{-i(s-a)x} dx \\ &= \bar{f}(s-a) \end{aligned}$$

Property 5 - Modulation Theorem

If $F\{f(x)\} = \bar{f}(s)$, then $F\{f(x) \cos ax\} = \frac{1}{2} [\bar{f}(s+a) + \bar{f}(s-a)]$

Proof:

$$\begin{aligned} F\{f(x) \cos ax\} &= \frac{1}{2} F\left[f(x)(e^{iax} + e^{-iax})\right] \\ &= \frac{1}{2} \left[F\{f(x) e^{iax}\} + F\{f(x) e^{-iax}\}\right] \\ &= \frac{1}{2} [\bar{f}(s-a) + \bar{f}(s+a)] \end{aligned}$$

Property 6 - Conjugate Symmetry Property

If $F\{f(x)\} = \bar{f}(s)$, then $F\{f^*(-x)\} = [\bar{f}(s)]^*$, where $*$ denotes complex conjugate.

Proof:

$$\begin{aligned} \bar{f}(s) &= \int_{-\infty}^{\infty} f(x) e^{-isx} dx \\ \therefore [\bar{f}(s)]^* &= \int_{-\infty}^{\infty} f^*(x) e^{isx} dx \\ &= \int_{-\infty}^{\infty} f^*(-t) e^{-ist} dt, & \text{[On Substituting } x = -t] \\ &= F\{f^*(-x)\} \end{aligned}$$

Property 7 - Transform of Derivatives

If $f(x)$ is continuous, $f'(x)$ is piecewise continuously differentiable, $f(x)$ and $f'(x)$ are absolutely integrable in $(-\infty, \infty)$ and $\lim_{x \rightarrow \pm\infty} [f(x) = 0]$, then $F\{f'(x)\} = is \bar{f}(s)$ where $\bar{f}(s) = F\{f(x)\}$

Proof:

By the first three conditions given, $F\{f(x)\}$ and $F\{f'(x)\}$ exist.

$$\begin{aligned} F\{f'(x)\} &= \int_{-\infty}^{\infty} f'(x) e^{-isx} dx \\ &= \left[e^{-isx} f(x) \right]_{-\infty}^{\infty} + is \int_{-\infty}^{\infty} e^{-isx} f(x) dx, \\ &\quad \text{[on substituting by parts]} \\ &= 0 + is F\{f(x)\}, \quad \text{[by the given condition]} \\ &= is \bar{f}(s) \end{aligned}$$

Example on how to solve Differential Equations using Fourier Transforms

Example 7

Solve the differential equation

$$\frac{d^2 y}{dx^2} + 3 \frac{dy}{dx} + 2y = e^{-x}, \quad x > 0$$

using Fourier transforms, given that $y(0) = 0$ and $y'(0) = 0$.

Solution:

Taking Fourier complex transforms on both sides of the given differential equation, we have

$$\begin{aligned} (is)^2 \bar{y}(s) + 3(is) \bar{y}(s) + 2 \bar{y}(s) &= F(e^{-x}), \quad x > 0 \\ &= F\{U(x) \cdot e^{-x}\}, \end{aligned}$$

[where $U(x)$ is the unit step function]

i.e.,

$$\begin{aligned}
[(is)^2 + 3(is) + 2] \bar{y}(s) &= \int_0^{\infty} e^{-(1+is)x} dx = \frac{1}{1+is} \\
\therefore \bar{y}(s) &= \frac{1}{(is+1)^2(is+2)} \\
&= \frac{-1}{is+1} + \frac{1}{(is+1)^2} + \frac{1}{is+2} \quad [\text{by partial fractions.}] \\
\therefore y &= -F^{-1}\left\{\frac{1}{is+1}\right\} + F^{-1}\left\{\frac{1}{(is+1)^2}\right\} + \left\{\frac{1}{is+2}\right\} \\
&= -U(x)e^{-x} + U(x).xe^{-x} + U(x).e^{-2x}
\end{aligned}$$

Since

$$\begin{aligned}
F\{U(x)xe^{-x}\} &= \int_0^{\infty} xe^{-(1+is)x} dx \\
&= \left[x \left\{ \frac{e^{-(1+is)x}}{-(1+is)} \right\} - \left\{ \frac{e^{-(1+is)x}}{(1+is)^2} \right\} \right]_0^{\infty} \\
&= \frac{1}{(1+is)^2}
\end{aligned}$$

i.e.,

$$y = -e^{-x} + xe^{-x} + e^{-2x}, x > 0$$

Example on how to solve Partial Differential Equations using Transform of Derivatives and Fourier Cosine Transform

Example 8

Solve the equation $\frac{\partial u}{\partial t} = \alpha^2 \frac{\partial^2 u}{\partial x \partial x}$, satisfying the boundary conditions $\frac{\partial u}{\partial x}(0, t) = k$, $t \geq 0$ and $u(x, t) \rightarrow 0$ as $x \rightarrow \infty$ and the initial condition $u(x, 0) = 0$.

Solution:

We know that,

If $f(0, y)$ is given but $\frac{\partial f}{\partial x}(0, y)$ is not known in a boundary value problem, Fourier sine transform is used. On the other hand, if $\frac{\partial f}{\partial x}(0, y)$ is given but $f(0, y)$ is not known, Fourier cosine transform is used.

Since $x > 0$ and $\frac{\partial u}{\partial x}(0, t)$ is given, we take Fourier cosine transforms of the equation with respect to x .

Thus

$$\frac{\partial}{\partial t} \bar{u}_C(s, t) = \alpha^2 \left[-s^2 \bar{u}_C(s, t) - \frac{\partial u}{\partial x}(0, t) \right]$$

i.e.,

$$\frac{\partial}{\partial t} \bar{u}_C(s, t) + \alpha^2 s^2 \bar{u}_C(s, t) = k\alpha^2 \quad (1)$$

Transform of the initial condition is

$$\bar{u}_C(s, 0) = 0 \quad (2)$$

Solving (1) and using (2), we get

$$\bar{u}_C(s, t) = A e^{-\alpha^2 s^2 t} - \frac{k}{s^2} \quad (3)$$

Using (2) in (3), we get $A = \frac{k}{s^2}$

$$\therefore \bar{u}_C(s, t) = \frac{k}{s^2} (e^{-\alpha^2 s^2 t} - 1)$$

Taking the inverse cosine transforms, we get

$$u(x, t) = \frac{2k}{\pi} \int_0^\infty \frac{1}{s^2} (e^{-\alpha^2 s^2 t} - 1) \cos xs \, ds$$

Property 8 - Derivatives of the Transform

If $F\{f(x)\} = \bar{f}(s)$, then $-iF\{xf(x)\} = \frac{d}{ds} \bar{f}(s)$

Proof:

$$\begin{aligned} \bar{f}(s) &= \int_{-\infty}^{\infty} e^{-isx} f(x) \, dx \\ \frac{d}{ds} \bar{f}(s) &= \int_{-\infty}^{\infty} \frac{d}{ds} [e^{-isx} f(x)] \, dx \\ &= (-i) \int_{-\infty}^{\infty} e^{-isx} [xf(x)] \, dx \\ &= -i \cdot F\{xf(x)\} \end{aligned}$$

Extending, we get

$$\frac{d^r}{ds^r} \bar{f}(s) = (-i)^r F\{x^r f(x)\}$$

Convolution Product

$$\int_{-\infty}^{\infty} f(x-u)g(u) du$$

is called the convolution product or simply the convolution of the functions $f(x)$ and $g(x)$ and is denoted by $f(x) * g(x)$.

Property 9 - Convolution Theorem

The Fourier transform of the convolution of two functions is the product of their Fourier transforms.

i.e., if $F\{f(x)\} = \bar{f}(s)$ and $F\{g(x)\} = \bar{g}(s)$, then

$$F\{f(x) * g(x)\} = \bar{f}(s) \cdot \bar{g}(s)$$

Proof:

$$\begin{aligned} F\{f(x) * g(x)\} &= \int_{-\infty}^{\infty} f(x) * g(x) e^{-isx} dx \\ &= \int_{-\infty}^{\infty} \left[\int_{-\infty}^{\infty} f(x-u)g(u) du \right] e^{-isx} dx \\ &= \int_{-\infty}^{\infty} g(u) \left[\int_{-\infty}^{\infty} f(x-u) e^{-isx} dx \right] du \quad [\text{on changing the order of integration.}] \\ &= \int_{-\infty}^{\infty} g(u) [e^{-ius} \bar{f}(s)] du, \quad [\text{by the shifting property.}] \\ &= \bar{f}(s) \cdot \int_{-\infty}^{\infty} g(u) e^{-isu} du \\ &= \bar{f}(s) \cdot \bar{g}(s) \end{aligned}$$

Inverting, we get

$$\begin{aligned} F^{-1}\{\bar{f}(s) \cdot \bar{g}(s)\} &= f(x) * g(x) \\ &= F^{-1}\{\bar{f}(s)\} * F^{-1}\{\bar{g}(s)\} \end{aligned}$$

Property 10 - Parseval's Identity or Energy Theorem

If $F\{f(x)\} = \bar{f}(s)$, then

$$\int_{-\infty}^{\infty} |f(x)|^2 dx = \frac{1}{2\pi} \int_{-\infty}^{\infty} |\bar{f}(s)|^2 ds$$

Proof:

By convolution theorem,

$$f(x) * g(x) = F^{-1}\{\bar{f}(s) \cdot \bar{g}(s)\}$$

i.e.,

$$\int_{-\infty}^{\infty} f(u) \cdot g(x-u) du = \frac{1}{2\pi} \int_{-\infty}^{\infty} \bar{f}(s) \bar{g}(s) e^{ixs} ds \quad (1)$$

Substituting $x = 0$ in (1), we get

$$\int_{-\infty}^{\infty} f(u) g(-u) du = \frac{1}{2\pi} \int_{-\infty}^{\infty} \bar{f}(s) \bar{g}(s) ds \quad (2)$$

(2) is true for any $g(u)$; take $g(u) = [f(-u)]^*$ and hence $g(-u) = [f(u)]^*$, where $[f(u)]^*$ is the complex conjugate of $f(u)$.

Also

$$\bar{g}(s) = F\{g(x)\} = F\{f(-x)\}^* = [Ff(x)]^* = [\bar{f}(s)]^*$$

[by conjugate symmetric property]

Using these in (2), we get

$$\int_{-\infty}^{\infty} f(u) [f(u)]^* du = \frac{1}{2\pi} \int_{-\infty}^{\infty} \bar{f}(s) [\bar{f}(s)]^* ds$$

i.e.,

$$\int_{-\infty}^{\infty} |f(u)|^2 du = \frac{1}{2\pi} \int_{-\infty}^{\infty} |\bar{f}(s)|^2 ds,$$

or

$$\int_{-\infty}^{\infty} |f(x)|^2 dx = \frac{1}{2\pi} \int_{-\infty}^{\infty} |\bar{f}(s)|^2 ds.$$

[on changing the dummy variable]

Example on how to calculate Fourier cosine and sine transforms of a given function using Parseval's Identity

Example 9

Using Parseval's identity for Fourier cosine and sine transforms of e^{-ax} , evaluate

$$(i) \int_0^{\infty} \frac{dx}{(a^2 + x^2)^2} \quad \text{and} \quad (ii) \int_0^{\infty} \frac{x^2}{(a^2 + x^2)^2} dx$$

Solution

$$(i) \quad F_C(e^{-ax}) = \int_0^{\infty} \cos sx \, dx = \frac{a}{s^2 + a^2}$$

By Parseval's identity,

$$\begin{aligned} \int_0^{\infty} |f(x)|^2 \, dx &= \frac{2}{\pi} \int_0^{\infty} |\bar{f}_C(s)|^2 \, ds \\ \int_0^{\infty} e^{-2ax} \, dx &= \frac{2}{\pi} a^2 \int_0^{\infty} \frac{ds}{(s^2 + a^2)^2} \end{aligned}$$

i.e.,

$$\begin{aligned} \int_0^{\infty} \frac{ds}{(s^2 + a^2)^2} &= \frac{\pi}{2a^2} \cdot \left[\frac{e^{-2ax}}{-2a} \right]_0^{\infty} \\ &= \frac{\pi}{4a^3}, \text{ if } a > 0 \end{aligned}$$

Changing the dummy variable s into x , we get the first result.

ii) Now

$$F_S(e^{-ax}) = \int_0^{\infty} e^{-ax} \sin sx \, dx = \frac{s}{s^2 + a^2}$$

By Parseval's identity,

$$\int_0^{\infty} |f(x)|^2 \, dx = \frac{2}{\pi} \int_0^{\infty} |\bar{f}_S(s)|^2 \, ds$$

i.e.,

$$\frac{2}{\pi} \int_0^{\infty} \frac{s^2}{((s^2 + a^2))^2} ds = \int_0^{\infty} e^{-2ax} dx$$

$$\int_0^{\infty} \frac{x^2 dx}{(x^2 + a^2)^2} = \frac{\pi}{4a}, \text{ if } a > 0 \quad [\text{on changing the dummy variables.}]$$

Property 11 - Parseval's Identity for Fourier Cosine and Sine Transforms

If $\bar{f}_C(s)$, $\bar{g}_C(s)$ are the Fourier cosine transforms and $\bar{f}_S(s)$, $\bar{g}_S(s)$ are the Fourier sine transforms of $f(x)$ and $g(x)$ respectively, then

$$(i) \int_0^{\infty} f(x) g(x) dx = \int_0^{\infty} \bar{f}_C(s) \bar{g}_C(s) ds = \int_0^{\infty} \bar{f}_S(s) \bar{g}_S(s) ds$$

$$(ii) \int_0^{\infty} |f(x)|^2 dx = \int_0^{\infty} |\bar{f}_C(s)|^2 ds = \int_0^{\infty} |\bar{f}_S(s)|^2 ds$$

Proof:

$$(i) \int_0^{\infty} \bar{f}_C(s) \bar{g}_C(s) ds = \int_0^{\infty} \bar{f}_C(s) \left[\sqrt{\frac{2}{\pi}} \int_0^{\infty} g(x) \cos sx dx \right] ds$$

$$= \int_0^{\infty} g(x) \left[\sqrt{\frac{2}{\pi}} \int_0^{\infty} \bar{f}_C(s) \cos xs ds \right] dx,$$

(changing the order of integration)

$$= \int_0^{\infty} f(x) g(x) dx$$

(ii) Replacing $g(x) = f^*(x)$ in (i) and noting that $F_C\{f^*(x)\}$

$$= \{\bar{f}_C(s)\}^* \text{ and } F_S\{f^*(x)\} = \{\bar{f}_S(s)\}^*, \text{ we get } \int_0^{\infty} f(x)$$

$$f^*(x) dx = \int_0^{\infty} \bar{f}_C(s) \{\bar{f}_C(s)\}^* ds = \int_0^{\infty} \bar{f}_S(s) \{\bar{f}_S(s)\}^* ds$$

$$\text{i.e., } \int_0^{\infty} |f(x)|^2 dx = \int_0^{\infty} |\bar{f}_C(s)|^2 ds = \int_0^{\infty} |\bar{f}_S(s)|^2 ds$$

Property 12

If $F_C\{f(x)\} = \bar{f}_C(s)$ and $F_S\{f(x)\} = \bar{f}_S(s)$, then

- (i) $\frac{d}{ds}\{\bar{f}_C(s)\} = -F_S\{x f(x)\}$ and
- (ii) $\frac{d}{ds}\{\bar{f}_S(s)\} = F_C\{x f(x)\}.$

Proof:

$$\begin{aligned} (i) \quad \bar{f}_C(s) &= \int_0^{\infty} f(x) \cos sx \, dx \\ \frac{d}{ds}\{\bar{f}_C(s)\} &= \int_0^{\infty} f(x) \{-x \sin sx\} \, dx \\ &= - \int_0^{\infty} \{x f(x)\} \sin sx \, dx \\ &= -F_S\{x f(x)\} \\ (ii) \quad \bar{f}_S(s) &= \int_0^{\infty} f(x) \sin sx \, dx \\ \frac{d}{ds}\{\bar{f}_S(s)\} &= \int_0^{\infty} f(x) \{x \cos sx\} \, dx \\ &= \int_0^{\infty} \{x f(x)\} \cos sx \, dx \\ &= F_C\{x f(x)\} \end{aligned}$$

5.1 Finite Fourier Transforms of Derivatives

- (i) $F_S\{f'(x)\} = -\frac{n\pi}{l} \bar{f}_C(n)$
- (ii) $F_C\{f'(x)\} = (-1)^n f(l) - f(0) + \frac{n\pi}{l} \bar{f}_S(n)$
- (iii) $F_S\{f''(x)\} = -\frac{n^2\pi^2}{l^2} \bar{f}_S(n) + \frac{n\pi}{l} \{f(0) - (-1)^n f(l)\}$
- (iv) $F_C\{f''(x)\} = -\frac{n^2\pi^2}{l^2} \bar{f}_C(n) + (-1)^n f'(l) - f'(0)$

Proof:

$$\begin{aligned}
F_S\{f'(x)\} &= \int_0^l f'(x) \sin \frac{n\pi x}{l} dx \\
&= \int_0^l \sin \frac{n\pi x}{l} d\{f(x)\} \\
&= \left\{ f(x) \sin \frac{n\pi x}{l} \right\}_0^l - \frac{n\pi}{l} \int_0^l f(x) \cos \frac{n\pi x}{l} dx \\
&= -\frac{n\pi}{l} \bar{f}_C(n) \\
F_C\{f'(x)\} &= \int_0^{\frac{\pi}{2}} 1 \cdot \cos nx dx - \int_{\frac{\pi}{2}}^{\pi} 1 \cdot \cos nx dx \\
&= \left[\frac{\sin nx}{n} \right]_0^{\frac{\pi}{2}} - \left[\frac{\sin nx}{n} \right]_{\frac{\pi}{2}}^{\pi} \\
&= \frac{1}{n} \left\{ \sin \frac{n\pi}{2} - 0 - \sin n\pi + \sin \frac{n\pi}{2} \right\} \\
&= \frac{2}{n} \sin \frac{n\pi}{2}, n \neq 0 \\
F_S\{f''(x)\} &= \int_0^{\frac{\pi}{2}} \sin nx dx - \int_{\frac{\pi}{2}}^{\pi} \sin nx dx \\
&= -\frac{1}{n} \left(\cos nx \right)_0^{\frac{\pi}{2}} + \frac{1}{n} \left(\cos nx \right)_{\frac{\pi}{2}}^{\pi} \\
&= \frac{1}{n} \left\{ 1 - 2 \cos \frac{n\pi}{2} + (-1)^n \right\}
\end{aligned}$$

5.2 Finite Fourier Sine Transform

Finite Fourier Sine Transform of $f(x)$ in $(0, l)$ is denoted by $\bar{f}_S(n)$ or $F_S\{f(x)\}$. If the function $f(x)$ is piecewise continuous in the interval $(0, l)$, then the Fourier sine transform of $f(x)$ in $(0, l)$

$$\bar{f}_S(n) = F_S\{f(x)\} = \int_0^l f(x) \sin \frac{n\pi x}{l} dx$$

5.3 Finite Fourier Cosine Transform

Finite Fourier Cosine Transform of $f(x)$ in $(0, l)$ is denoted by $\bar{f}_C(n)$ or $F_C\{f(x)\}$. If the function $f(x)$ is piecewise continuous in the interval $(0, l)$, then the Fourier cosine transform of $f(x)$ in $(0, l)$

$$\bar{f}_C(n) = F_C\{f(x)\} = \int_0^l f(x) \cos \frac{n\pi x}{l} dx$$

Example on how to calculate Finite Fourier Sine and Cosine Transforms of a given function

Example 10

Find the Finite Fourier Sine and Cosine Transforms of $\left[1 - \frac{x}{\pi}\right]^2 \sin(0, \pi)$.

Solution:

$$\begin{aligned} F_s\left\{\left[1 - \frac{x}{\pi}\right]^2\right\} &= \int_0^\pi \left[1 - \frac{x}{\pi}\right]^2 \sin nx dx \\ &= \left\{\left[1 - \frac{x}{\pi}\right]^2 \left[\frac{-\cos nx}{n}\right] \left[\frac{-2}{\pi}\right] \left[\frac{1-x}{\pi}\right] \left[\frac{-\sin nx}{n^2}\right] + \frac{2}{\pi^2} \frac{\cos nx}{n^3}\right\}_0^\pi \\ &= \frac{1}{n} + \frac{2}{\pi^2 n^3} \{(-1)^n - 1\} \\ F_C\left\{\left[1 - \frac{x}{\pi}\right]^2\right\} &= \int_0^\pi \left[1 - \frac{x}{\pi}\right]^2 \cos nx dx \\ &= \left\{\left[1 - \frac{x}{\pi}\right]^2 \frac{\sin nx}{n} - \left[\frac{-2}{\pi}\right] \left[1 - \frac{x}{\pi}\right] \left[\frac{-\cos nx}{n^2}\right] + \frac{2}{\pi^2} \left[\frac{-\sin nx}{n^3}\right]\right\}_0^\pi \\ &= \frac{2}{\pi n^2}, n \neq 0 \end{aligned}$$

5.4 Inverse Finite Fourier Sine Transform

Inverse finite sine fourier transform of $\bar{f}_S(n)$ is denoted by $F_S^{-1}\{\bar{f}_S(n)\}$. If $\bar{f}_S(n)$ is the finite Fourier cosine Transform of $f(x)$ in $(0, l)$, then the Inverse Finite Fourier Sine Transform of $\bar{f}_S(n)$

$$f(x) = \frac{2}{l} \sum_{n=1}^{\infty} \bar{f}_S(n) \sin \frac{n\pi x}{l}$$

Example on how to calculate Inverse Finite Fourier Sine Transform of a given function

Example 11

Find $f(x)$, if its finite sine transform is given by $\bar{f}_S(n) = \frac{1 - \cos n\pi}{n^2\pi^2}$ in $0 < x < \pi$.

Solution:

The inverse finite Fourier sine transform is given by

$$\begin{aligned} f(x) &= \frac{2}{\pi} \sum_{n=1}^{\infty} \bar{f}_S(n) \sin nx \\ &= \frac{2}{\pi} \sum_{n=1}^{\infty} \left[\frac{1 - \cos n\pi}{n^2\pi^2} \right] \sin nx \\ &= \frac{2}{\pi^3} \sum_{n=1}^{\infty} \left\{ \frac{1 - (-1)^n}{n^2} \right\} \sin nx \\ &= \frac{4}{\pi^3} \sum_{n=1,3,5,\dots}^{\infty} \frac{1}{n^2} \sin nx \\ &= \frac{4}{\pi^3} \sum_{n=1}^{\infty} \frac{1}{(2n-1)^2} \sin (2n-1)x \end{aligned}$$

5.5 Inverse Finite Fourier Cosine Transform

Inverse finite cosine fourier transform of $\bar{f}_C(n)$ is denoted by $F_C^{-1}\{\bar{f}_C(n)\}$
If $\bar{f}_C(n)$ is the finite Fourier sine Transform of $f(x)$ in $(0, l)$, then the Inverse Finite Fourier Cosine Transform of $\bar{f}_C(n)$

$$f(x) = \frac{1}{l} \bar{f}_C(0) + \frac{2}{l} \sum_{n=1}^{\infty} \bar{f}_C(n) \cos \frac{n\pi x}{l}$$

Example on how to calculate Inverse Finite Fourier Cosine Transform of a given function

Example 12

Find $f(x)$ if its finite cosine transform is given by

$$\bar{f}_c(n) = \frac{1}{(2n+1)^2} \cos \frac{2n\pi}{3} \text{ in } 0 < x < 1$$

Solution:

The inverse finite Fourier cosine transform in $(0, l)$ is given by

$$f(x) = \frac{l}{l} \bar{f}_C(0) + \frac{2}{l} \sum_{n=1}^{\infty} \bar{f}_C(n) \cos \frac{n\pi x}{l}$$

Here $l = 1$ and $\bar{f}_C(0) = 1$

$$\therefore f(x) = 1 + 2 \sum_{n=1}^{\infty} \frac{1}{(2n-1)^2} \cos \frac{2n\pi}{3} \cos n\pi x$$

CONCLUSION

Fourier Transform has multitude of applications in almost all areas of life. Right from the cell phones that we use in our day to day life to Radio Astronomy, fourier transform plays a vital role in developing today's modern world of technology.

Fourier transform clearly shows us that any waveform can be re-written as the sum of sinusoidal functions. In other words, wherever there is a waveform which is absolutely integrable with finite number of maxima, minima and discontinuities, fourier transform can be definitely applied.

Among all it's tremendous applications, the most important contribution of fourier transform is the rapid diagnosis of COVID 19 using FT-IR ATR spectroscopy combined with machine learning. It offered a simple, free-label and cost-effective solution for high-throughput screening of suspect patients for COVID-19 in health care centres and emergency departments. So, by making effective researches on fourier transform, mankind will be much benefited.

REFERENCES

References

- [1] Integral Transforms and Their Applications by Lokenath Debnath and Dambaru Bhatta, Chapman and Hall/CRC, 3rd Edition, November 7, 2014, ISBN-13 : 978-1482223576.
- [2] Fourier Transform and Its Applications by Ronald Newbold Bracewell, McGraw-Hill, 2nd Edition, January 1, 1978, ISBN-13 : 978-0070070134.
- [3] Fundamentals of Fourier Transform Infrared Spectroscopy by Brian C. Smith, CRC Press, 2nd Edition, March 9, 2011, ISBN-13 : 978-1420069297.
- [4] The Fourier Transform: A Tutorial Introduction by James V Stone, Sebtel Press, Annotated Edition, April 11, 2021, ISBN-13 : 978-1916279148.
- [5] Fourier Series, Fourier Transform and Their Applications to Mathematical Physics by Valery Serov, Springer, 1st 2017 Edition, December 18, 2017, ISBN-13 : 978-3319652610.

FRACTAL THEORY

Project Report submitted to

ST.MARY'S COLLEGE (AUTONOMOUS), THOOTHUKUDI

Affiliated to

MANONMANIAM SUNDARANAR UNIVERSITY, TIRUNELVELI

In partial fulfilment of the requirement for the award of degree of

Bachelor of Science in Mathematics

Submitted by

NAME

ANTO SNOW SHERINE. I

JEBENA. J

JEFFRINA. J

JINCY JOSE. P

PARIPOORANA JEFFRINA. J

REG.NO.

19AUMT02

19AUMT14

19AUMT15

19AUMT16

19AUMT32

Under the Guidance of

Dr. A. PUNITHA THARANI M.Sc., M.Phil., Ph.D.

Associate Professor of Mathematics and COE

St. Mary's College (Autonomous), Thoothukudi.



Department of Mathematics

St. Mary's College (Autonomous), Thoothukudi

(2021 - 2022)

CERTIFICATE

We hereby declare that the project report entitled "**FRACTAL THEORY**" being submitted to **St. Mary's College (Autonomous), Thoothukudi** affiliated to **Manonmaniam Sundaranar University, Tirunelveli** in partial fulfilment for the award of degree of **Bachelor of Science in Mathematics** and it is a record of work done during the year 2021 - 2022 by the following students:

NAME	REG.NO.
ANTO SNOW SHERINE. I	19AUMT02
JEBENA. J	19AUMT14
JEFFRINA. J	19AUMT15
JINCY JOSE. P	19AUMT16
PARIPOORANA JEFFRINA. J	19AUMT32



Signature of the Guide

Dr. A. Punitha Tharanj

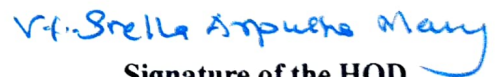
M.Sc., M.Phil., Ph.D.

Associate Professor,

Dept. of Mathematics,

St. Mary's College (Autonomous),

Thoothukudi - 628 001.



Signature of the HOD

Dr. V.L. Stella Arputha Mary

M.Sc., M.Phil., B.Ed., Ph.D.,

Head & Asst. Professor of Mathematics

St. Mary's College (Autonomous)

Thoothukudi-628 001.



Signature of the Examiner



Signature of the Principal

Principal

St. Mary's College (Autonomous)

Thoothukudi - 628 001.

DECLARATION

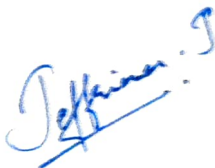
We hereby declare that the project reported entitled "**FRACTAL THEORY**", is our original work. It has not been submitted to any university for any degree or diploma.



(ANTO SNOW SHERINE. D)



(JEBENA. J)



(JEFFRINA. J)



(JINCY JOSE. P)



(PARIPOORANA JEFFRINA. J)

ACKNOWLEDGEMENT

First of all, we thank Lord Almighty for showering his blessings to undergo this project.

With immense pleasure, we register our deep sense of gratitude to our guide **Dr. A. Punitha Tharani M.Sc., M.Phil., Ph.D.** for having imparted necessary guidelines throughout the period of our studies.

We thank our beloved Principal, **Rev. Dr. Sr. A.S.J. Lucia Rose M.Sc., M.Phil., Ph.D., PGDCA** and the Head of the Department, **Dr. V. L. Stella Arputha Mary M.Sc., M.Phil., B.Ed., Ph.D.** for providing us the help to carry out our project work successfully.

Finally, we thank all those who extended their helping hands regarding this project.

FRACTAL THEORY

CONTENT

1.	Introduction	03
2.	What is meant by fractals?	03
3.	History of fractals.....	05
4.	Definitions and classifications	08
	4.1. Some definitions of fractals	
	4.2. Exact self-similarity	
	4.3. Quasi self-similarity	
	4.4. Statistical self-similarity	
	4.5. Iterative fractals	
	4.6. Recursive fractals	
	4.7. Random fractals	
5.	Fractal dimensions.....	11
	5.1. Scaling factor	
	5.2. Topological dimensions	
	5.3. The Hausdorff dimension	
	5.4. Box counting dimension	
	5.5. Iterated Functions Systems (IFS)	
6.	Examples of Exactly Self-Similarity	
	Fractals.....	14
	6.1. Cantor set	
	6.2. Koch curve	
	6.3. Minkowski curve	
	6.4. Koch snowflake	
	6.5. Sierpiński sieve (triangle)	
	6.6. Sierpiński carpet and tetrahedron	
	6.7. Menger sponge	
7.	Examples of Quasi Self-Similarity	
	Fractals.....	21
	7.1. Julia set	

7.2.	Mandelbrot set	
8.	Applications of Fractal Geometry.....	22
8.1.	Fractal geometry in architecture and civil engineering	
8.2.	Fractals in Computer Graphics	
8.3.	Fractals in Biological Science	
8.4.	Fractals in Film Industry	
8.5.	Fractals in Astrophysics	
8.6.	Fractals in Image Compression	
8.7.	Fractals in Fluid Mechanics	
8.8.	Fractals in Astronomy	
8.9.	Fractals in Telecommunications	
8.10.	Fractals in Antenna	
9.	Conclusion.....	33
	References.....	34

1.INTRODUCTION

- In the past, mathematics has been concerned largely with sets and functions to which the methods of classical calculus can be applied.
- Sets or functions that are not sufficiently smooth or regular have tended to be ignored as 'Pathological' and not worthy of study.
- Certainly, they were regarded as individual curiosities and only rarely were thought of as a class to which a general theory might be applicable.



- In recent years this attitude has changed. It has been realized that a great deal can be said, and is worth saying, about the mathematics of non-smooth sets.
- Irregular sets provide a much better representation of many natural phenomena than do the figures of classical geometry.
- Fractal Geometry provides a general framework for the study of such irregular sets.
- Fractal Geometry is concerned with the properties of fractal objects usually simply known as Fractals.

2.WHAT IS MEANT BY FRACTAL?

- A fractal is generally “a rough or fragmented geometric shape that can be split into parts, each of which is (at least approximately) a reduced- size copy of the whole,” a property called self-similarity.



shutterstock.com - 2123145290

- The term was coined by Benoît Mandelbrot in 1975 and was derived From the Latin fractus meaning “broken” or “fractured.” Mathematical fractals are based on a set of equations that are all. Characterized by iteration, a form of feedback based on recursion The fractal dimension would be a value that gives us an insight into the extent to which the fractal fills the space in which it is located. In other words, the fractal dimension is used to express the density at which the object fills the space, or to express how many new parts appear when increasing the resolution. The fractal dimension is not an integer and it is normally greater than the Euclidean dimension . For self-similar objects F it is natural define the self-similarity dimension or scaling dimension $d(F)$ by the expression:

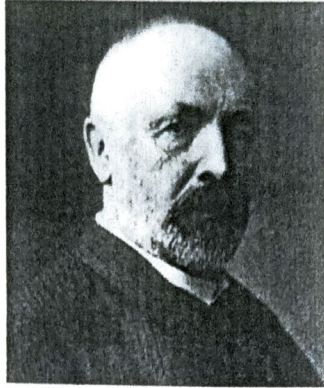
$$d(F) = \frac{\log N}{\log r}$$

where N is the number of new copies of the object observed after magnification, and r is the magnification factor.

3.HISTORY OF FRACTALS

- The mathematics behind fractals began to take shape in the 17th century when mathematician and philosopher Leibniz considered recursive self-similarity (although he made the mistake of thinking that only the straight line was self-similar in this sense).
- It took until 1872 before a function appeared whose graph would today be considered fractal, when Karl Weierstrass gave an example of a function with the non-intuitive property of being everywhere continuous but nowhere differentiable.
- In 1904, Helge Von Koch, dissatisfied with Weierstrass's very abstract and analytic definition, gave more geometric definition of a similar function, which is now called the Koch Snowflake.
- In 1915, Wacław Sierpinski constructed his triangle and, one year later, his carpet. Originally these geometric fractals were described as curves rather than the 2D shapes that they are known as in their modern constructions.
- In 1918, Bertrand Russell had recognized a "supreme beauty" within the mathematics of fractals that was then emerging.
- The idea of self-similar curves was taken further by Paul Pierre Levy, who, in his 1938 paper *Plane or Space Curves and Surfaces Consisting of Parts Similar to the Whole* described a new fractal curve, the Levy C Curve.

- George Cantor also gave examples of subsets of the real line with unusual properties – these Cantor sets are also now recognized as fractals.



George Cantor

- Iterated functions in the complex plane were investigated in the late 19th and early 20th centuries by Henri Poincare, Felix Klein, Pierre Fatou and Gaston Julia.
- However, without the aid of modern computer graphics, they lacked the means to visualize the beauty of many of the objects that they had discovered.

GASTON JULIA

Gaston Julia (1893-1978) was a French mathematician who published a book on the iteration of rational functions in 1918.



Before computers, he had to draw the sets of functions by hand. These types of fractals are now called Julia sets.

- In the 1960s, Benoit Mandelbrot started investigating self-similarity in papers such as *How Long Is the Coast of Britain? Statistical self-similarity and fractional dimension*, which is built on earlier work by Lewis Fry Richardson.
- Finally, in 1975 Mandelbrot coined the word “fractal” to denote an object whose Hausdorff-Besicovitch dimension is greater than its topological dimension.

BENOIT MANDELBROT

Benoit Mandelbrot is a mathematics professor at Yale University. He used a computer to explore Julia's iterated functions, and found a simpler equation that included all the Julia sets. This Mandelbrot set is named after him. He is the “Father of Fractals”.

He illustrated this mathematical definition with striking computer-constructed visualizations. These images captured the popular imagination; many of them were based on recursion, leading to the popular meaning of the term “Fractal”.



Mandelbrot realized that it is very often impossible to describe nature using only Euclidean Geometry, that is in terms of straight lines, circles, cubes and such like.

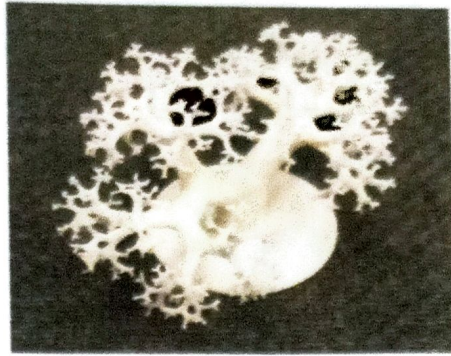
He proposed that fractals and fractal geometry could be used to describe real objects, such as trees, lightning, river meanders and coast lines, to name but a few.

4.DEFINITIONS AND CLASIFICATIONS

4.1.Some Definitions of Fractals:

- A fractal is an object which appears self-similar under varying degrees of magnification. In effect, possessing symmetry across scale. With each small part of the object replicating the structure of the whole.
- A fractal is defined to be a set with Hausdorff dimension strictly greater than its topological dimension.
- According to Mandelbrot, a fractal is a set for which the Hausdorff dimension strictly exceeds the topological dimension

and a fractal is a shapes made up of parts similar to the whole in some way



CLASSIFICATION:

4.2.Exact self-similarity –

This is the strongest type of self-similarity; the fractal appears identical at different scales. Fractals defined by iterated function systems often display exact self-similarity.

4.3.Quasi-self-similarity –

This is a loose form of self-similarity; the fractal appears approximately (but not exactly) identical at different scales. Quasi-self-similar fractals contain small copies of the entire fractal in distorted and degenerate forms. Fractals defined by recurrence relations are usually quasi-self-similar but not exactly self-similar.

4.4.Statistical self-similarity –

This is the weakest type of self-similarity; the fractal has numerical or statistical measures which are preserved across scales.

Not reasonable definitions of “fractal” trivially imply some form of statistical self-similarity. (Fractal dimension itself is a numerical measure which is preserved across scales.) Random fractals are examples of fractals which are statistically self-similar, but neither exactly nor quasi-self-similar.



According to the type of formation, fractals can be divided into:

4.5.Iterative fractals - formed by copying and rotating and/or translating the copy, and

possibly replacing an element with a copy - these are self-similar fractals;

4.6.Recursive fractals - are defined by a recursive mathematical formula that determines

whether a given point of space (e.g. complex or Gauss plane) belongs to the set or not

- these are quasi-self-similar fractals;

4.7.Random fractals - have the lowest degree of self-similarity and are frequently found in

nature (coastlines, river branches and flows, mountain ranges, jungles, tree roots

and tops, leaves, flowers, clouds, lightnings, climate systems, snowflakes, bacteria,

lungs, blood vessel systems...) - these are statistically self-similar fractals

5.FRACTAL DIMENSION

DIMENSIONS

Dimension is a property of a mathematical object that refers to the extent it occupies the space in which it is embedded. There are many formal definitions of dimension, if the definition allows non-integer values (a fraction), it is a Fractal dimension. The box-counting dimension are defined over a vector space, but there is also the packing dimension, compass dimension etc

General rule for Dimensions

- A figure is in D dimensions
- If I magnify the length by R , then I would get R^D copies
- $N = R^D$

The dimension formula

Define R as the magnifying factor,

Define N as the number of identical("self-duplicating") copies.

Then the dimension of a figure is:

$$D = \frac{\log(N)}{\log(R)}$$

5.1. Scaling Factor

We can divide the object in N self-similar pieces then, how to get original object from size of these N pieces?

Scaling Factor: If we want to get original object from any part of self-similar then we have to scale the object using scaling factor.

For example:

If we divide the line in 2 equal pieces then SF is 4

If we divide the plane in 4 equal pieces then SF is 2

5.2. Topological Dimension:

The topological dimension of a set is always an integer and is zero if it is totally disconnected, one if each point has arbitrarily small neighborhoods with boundary dimension zero and so on

Fractal Dimension:

Let (X, d) be a metric space. Let $H(X)$ be a collection of all non empty compact subsets of X . Let $A \in H(X)$. For each $\varepsilon > 0$. Let $N(A, \varepsilon)$ denote the smallest number of closed balls of radius $\varepsilon > 0$ needed to cover A . If

$$D = \lim_{\varepsilon \rightarrow 0} \left\{ \frac{\ln(N(A, \varepsilon))}{\ln(\frac{1}{\varepsilon})} \right\}$$

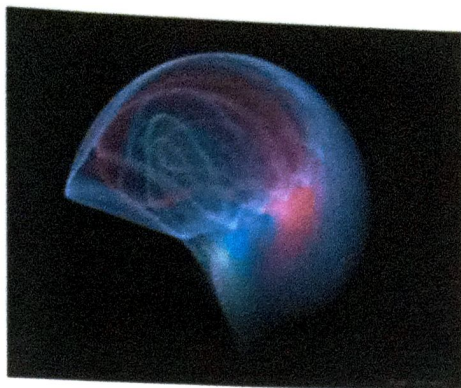
exists, then D is called the Fractal Dimension of A and we will say A has Fractal Dimension D .

The Hausdorff Distance

Definition: Let (X, d) be a complete metric space. Then the Hausdorff distance between two points $A, B \in H(X)$ is defined by

$$h(A, B) = \max\{d(A, B), d(B, A)\}$$

Theorem: The Hausdorff distance is metric on $H(X)$



5.3. Hausdorff Dimension

Let F be a subset of R^n and s is a non-negative integer. For any $\delta > 0$ we define

$$H^s(F) = \lim_{\delta \rightarrow 0} H^s_\delta(F)$$

We call $H^s(F)$, the s -dimensional Hausdorff measure.

There is a critical value of s at which $H_s(F)$ jumps from ∞ to 0. This critical value is called the **Hausdorff dimension**

5.4. Box Counting Dimension

Let F be any non-empty bounded subset of R^n and let $N_\delta(F)$ be the smallest number of sets of diameter at most δ which can cover F . If these are equal we refer to the common value as the box-counting dimension or box dimension of F

$$\dim_B F = \lim_{\delta \rightarrow 0} \log N_\delta(F) / -\log \delta$$

5.5. Iterated Function Systems (IFS)

- Let K be a closed subset of R^n . Let S_1, S_2, \dots, S_n be contractions on K . We call a subset K of D invariant (Attractor) for the transformations S_i if

$$K = \bigcup_{i=1}^n S_i[K]$$

As we shall see, such invariant set are often fractals.

- A finite families of contractions on X forms an iterated function systems.

6. EXAMPLES OF SELF SIMILAR FRACTALS

6.1. Cantor set

A subset of separated points remaining after dividing a line segment of length 1 into three equal subsegments, removing all the points from the middle subsegment, dividing each of the remaining subsegments into three, removing all the points from the middle one of these three subsegments and continuing so infinitely many times.

PROPERTIES OF CANTOR SETS

- Cantor set contain 2^k intervals of length 3^{-k} .
- It satisfies open set Condition.

$D \subset R^n$. a non-empty compact subset F of D an attractor (or invariant set) for the IFS if

$$F = \bigcup_{i=1}^m S_i(F)$$

S_i satisfy the open set condition if there exists a non-empty bounded open set V such that

$$V \supseteq \bigcup_{i=1}^m S_i(V)$$

with the union disjoint.

- It can be determined by the Iterated Function Systems (IFS) and it contains self-similarity.
- It has Hausdorff dimension.
- Cantor set is the perfect set. Because it is closed and dense in itself.
- Cantor set is a dense set.
- It contains countable set of points.
- Cantor set contains the set of numbers in $[0,1]$ whose base three expansions do not contain the digit 1.
- $H^s(E) = 1$ if E is a Cantor set $\dim(E) = \frac{\log 2}{\log 3} = 0.6309$ set contains no intervals.
- The Cantor set C has no isolated points, that is if $a \in C$, then for every $\varepsilon > 0$, the interval $(a - \varepsilon, a + \varepsilon)$ contains points of C , in addition to a .
- Cantor set is closed.

$$C \subset [0,1] \subset \mathbb{R}^n$$

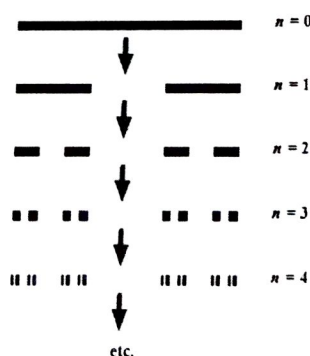
- Let $\{Q_1, Q_2, \dots\}$ be a countable collection of non empty sets in \mathbb{R}^n , such that

1. $Q_{k+1} \subseteq Q_k (k = 1, 2, \dots)$
2. Each set Q_k is closed and Q_1 is bounded.

Then the intersection $\bigcap_{k=1}^{\infty} Q_k$ is closed and non-empty.

- Cantor set is a compact set having measure zero.

Cantor set



6.2. Koch curve

It is constructed by dividing a line segment of length a into three equal segments, each of length $\frac{a}{3}$, and then adding to the middle segment two more line segments of equal lengths so that they form an equilateral triangle together with the middle segment; then the middle segment is removed and now we have four line segments of equal lengths (each of them being $\frac{a}{3}$); in the second step the procedure from the first step is repeated for each of these four segments, etc.

PROPERTIES OF KOCH CURVE

- The Koch curve has an infinite length because each time the steps above are performed on each line segment of the figure there are four times as many line segments in the previous stage.
- Hence the total length increases by one third and thus the length at step n will be $\left(\frac{4}{3}\right)^n$ of the original triangle perimeter: the fractal dimension is $\frac{\log 4}{\log 3} \cong 1.26$ greater than the dimension of a line but less than Peano's space-filling curve.
- The Koch curve is continuous everywhere but differentiable nowhere.

Koch Curve

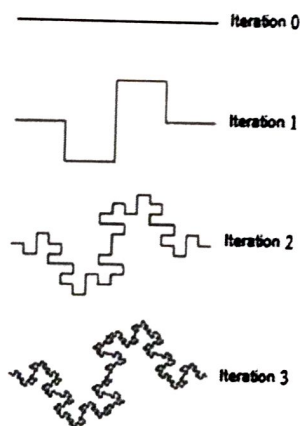


6.3. Minkowski curve

The **Minkowski sausage** or **Minkowski curve** is a fractal first proposed by and named for Hermann Minkowski as well as its casual resemblance to a sausage or sausage links. The initiator is a line segment and the generator is a broken line of eight parts one fourth the length.

The Sausage has a Hausdorff dimension of $(\frac{\ln 8}{\ln 4}) = 1.5 = 3/2$. It is therefore often chosen when studying the physical properties of non-integer fractal objects. It is strictly self-similar. It never intersects itself. It is continuous everywhere, but differentiable nowhere. It is not rectifiable. It has a Lebesgue measure of 0. The type 1 curve has a dimension of $\frac{\ln 5}{\ln 3} \approx 1.46$.

Multiple Minkowski Sausages may be arranged in a four sided polygon or square to create a quadratic Koch island or Minkowski island/[snow]flake:



Initial line segment and the first three iterations of Minkowski curve.

6.4. Koch snowflake

It is obtained when the iteration procedure described for a Koch curve, instead of a line segment of length a , starts with an equilateral triangle with sides of length a .

The Koch snowflake can be built up iteratively, in a sequence of stages. The first stage is an equilateral triangle, and each successive stage is formed by adding outward bends to each side of the previous stage, making smaller equilateral triangles. The areas enclosed by the successive stages in the construction of the snowflake converge to $8/5$ times the area of the original triangle, while the perimeters of the successive stages increase without bound. Consequently, the snowflake encloses a finite area, but has an infinite perimeter.



Initial equilateral triangle and the first six iterations of Koch snowflake

6.5.Sierpiński sieve (triangle)

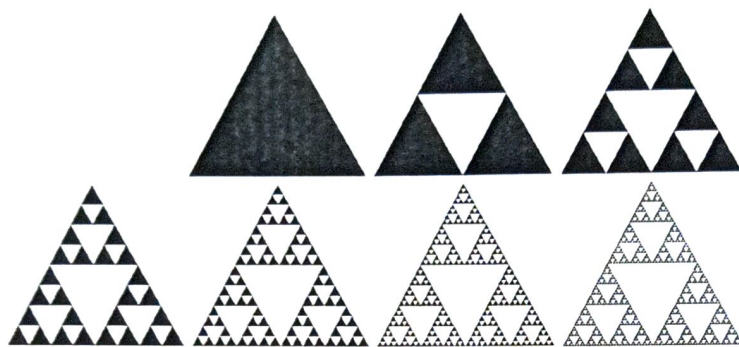
The construction starts from the initial equilateral triangle, by removing a triangle (also equilateral) obtained by connecting midpoints of the sides of the initial triangle; the remaining three equilateral triangles represent the starting point for the next step, etc.

PROPERTIES OF SIERPINSKI SIEVE

- The sierpinski triangle has Hausdorff dimension $\frac{\log(3)}{\log(2)} \cong 1.585$, which follows from the fact that it is a union of three copies of itself, each scaled by a factor of $1/2$.
- If one takes Pascal's triangle with $2n$ rows and colors the even numbers white, and the odd numbers black, the result is

an approximation to the sierpinski triangle. More precisely, the limit as n approaches infinity of this parity - colored $2n$ -row Pascal triangle is the sierpinski triangle.

- The area of a sierpinski triangle is zero (in Lebesgue measure). The area remaining after each iteration is clearly $\frac{3}{4}$ of the area from the previous iteration, and an infinite number of iteration results in zero.



Initial equilateral triangle and the first six iterations of the Sierpiński sieve (triangle)

Non – Integer Dimension

Using this fractal as an example, we can prove that the fractal dimensions is not an integer. Looking at the picture of the first step in building the Sierpiński Triangle, we can notice that if the linear dimension of the basis triangle is doubled, then the area of the whole fractal (black triangles) increases by a factor of three.

Using the pattern given above, we can calculate a dimension for the Sierpiński Triangle.

$$D = \frac{\log 3}{\log 2} = 1.585$$

The result of this calculation proves the non-integer fractal dimension.

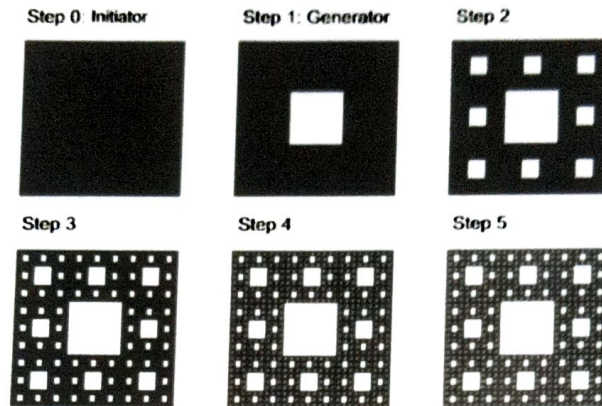
The number of triangles in the Sierpiński Triangle can be calculated with the formula:

$$N = 3^k$$

Where N is the number of triangles and k is the number of iterations

6.6. Sierpiński carpet and tetrahedron

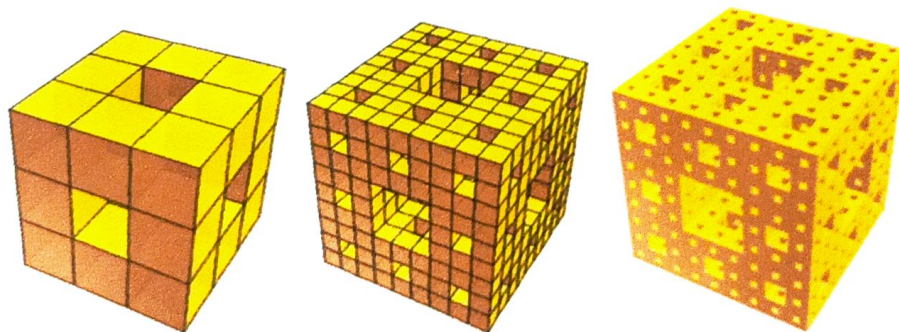
They are obtained when a similar procedure of iteration described for Sierpiński sieve is applied to square or tetrahedron



Initial square and the first five iterations of Sierpiński carpet.

6.7. Menger sponge

It is a three-dimensional analogue of the Sierpiński carpet; each side of the Menger sponge is a Sierpiński carpet, and each diagonal is a Cantor set.



The first three iterations of the Menger sponge – cube with volume 0 and surface area ∞

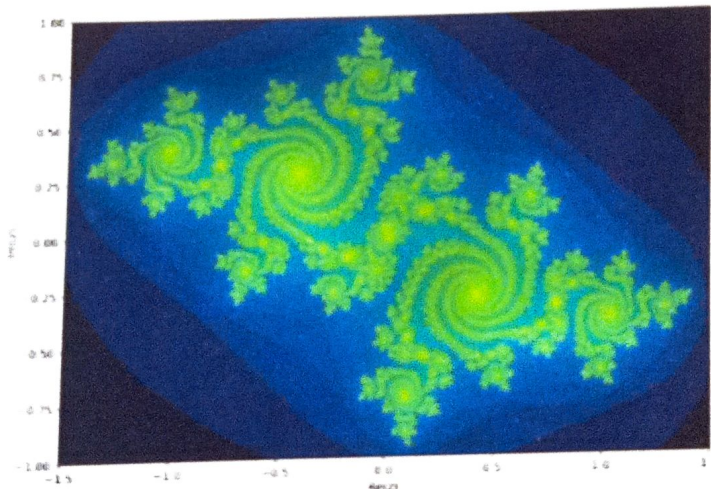
Self-Similar Objects and Fractal Dimensions

- Fractal dimension is a measure of how “complicated” a self-similar figure is.
- i.e., to measure the fractal dimension, the picture must be self-similar.
- Self-similar regular shapes: Line, Plane, Cube
- Self-similar irregular shapes: Cauliflower, Galaxy, Coast Line.

7.EXAMPLES OF QUASI SELF SIMILAR FRACTALS

7.1.Julia Set

- The boundary of the basin of infinity is non empty. This boundary is called Julia Set.
- Julia sets for the map $f(x) = z^2 + c$

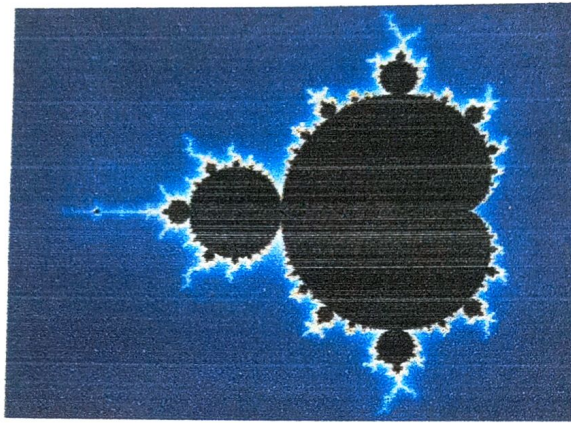


Julia set

7.2.Mandelbrot Set

It's concept is closely linked to the concept of a Julia set. A Mandelbrot Set is formed using very simple algorithms as the Julia set. To draw a Mandelbrot Set do the following:

- Make a complex number z equal to $0 + 0i$.
- Choose some point on the complex plane, and make a complex number c equal to its coordinates.
- Make $z = z^2 + c$ and make this change a lot of time.
- If the number did not go to infinity, it belongs to the set and you can mark it. Otherwise you can color the point depending on how fast it escaped to infinity.
- Repeat steps 1-5 for all the points on the plane.



Mandelbrot set

8.APPLICATIONS OF FRACTAL GEOMETRY

The facts that fractals are abundant in nature and natural phenomena, is itself a testimony to the potential applicability and design efficiency of these shapes. Fractal shapes capture the fine details and organic irregularity of natural forms like clouds, coast lines and land shapes.

Fractals have variety of applications in science. Because it's property of self-similarity exists everywhere. They can be used to model plants, blood vessels, nerves, explosions, clouds, mountains, turbulence, etc. Fractal geometry models natural objects more closely than does other geometries.

Engineers have begun designing and constructing fractals in order to solve partial engineering problems. Fractals are also used in computer graphics are even in composing music.

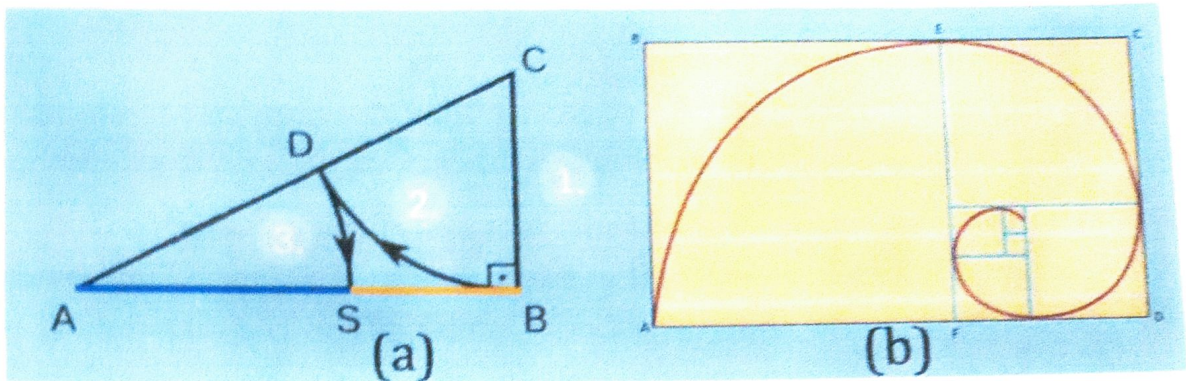
Fractal geometry has permeated many areas of science. Such as astrophysics, biological science, and has become one of the most important techniques in computer graphics. Architects are using fractal geometries to create more impressive buildings. Digital artists use fractal geometries to create interesting art work which engages views at variable scales Game designers are always seeking to create natural organic environments. Which do not seem to be constructed and synthetic. Fractal geometry can be applied in such environments to include random elements which can enrich user experience.

Fractals are also used to generate natural patterns which can create effective camouflage and preclude artificial repetitive motifs. Fractals have been used by seismologists to understand earthquake phenomena and gain deeper understanding of the earth physical constitution. As well as the distribution pattern of earthquakes. Financial theorists have even applied fractals to understand and forecast stock market patterns.

8.1. Fractal geometry in architecture and civil engineering

Intentionally or unintentionally, architects, builders and other construction experts have been using mathematics and geometry since ancient times as the most basic but nevertheless rather valuable tool in almost all stages of architectural and construction projects. History remembers great builders and architects who were also great mathematicians, and vice versa (Vitruvius, Leonardo Da Vinci,...). Mathematics and architecture have always been close, most of all because of their common aspiration for order and beauty . The discovery of fractal geometry (or the geometry of nature), attributed to the French-American mathematician Benoit Mandelbrot, has inevitably led to a great revolution in natural and technical sciences, and consequently in architecture and civil engineering as fields of

technical sciences. The reason why fractals are used so much lies in that many shapes in nature (coastlines, river branches and flows, mountain ranges, jungles, tree roots and tops, leaves, flowers, clouds, lightnings, climate systems, snowflakes, bacteria, lungs, blood vessel systems...) are irregular and rugged, and offer these irregularities on different scales. The characteristics of all these shapes are essentially contrary to the characteristics of regular geometric shapes and bodies of Euclidean geometry (ball, cube, pyramid, cone), but can be considerably better represented using fractals (nature is fractal). In other words, fractal geometry, in contrast to the Euclidean one, offers considerably better methods for describing natural objects. The rugged characteristics of nature are not modeled by smooth shapes of Euclidean geometry, but it is the new approach of fractal complexity that copes with the irregularities of the structure itself. Although complex, a fractal is usually described by a simple algorithm, indicating that there is a law behind the greatest ruggedness and irregularities. Despite the fact that fractal geometry developed only in the late 20th century, fractals have been known to people from times immemorial, they were just not recognized as such (for example, golden ratio was studied by Pythagoras and Euclid in connection with the construction of dodecahedron and icosahedron, i.e. polyhedra bounded by twelve and twenty planes, respectively). Architects and builders have used fractals as decorative elements from ancient times. As examples of ancient architecture in which fractal components are present or dominant, the most remarkable are Egyptian pyramids, Buddhist and Hindu temples, Gothic cathedrals, but also some cathedrals from the earlier period. Historians of mathematics and art, even today, have been unable to determine with certainty when golden ratio appeared for the first time in some of the old civilizations, but they all agree that golden ratio, purposely or not, was applied in ancient Egypt in the construction of the Cheops Pyramid in Giza (the ratio of the height of the facet to half the length of the base edge), one of the seven wonders of the old world that still exists today.



(a) Construction of golden ratio, $AB : BC = 2 : 1$; (b) Golden rectangle a golden spiral

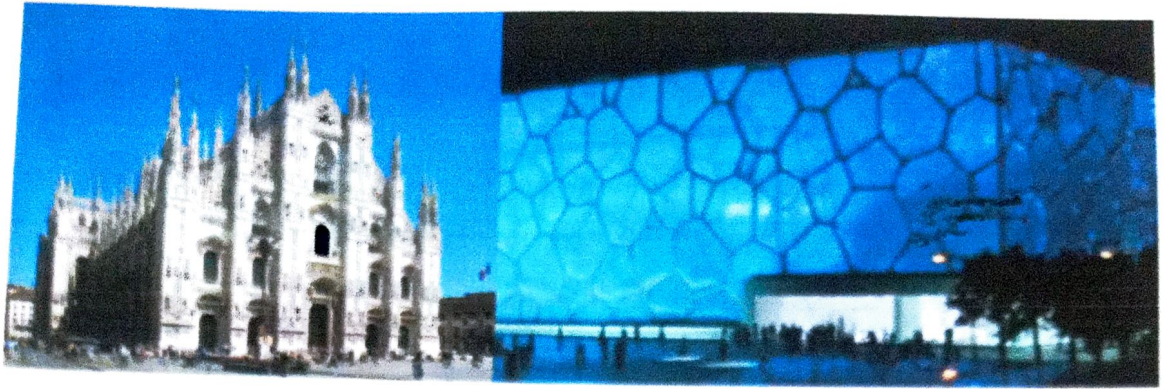
It was precisely the golden ratio that was notably used in ancient and Hindu architecture, Gothic architecture, Renaissance, and later in classicism, usually for the design of faces or ground plans of temples, mausoleums, churches and cathedrals (Parthenon at Athenian Acropolis (432 BC), Taj Mahal in Agra (1653, Figure 6), cathedrals in Anagni (1104), Florence (1436), Milan (Figure 8), Paris (1345), Reims (1275),...). The golden ratio (Latsectio aurea, or divine ratio, is the ratio of parts of a line segment in which the whole segment $a + b = AB$ is related to the larger part $b = AS$ as the larger part b is related to the smaller part

$$a = SB, \text{ or } (a + b) : b = b : a = \phi, \phi = (1 + \sqrt{5}) / 2 \approx 1.618$$

The golden rectangle or a rectangle whose sides are in the golden ratio, is considered to be a fractal because it has the following property of self-similarity: when a square is removed from (or added to) a golden rectangle, a smaller (or larger) rectangle with entirely the same golden ratio always remains, and so indefinitely. The golden spiral or the spiral formed by the arcs connecting opposite corners of inserted squares, is actually a logarithmic spiral, i.e. a spiral for which it is true that any tangent makes the same angle with the spiral radius. The center of the golden spiral is at the intersection of diagonals of golden rectangles, which intersect each other at right angle, at a ratio that involves the golden section again. It is worth mentioning that, as it expands, it changes in size but it never changes in shape, i.e., it carries in itself a growth matrix that is present in nature in many

examples. More precisely said, wherever nature needs economical and regular filling of space, there is a golden spiral.

Essentially, it is true that whenever we notice an exceptional beauty and harmony, we will usually reveal the presence of golden ratio so one should not wonder why this concept, which connects mathematics, nature, science, engineering and art in a very unusual and interesting way, is present in all aspects of human life. Human aspiration is to be surrounded by structures and works pleasant to the eye, so it is logical to expect the magic of golden ratio to be found in the pores of mathematics, architecture, painting, sculpture, music and many other scientific disciplines . Numerous experiments (although not all) in which respondents were asked to choose among a certain number of rectangles one that they liked most, show that people prefer a rectangle whose sides are in the ratio ϕ . The first such experiment was performed by one of the founders of modern psychology, Gustav T. Fechner. A team of colleagues from the Department of Civil Engineering at University North performed similar experiments during classes of history of architecture, and the results of these experiments show that their students choose the golden rectangle as the most pleasing to their eye .The concept shared by the Cheops Pyramid in Giza, the temple of Parthenon at Athenian Acropolis, the Taj Mahal mausoleum in Agra, the Constantine's triumphal arch in Rome, the Cathedral of Notre Dame in Paris, the United Nations headquarters building in New York, a human body, a regular pentagon and a regular pentagram, design of credit cards, a common snail, sunflower ... makes mathematics a universal science. Like no other concept that appeared in the history of mathematics, it inspired many thinkers of various disciplines to discover its presence in various fields of life and periods of human existence .

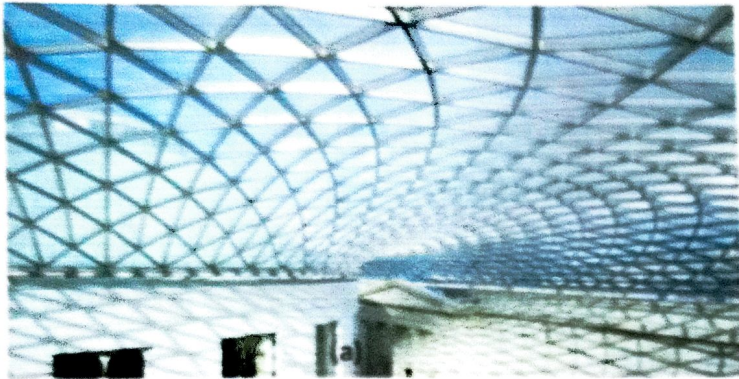


The Milan Cathedral (built from the late 14th to the middle of 19th century) and Beijing National Aquatic Center (2008)

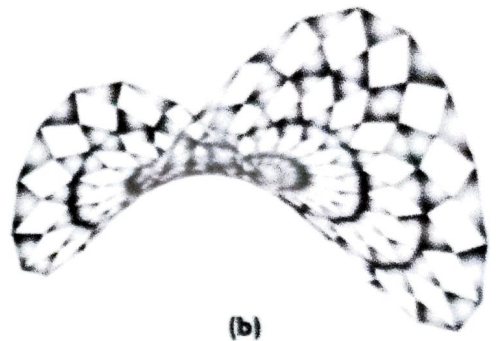
A considerable number of examples of fractal geometry principles being used in construction and design can be found in modern architecture. Two viewpoints or two approaches to the fractal concept typically stand out: one approach attempts to simulate natural forms or apply similar forms of different sizes at different scales in project design, while the other approach attempts to measure (the measurement is based on calculation of fractal dimension) the complexity of forms. In both approaches, fractal geometry provides a quantitative means to describe the complexity of the physical appearance of an architectural product and the degree to which it qualifies as a "fractal". The National Aquatic Center in Beijing or the British Museum in London, a typical example of the classic method of opposing the old and the new, and the spiral staircase in the Vatican Museums are just a few modern and magnificent examples of architecture with fractal components.

In the last two decades, primarily thanks to the rapid development of computer technologies and computer graphics, civil engineers have been provided with powerful tools for modeling and analysis of structures with highly nonlinear structure (e.g. hyperbolic paraboloid), modeling and regulation of rivers and sea shores, techniques they knew from before, but also for creating something completely new, which normally relies on disruption of the established and usual order of things. Even Mandelbrot used the example of coastline as a fractal – inlets look like bays, capes look like peninsulas; if we came closer,

every rock would look like a peninsula... Today, there are other specialized software solutions that enable engineers to apply fractal geometry in the design of grid or reticulated shell structures, but also in the regulation of rivers and sea shores, or to make models of simple repeating forms, that are close to the fractal concept, applying a set of rules of non-linear transformations in the process.

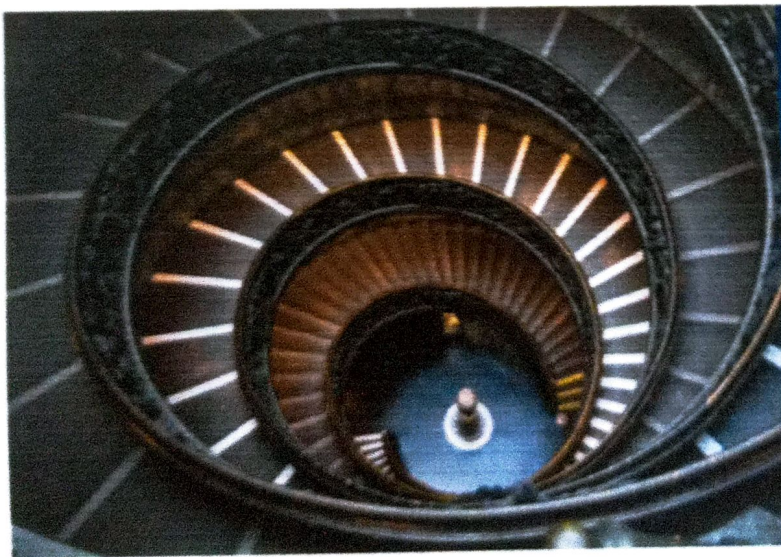


The British Museum in London

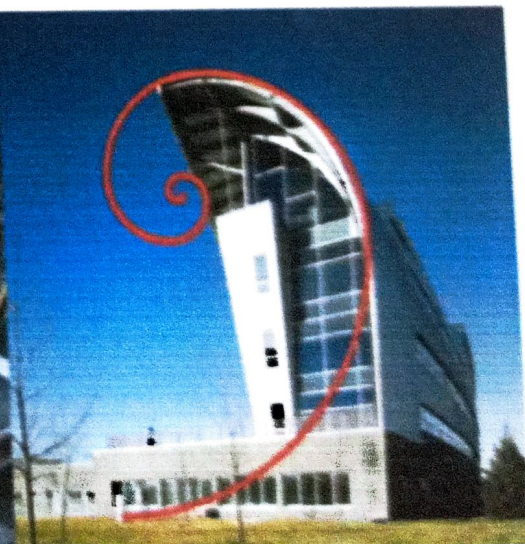


(b)

Hyperbolic Paraboloid



*Spiral staircase in the Vatican
Museums (1932)*



*City of Calgary Water
Centre (2008)*

8.2.Fractals in Computer Graphics

The biggest usage of fractals in everyday life is in computer science. Many images compression schemes use fractal algorithms to compress computer graphics files to less than a quarter of their original size.

Computer graphics artists uses many fractals forms to create text termed landscapes and other intricate models.

It's possible to create all sorts up realistic "Fractal forgeries" images of natural scene, such a lunar landscape, mountain ranges and coastlines. We can see them it may special effects in Hollywood movies and also in television ads. The "genesis effect" in the films "star trek II". "The worth of khan" was created using fractal used to create the geography of a moon. and to draw the outline of dreaded "death star". But fractal signals can be used to model natural objectives. Allowing up to define mathematically our environment with a higher accuracy than ever before.

8.3.Fractals in Biological Science

Biological scientists have traditionally model nature using Euclidean representations of natural object or series. They represented heartbeats as sine waves. Conifer trees as cones, animals habit a simple area, and cell membranes as curves or simple surfaces however scientists have come to recognize that many natural constructs are better characteristic using fractal geometry. Biological systems and processes are typically characterized by many levels of substructure with some general pattern repeated in an ever-decreasing cascade.

Scientists discovered that basic architecture of a chromosome in tree like: every chromosome consist of many "mini chromosomes" and therefore can be treated as fractal for a human chromosome, for in theory one can argue that everything existent on this world is fractal: -

- The branching of tracheal tubes
- The leaves in trees
- The veins in hand
- Water swirling and twisting out of a tap
- A puffy cumulus clouds
- Tiny oxygen molecules or the DNA molecules
- The stocks market

All of these are fractals from people ancient civilizations to the marker of star trek II: The worth of khan scientists. Mathematicians and artists alike have been captivated by fractal and have utilized them in their work.

8.4.Fractals in Film Industry

One of the more trivial applications of fractals is their visual effect. Not only do fractals have a stunning aesthetic value that is, they are remarkably pleasing to the eye, but they also have a way to trick the mind. Fractals have been used commercially in the film industry. Fractal images are used as an alternative to costly elaborate sets to produce fantasy land scape.

8.5.Fractals in Astrophysics

Nobody really how many stars actually glitter in our skies, but have you ever wondered how they were formed and ultimately found their home in the world? Astrophysicist believe that the key to this problem in the fractal nature of interstellar gas. Distributions are hierarchical, like smoke trails or billow cloud in the sky and the clouds in space. Giving them an irregular but repetitive pattern that would be impossible to describe without the help of fractal geometry

8.6.Fractals in Image Compression

Most use full application of fractals and Fractal geometry in image compression it is also one of the more controversial ideas. The basic concept behind of fractal image compression is to take an image and express it as an it rated system of functions the image can be quickly displayed, and at any magnification with infinite levels of fractal details. The largest problems behind its ideas is deriving the system of functions which describe an image.

8.7.Fractals in Fluid Mechanics

The study of turbulence in flows is very adapted to fractals. Turbulent flows are chaotic and very difficult to model correctly. A fractal representation of them helps engineers and physicists to better understand complex flows. Flames can also be simulated. Porous media have a very complex geometry and are well represented by fractal. This is actually used in petroleum science.

8.8.Fractals in Astronomy

Fractals will may be revolutionize the way that the universe is seen. Cosmologists usually assume that matter is spread uniformity across space. But observations show that is not true. Astronomers agree with that assumptions on “small” scales. But most of them think that the universe is smooth at very large scales. However, a dissident group of scientists claims that the structure of the universe is fractal at all scales. If this new theory is proved to be correct, even the big bung models should be adapted. Some years ago, we proposed a new approach for the analysis of galaxy and cluster correlations abused on the concepts and methods of modern statistical physics. This led to the surprising result that galaxy correlations are fractal and not homogeneous up to the limits of the available catalogues. In the

meantime, many more redshifts have been measured and we have extended our method also to the analysis of number counts and angular catalogues. The result is that galaxy structures are highly irregular and self-similar. The usual statistical method, based on the assumption of homogeneity, are therefore inconsistent for all the length scales probed until now. A new more general conceptual framework is necessary to identify the real physical properties of these structures. But present cosmologists need more data about the matter distribution in the universe to prove (or not) that we are living in a fractal universe.

8.9.Fractals in Telecommunications

A new application is fractal- shaped antennae that reduce greatly the size and the weight of the antennae. Fractenna is the company which sells these antennae. The benefits depend on the fractal applied, frequency of interest, and so on. In general, the fractal parts produce 'fractal loading' and makes the antenna smaller for given frequency of use. Practical shrinkage of 2-4 times is realizable for acceptable performance. Surprisingly high performance is attained.

8.10.Fractal Antenna

A fractal antenna is an antenna that uses a fractal, self-similar design to maximize the length, or increase the perimeter (on inside section or the outer structure), of material that can receive or transmit electromagnetic radiation within a given total surface area or volume. Cohen use this concept of fractal antenna. And it is theoretically it is proved that fractal design is the only design which receives multiple signals.

9.CONCLUSION

We presented a brief overview of one of the greatest secrets of nature's design: rugged, irregular, self-similar infinite objects developed through multiple repetition called fractals through this project. Fractal Theory is one of the most important concepts in Mathematics. Fractal shapes are used to capture the fine details and organic irregularity of natural forms like clouds, coast lines land shapes, etc. Fractal theory helps us to describe objects like trees, clouds, lightning, river meanders etc. The fact that irregular sets provide a much better representation of many natural phenomena than do the figures of classical geometry has led to the introduction of fractal Theory. The project displays the classifications of fractals based on their formation and appearance. It provides the history of fractals and the list of mathematicians who have contributed to it. It also provides brief examples for exactly Self similarity fractals such as cantor set, Koch curve, etc. and Quasi Self similarity fractals such as Julia set, Mandelbrot sets, etc. Fractals have variety of applications in science because it's property of self-similarity exists everywhere. It demonstrates the applications of fractals in various fields like architecture, computer graphics, biological sciences, film industry, astrophysics, Image Compression, Fluid Mechanics, astronomy, Telecommunication and Fractal Antenna. Game designers are always seeking to create natural organic environments. Fractal geometry can be applied in such environments to include random elements which can enrich user experience. Fractals are also used to generate natural patterns which can create effective camouflage and preclude artificial repetitive motifs. Hence making fractals an important concept. The facts that fractals are abundant in nature and natural phenomena, is itself a testimony to the potential applicability and design efficiency of these shapes.

References

- B B. Mandelbrot, 'The Fractal Geometry of Nature', San Francisco, W. H. Freeman and company, 1982.
- B B. Mandelbrot, 'The Fractal Geometry of Nature' 1981 yearbook of science and future, Encyclopedia Britannica, Chicago, 1980.
- Sala, N.: Fractal geometry and architecture: Some interesting connections, WIT Transactions on the Built Environment, 2006, Vol. 86, pp. 163–173.
- Vyzantiadou, M.A., Avdelas, A.V., Zafiropoulos, S.: The application of fractal geometry to the design of grid or reticulated shell structures, Computer–Aided Design, 2007, Vol. 39, No. 1, pp. 51–59.
- Zlatić, S.: Zlatni rez, Technical journal, 2013, Vol. 7, No. 1, pp. 84–90.
- Wolfram Alpha LLC, 2019. Wolfram|Alpha.

Websites

- ❑ Fractal- Wikipedia
<https://en.wikipedia.org/wiki/Fractal>
- ❑ Fractal antenna
https://slideplayer.com/slide/10852710/39/images/4/WHA+T+IS+A+FRACTAL+ANTENNA.jpg?hl=en_IN
https://en.wikipedia.org/wiki/Fractal_antenna
- ❑ <https://fractalfoundation.org>

**STUDIES ON
FUZZY GRAPH THEORY**

Project Report submitted to

ST.MARY'S COLLEGE (AUTONOMOUS), THOOTHUKUDI

Affiliated to

MANONMANIAM SUNDARANAR UNIVERSITY, TIRUNELVELI

In partial fulfillment of the requirement for the award of degree of

Bachelor of Science in Mathematics

Submitted by

NAME

REG.NO.

ARUNA. A

19AUMT05

ASHINA PARVEEN. S

19AUMT06

HEMALATHA .B

19AUMT13

MARIAMMAL. P

19AUMT24

SANTHANA SUNDARI. M

19AUMT39

Under the Guidance of

Dr. G. PRISCILLA PACIFICA M.Sc., B.Ed., M.Phil., Ph.D., SET

Assistant Professor of Mathematics

St. Mary's College (Autonomous), Thoothukudi.



Department of Mathematics

St. Mary's College (Autonomous), Thoothukudi

(2021 - 2022)

CERTIFICATE

We hereby declare that the project report entitled "**STUDIES ON FUZZY GRAPH THEORY**" being submitted to **St. Mary's College (Autonomous), Thoothukudi** affiliated to **Manonmaniam Sundaranar University, Tirunelveli** in partial fulfillment for the award of degree of **Bachelor of Science in Mathematics** and it is a record of work done during the year 2021 - 2022 by the following students :

NAME

REG.NO.

ARUNA. A
ASHINA PARVEEN. S
HEMALATHA. B
MARIAMMAL. P
SANTHANA SUNDARI.M

19AUMT05
19AUMT06
19AUMT13
19AUMT24
19AUMT39

G. Priscilla Dancifia
17.05.22
Signature of the Guide

SAD
Signature of the Examiner

V. L. Stella Arputha Mary
Signature of the HOD
Dr. V.L. Stella Arputha Mary
M.Sc., M.Phil., B.Ed., Ph.D.,
Head & Asst Professor of Mathematics
St. Mary's College (Autonomous)
Thoothukudi-628 001.
Lucia Rose
Signature of the Principal
Principal
St. Mary's College (Autonomous)
Thoothukudi-628 001.

DECLARATION

We hereby declare that the project reported entitled "**STUDIES ON FUZZY GRAPH THEORY**", is our original work. It has not been submitted to any university for any degree or diploma.

A. Aruna
(ARUNA. A)

Ashina Parveen . S
(ASHINA PARVEEN. S)

B. Hemalatha
(HEMALATHA.B)

P. Mariammal
(MARIAMMAL. P)

M. Santhana Sundari
(SANTHANA SUNDARI. M)

ACKNOWLEDGEMENT

First of all, we thank Lord Almighty for showering his blessings to undergo this project.

With immense pleasure, we register our deep sense of gratitude to our guide **Dr. G.PRISCILLA PACIFICA M.Sc., B.Ed., M.Phil., Ph.D., SET** and the Head of the Department, **Dr. V. L. Stella Arputha Mary M.Sc., M.Phil., B.Ed., Ph.D.** for having imparted necessary guidelines throughout the period of our studies.

We thank our beloved Principal, **Rev. Dr. Sr. A.S.J. Lucia Rose M.Sc., M.Phil., Ph.D., PGDCA** for providing us the help to carry out our project work successfully.

Finally, we thank all those who extended their helping hands regarding this project.

STUDIES
ON
FUZZY GRAPH THEORY

CONTENTS

INTRODUCTION

1	PRELIMINARIES	2
2	FUZZY REGULAR GRAPHS	
2.1	Introduction	12
2.2	Fuzzy Regular Graphs	12
2.3	Adjacency Matrices of fuzzy Regular Graphs	16
2.4	Product of Fuzzy Regular Graphs	17
3	MATCHING IN FUZZY GRAPHS	
3.1	Introduction	19
3.2	Fuzzy Matching Sets	20
3.3	The Linear Programming Formulation	21
3.4	Perfect Fuzzy Set	23
3.5	Weighted Matching in Fuzzy Graph	25
4	S-MORPHISM IN FUZZY GRAPHS	
4.1	Introduction	27
4.2	S-Morphism in Fuzzy Graph	27
4.3	Equivalent Condition For S-Morphism	30
5	APPLICATIONS	33
6	CONCLUSION	35
7	REFERENCES	36

INTRODUCTION

In 1973, Kaufmann defined Fuzzy Graphs for the first time. Then Azriel Rosenfeld developed the theory of fuzzy graphs in 1975. Fuzzy Graph theory, a combination of graph theory and fuzzy set theory have been applied in various fields of science and engineering. We, now have several fuzzy areas like fuzzy Algebra, fuzzy Topology, fuzzy logic and fuzzy optimization. Fuzzy sets were proposed in order to give a degree of membership to an element in a given set. The traditional logical membership has only two choices, namely that an element belongs to a set or does not belong to that set. The crisp set theory is based on this logic and all results in pure mathematics are derived on this logic. The basic concepts of fuzzy sets can be found in the book “Fuzzy sets and Fuzzy logic” by George Klir and Bo Yuan. The fuzzy graph theory can be used in a wide range of domains in which information is incomplete or imprecise, such as bioinformatics.

Applications of fuzzy logic and fuzzy graph theory in Decision – making, Pattern recognition, Image processing, Control system, Neural networks, Genetic algorithm and in many other areas have been significant results.

The Project consists of four chapters.

In chapter 1, we have discussed about the basic concepts of fuzzy graphs, strength of connectedness between two vertices, path and bridges.

In chapter 2, we have discussed about Fuzzy Regular Graphs and derive several results with respect to regularity.

In chapter 3, we have discussed about the concept of Matching in Fuzzy Graphs.

In chapter 4, we have discussed about the concept of morphism, based on the strength of connectedness between two vertices.

CHAPTER 1 PRELIMINARIES

Definition 1.1: Let S be any non-empty set. A fuzzy subset of S is a mapping

$\mu: S \rightarrow [0, 1]$, where $[0, 1]$ denotes the closed interval of the set of real numbers. We say μ as a *fuzzy subset* of S

Definition 1.2: Let μ be a fuzzy subset of S . We let $\mu^t = \{x \in S / \mu(x) \geq t\}$ for all $t \in [0, 1]$. The sets μ^t are called *level sets or t-cuts* of μ .

Definition 1.3: We define the $\text{supp}(\mu)$ as the set $\{x \in S / \mu(x) > 0\}$. The height of μ is defined as $h(\mu) = \vee \{ \mu(x) / x \in S \}$ where \vee denotes supremum.

Definition 1.4: Let μ and ν be two fuzzy subsets of S . Then,

- (1) $\mu \subseteq \nu$ if $\mu(x) \leq \nu(x)$ for all $x \in S$
- (2) $\mu \subset \nu$ if $\mu(x) \leq \nu(x)$ for all $x \in S$ and there exists one $x \in S$ such that $\mu(x) < \nu(x)$.
- (3) $\mu = \nu$ if $\mu(x) = \nu(x)$ for all $x \in S$

Definition 1.5: Let μ and ν be two fuzzy subsets of S . Then $\mu \cup \nu$ is the fuzzy subset of S defined by $(\mu \cup \nu)(x) = \max\{\mu(x), \nu(x)\}$ for all $x \in S$ and $\mu \cap \nu$ is the fuzzy subset of S defined by $(\mu \cap \nu)(x) = \min\{\mu(x), \nu(x)\}$ for all $x \in S$.

Definition 1.6 : A fuzzy graph $G = (V, \mu, \rho)$ is a nonempty set V together with a pair of functions $\mu: V \rightarrow [0, 1]$ and $\rho: V \times V \rightarrow [0, 1]$ such that for all x, y in V , we have $\rho(x, y) \leq \mu(x) \wedge \mu(y)$. For simplicity, we denote the fuzzy graph by $G = (\mu, \rho)$ or by (G, μ, ρ)

Example 1.7: Let $V = \{a, b, c, d, e\}$. The values of μ are given in brackets in the following figure. The nonzero edge weights are given along the edges

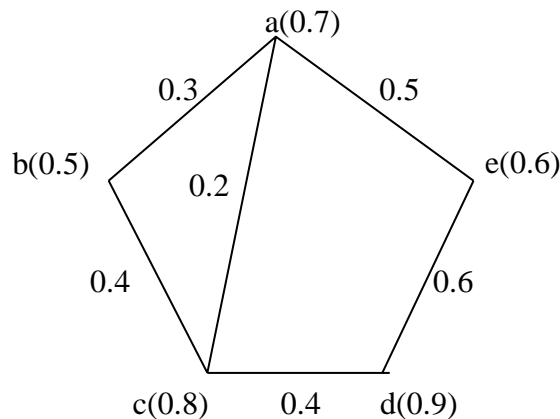


Fig :1

Definition 1.8: The fuzzy graph $H = (v, \tau)$ is called a partial fuzzy sub graph of

$G = (V, \mu, \rho)$ if $v \subseteq \mu$ and $\tau \subseteq \rho$. (i.e) $v(x) \subseteq \mu(x)$ and $\tau(x, y) \leq \rho(x, y)$)

for all $x, y \in V$. We say the partial fuzzy subgraph (v, τ) spans the fuzzy graph (μ, ρ) if $v = \mu$. In this case, we call (v, τ) a spanning fuzzy subgraph of (μ, ρ) .

Definition 1.9: Let $G = (V, \mu, \rho)$ be a fuzzy graph. For any fuzzy subset v of V such that $v \subseteq \mu$ the partial subgraph of (μ, ρ) induced by v is the maximal partial fuzzy subgraph of (μ, ρ) that has the fuzzy vertex set v .

(i.e) $\tau(x, y) = v(x) \wedge v(y) \wedge \rho(x, y)$ for all $x, y \in V$.

Matrix representation of a fuzzy graph.

A fuzzy graph can be represented by an adjacency matrix where rows and columns are indexed by the vertex set V and the $(i, j)^{\text{th}}$ entry is $\rho(x, y)$ and where $\rho(x, x) = \mu(x)$.

Since ρ is a fuzzy relation on μ it follows that any diagonal element of ρ is larger than or equal to the elements in its column. For computational purposes, we omit the diagonal entries. The fuzzy graph given in the figure 1.5.7 has matrix representation

$$\begin{pmatrix} 0.7 & 0.3 & 0.2 & 0 & 0.5 \\ 0.3 & 0.5 & 0.4 & 0 & 0 \\ 0.2 & 0.4 & 0.8 & 0.4 & 0 \\ 0 & 0 & 0.4 & 0.9 & 0.6 \\ 0.5 & 0 & 0 & 0.6 & 0.6 \end{pmatrix}$$

The Composition of two fuzzy relations is defined as follows.

The composition of two fuzzy relations μ and ρ of $S \times S$ is defined by

$$(\mu \circ \rho)(x,z) = \sup_{y \in S} \{ \mu(x,y) \wedge \rho(y,z) \}$$

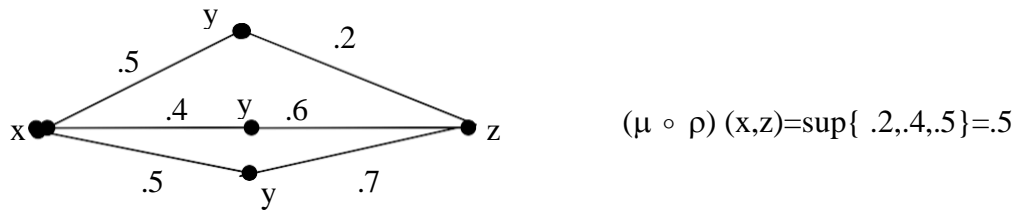


Fig (2)

As Composition of fuzzy relations is associative, we can talk about $\rho \circ \rho = \rho^2$ and other powers of ρ .

We define $\rho^\infty(x,y) = \sup \{ \rho^k(x,y) / k=1,2,3,\dots \}$. The composition can be computed, similar to matrix multiplication, where the addition is replaced by \vee

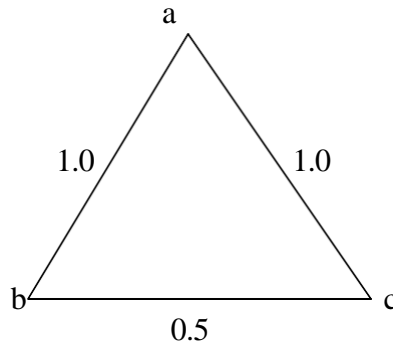
and the multiplication is replaced by \wedge . We define $\rho^0(x,y) = 0$ if $x \neq y$ and $\rho^0(x,y) = \mu(x)$ otherwise.

Definition 1.10: A path in a fuzzy graph (μ, ρ) is a sequence of distinct vertices $x_0, x_1, x_2, \dots, x_n$ such that $\rho(x_{i-1}, x_i) > 0, 1 \leq i \leq n$. The strength of the path is defined as $\Lambda\{\rho(x_{i-1}, x_i) \mid i = 1, 2, \dots, n\}$ where Λ denotes the minimum. Here 'n' is the length of the path. A single vertex is a path of length 0. A strongest path joining any two vertices has strength $\rho^\infty(x, y)$.

Definition 1.11: A partial fuzzy subgraph (μ, ρ) is said to be connected if for all $x, y \in \text{supp}(\mu)$, $\rho^\infty(x, y) > 0$.

Definition 1.12: Let $G = (V, \mu, \rho)$ be a fuzzy graph. Let x, y be two distinct vertices and let G' be the partial fuzzy subgraph of G obtained by deleting the edge (x, y) . That is, $G' = (\mu, \rho')$ where $\rho'(x, y) = 0$ and $\rho' = \rho$ for all other pairs. We say that (x, y) is a bridge in G if $\rho'^\infty(u, v) < \rho^\infty(u, v)$ for some u, v . In other words, deleting the edge (x, y) reduces the strength of connectedness between some pair of vertices.

Example 1.13:



Fig(3)

Here the edge 'ab' is a bridge, while 'bc' is not a bridge.

Definition 1.14: $G = (\mu, \rho)$ is called a tree if $(\text{supp}(\mu), \text{supp}(\rho))$ is a tree.

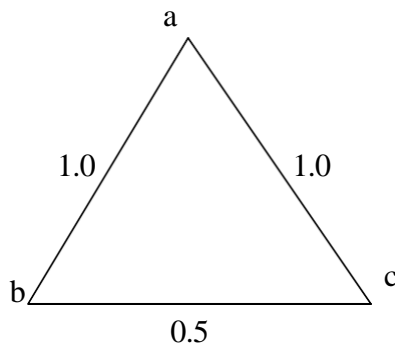
$G = (\mu, \rho)$ is called a fuzzy tree if (μ, ρ) has a fuzzy spanning subgraph (μ, ν) which is a tree such that $\forall (u, v) \in \text{supp}(\mu)$ but not in $\text{supp}(\nu)$ we have $\rho(u, v) < \nu^\infty(u, v)$. That is, there exists a path in (μ, ν) between u and v whose strength is greater than $\rho(u, v)$.

Definition 1.15: $G = (\mu, \rho)$ is called a cycle if $(\text{supp}(\mu), \text{supp}(\rho))$ is a cycle. G

$G = (\mu, \rho)$ is called a fuzzy cycle if (μ, ρ) if $(\text{supp}(\mu), \text{supp}(\rho))$ is a cycle and there does not exist a unique $(x, y) \in \text{supp}(\rho)$ such that

$$\rho(x, y) = \Lambda\{\rho(u, v) \mid (u, v) \in \text{supp}(\rho)\}.$$

Example 1.16:



Fig(4)

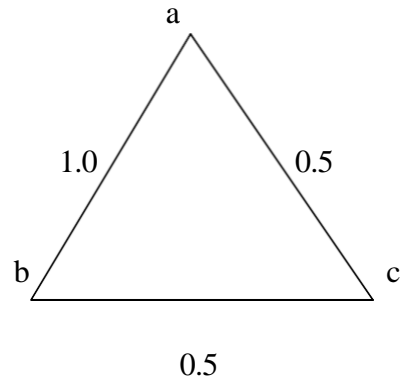
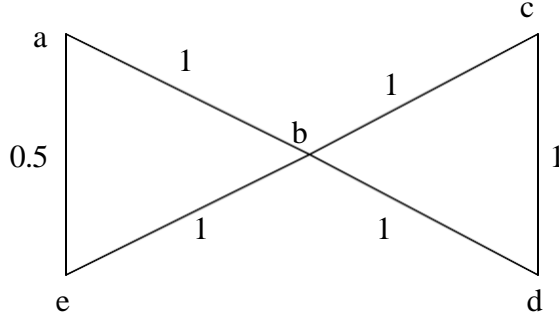


Fig (5)



Fig(6)

Definition 1.17: The complement [12] of a fuzzy graph $G = (V, \mu, \rho)$ is defined as $\bar{G} = (V, \mu, \bar{\rho})$ where $\bar{\rho}(x, y) = \mu(x) \wedge \mu(y) - \rho(x, y)$.

Example 1.18:

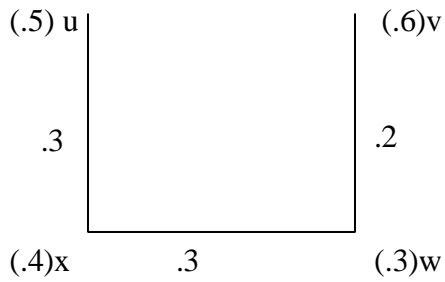


Fig (7)

G

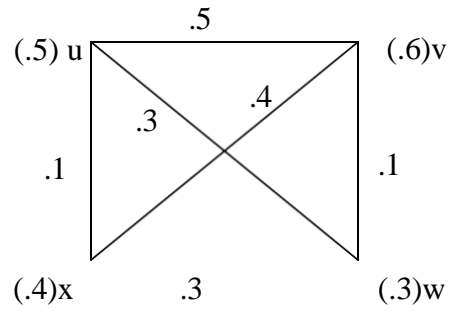


Fig (8)

\bar{G}

It is shown in [12], that $\bar{\bar{G}} = G$ and that for a self-complementary fuzzy

graph ($G = \bar{G}$), we have $\sum_{u \neq v} \rho(u, v) = \frac{1}{2} \sum_{u \neq v} (\mu(u) \wedge \mu(v))$.

Definition 1.19: The Cartesian product of two fuzzy graphs (G_1, μ_1, ρ_1) and (G_2, μ_2, ρ_2) is defined as the fuzzy graph $(G_1 \times G_2, \mu_1 \times \mu_2, \rho_1 \times \rho_2)$ where $G_1 \times G_2$ has the vertex set $V_1 \times V_2$, and the edge set is given by

$$E = \{ ((u, u_2), (u, v_2)) / u \in V_1 \text{ \& } u_2 v_2 \in E_2 \} \cup \{ ((u_1, w), (v_1, w)) / w \in V_2 \text{ \& } u_1 v_1 \in E_1 \}$$

with

$$(\mu_1 \times \mu_2)((u_1, u_2)) = \mu_1(u_1) \wedge \mu_2(u_2);$$

$$(\rho_1 \times \rho_2)((u, u_2), (u, v_2)) = \mu_1(u) \wedge \rho_2(u_2, v_2) \text{ and}$$

$$(\rho_1 \times \rho_2)((u_1, w), (v_1, w)) = \mu_2(w) \wedge \rho_1(u_1, v_1)$$

Definition 1.20: The Composition of two fuzzy graphs (G_1, μ_1, ρ_1) and (G_2, μ_2, ρ_2) is defined as the fuzzy graph $(G_1[G_2], \mu_1 \bullet \mu_2, \rho_1 \bullet \rho_2)$ where $G_1[G_2]$ has the vertex set $V_1 \times V_2$, and the edge set is given by

$$E = \{ ((u, u_2), (u, v_2)) / u \in V_1 \text{ \& } u_2 v_2 \in E_2 \} \cup \{ ((u_1, u_2), (v_1, v_2)) / u_1 v_1 \in E_1, u_2, v_2 \in V_2 \} \text{ with}$$

$$(\mu_1 \bullet \mu_2)((u_1, u_2)) = \mu_1(u_1) \wedge \mu_2(u_2);$$

$$(\rho_1 \bullet \rho_2)((u, u_2), (u, v_2)) = \mu_1(u) \wedge \rho_2(u_2, v_2) \text{ and}$$

$$(\rho_1 \bullet \rho_2)((u_1, u_2), (v_1, v_2)) = \mu_2(u_2) \wedge \mu_2(v_2) \wedge \rho_1(u_1, v_1)$$

Definition 1.21: The Tensor product of two fuzzy graphs (G_1, μ_1, ρ_1) and (G_2, μ_2, ρ_2) is defined as the fuzzy graph $(G_1 \otimes G_2, \mu_1 \otimes \mu_2, \rho_1 \otimes \rho_2)$ where the vertex set $V_1 \times V_2$, and the edge set is given by

$$E = \{((u_1, u_2), (v_1, v_2)) / u_1 v_1 \in E_1, u_2 v_2 \in E_2\} \text{ with}$$

$$(\mu_1 \otimes \mu_2)(u_1, u_2) = \mu_1(u_1) \wedge \mu_2(u_2); \text{ and}$$

$$(\rho_1 \otimes \rho_2)((u_1, u_2), (v_1, v_2)) = \rho_1(u_1, v_1) \wedge \rho_2(u_2, v_2).$$

Definition 1.22: Let $G = (V, X)$ be a graph where $V = \{v_1, v_2, \dots, v_n\}$ and

$$V_i = \{v_i, x_{i1}, x_{i2}, \dots, x_{iq_i}\} \text{ where } x_{ij} \in X \text{ and } x_{ij} \text{ has } v_i \text{ as a vertex } j=1, 2, 3, \dots, q_i,$$

$i=1, 2, 3, \dots, n$. Let $S = \{S_1, S_2, \dots, S_n\}$. Let $T = \{(S_i, S_j) / S_i, S_j \in S, S_i \cap S_j \neq \emptyset\}$.

Then (S, T) is an intersection graph. For this graph define fuzzy subsets τ, γ of S and T by $\tau(S_i) = \mu(v_i) \forall S_i$ and $\gamma(S_i, S_j) = \rho(v_i, v_j) \forall (S_i, S_j) \in T$. Then, (S, T) is called an intersection graph.

Definition 1.23: The line graph of a graph G is by definition intersection

graph Where $L(G) = \{Z, W\}$ where $Z = \{ \{x\} \cup \{u_x, v_x\} / x \in X, u_x, v_x \in V, x = (u_x, v_x) \}$ and $W = \{(S_x, S_y) / S_x \cap S_y \neq \emptyset, x \in X, x \neq y\}$. Define the fuzzy subsets

λ, ω of Z and W by $\lambda(S_x) = \rho(x)$ and $\omega(S_x, S_y) = \rho(x) \wedge \rho(y)$. Then (λ, ω) is a fuzzy sub graph of $L(G)$. called the fuzzy line sub graph corresponding to (μ, ρ) .

The following theorems will be used in subsequent chapters.

Theorem 1.24: The following are equivalent

- (1) (x,y) is a Bridge
- (2) $\rho'^{\infty}(x,y) < \rho^{\infty}(x,y)$
- (3) (x,y) is not the weakest edge of any cycle

Theorem 1.25: Let $G = (\mu, \rho)$ be a cycle. Then (μ, ρ) is a fuzzy cycle if and only if (μ, ρ) is not a fuzzy tree.

Strong arcs in fuzzy graphs

Definition 1.26: The maximum of strengths of all paths from x to y is called $\text{CONN}_G(x,y)$. We say G is connected if $\text{CONN}_G(x,y) > 0$ for all x,y

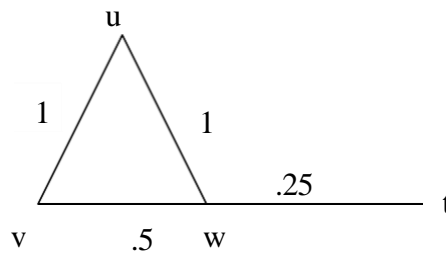
Definition 1.27: We call the arc (x,y) strong in G if $\rho(x,y) > 0$ and $\rho(x,y) \geq \text{CONN}_{G-(x,y)}(x,y)$. where $G-(x,y)$ is obtained by removing the edge (x,y) .

Definition 1.28: A path from x to y is called strong if every edge in it is strong

Note 1.29: An arc of maximum weight is strong but the converse is not true
(consider a fuzzy cycle, in which every arc is strong, even the weakest edge

Note 1.30: A path of maximum strength need not be a strong path.

Example 1.31:



The two paths u, w, t and u, v, w, t have maximum strength .25 but u, v, w, t is not a strong path since the arc (v, w) is not strong.

CHAPTER 2

FUZZY REGULAR GRAPHS

§2.1: Introduction

In this chapter, we see the concept of fuzzy regular graphs. We derive several results concerning fuzzy regular graphs. We prove if G is a fuzzy regular graph then so is its complement. We prove that Cartesian product, Tensor product, Composition of two fuzzy regular graphs is fuzzy regular. We derive a necessary and sufficient condition for a fuzzy graph to be fuzzy regular. We prove the existence of a fuzzy regular graph containing the given fuzzy graph as an induced fuzzy sub graph. Then, we define s -regular fuzzy graphs and derive a necessary and sufficient condition for a fuzzy graph to be s -regular. Finally we introduce another type of regularity called pseudo-regular graphs and relate them with fuzzy regular graphs and regular graphs.

§2.2. Fuzzy regular graphs

Definition 2.2.1: Let $G = (V, \mu, \rho)$ be a fuzzy graph. The degree of a vertex v is defined as $d(v) = \sum_{u \neq v} \rho(u, v)$. The minimum of degrees of the vertices is denoted by $\delta(G)$ and the maximum of the degrees of the vertices is denoted by $\otimes(G)$.

Definition 2.2.2: A fuzzy graph is called fuzzy regular if $d(u) = d(v)$ for all $u, v \in V, u \neq v$

Example 2.2.3

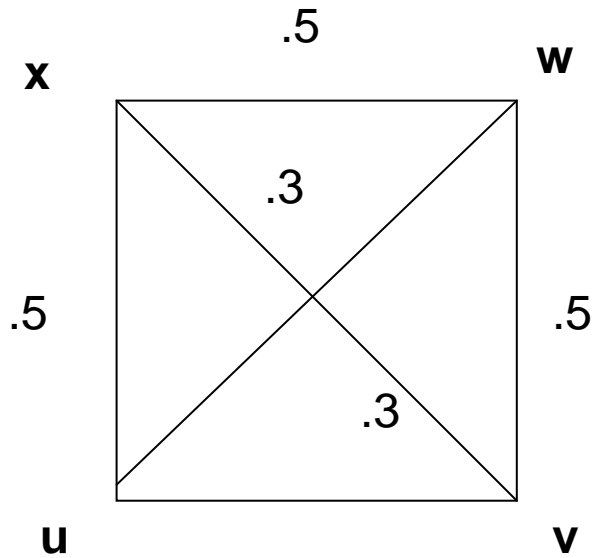


Fig:10

Example 2.2.4

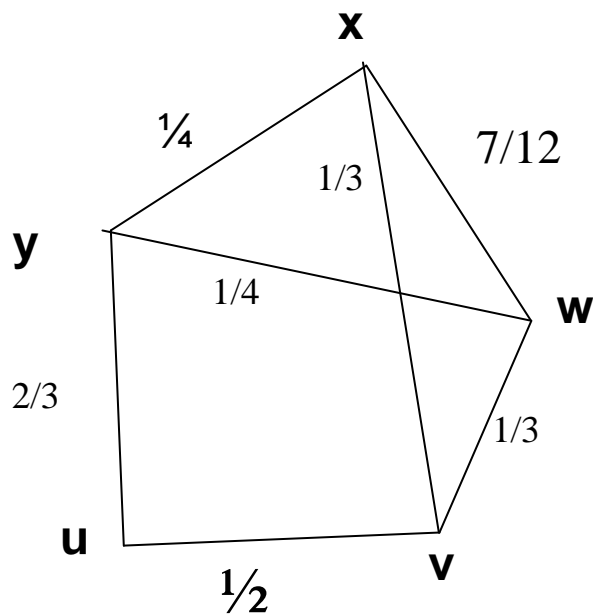


Fig:11

Example 2.2.5: We give an example of a crisp graph which cannot be fuzzy regular. The graph in the following can never be fuzzy regular

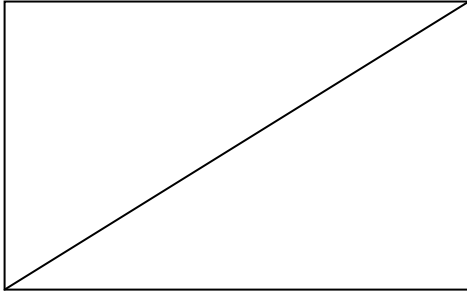


Fig:12

Here after, we assume $\mu(x) = 1$ for all the vertices x .

The following results are based on this assumption.

Theorem 2.2.6: If $G(\mu, \rho)$ is a fuzzy regular graph of degree k , then \bar{G} is fuzzy regular of degree $p-k-1$

Proof: By definition of \bar{G} , $\bar{\rho}(x,y) = \mu(x) \wedge \mu(y) - \rho(x,y) = 1 - \rho(x,y)$.

By taking the sum over the vertices y adjacent to x , we get $d(\bar{x}) = p-1-k$, where p is the number of vertices, we get the result.

Theorem 2.2.7: A necessary and sufficient condition for a fuzzy graph G to be fuzzy regular is that $\delta(G) + \delta(\bar{G}) = p-1$ where p is the no of vertices.

Proof: Suppose G is fuzzy regular of degree k then \bar{G} is fuzzy regular of degree $p-k-1$.

$$\delta(G) + \delta(\bar{G}) = k + p - k - 1 = p - 1$$

Conversely, assume $\delta(G) + \delta(\bar{G}) = p - 1$. Suppose G is not fuzzy regular then G has two vertices u and v such that $a = d(u)$ and $b = d(v)$ with $a < b$.

$$\Rightarrow \delta(G) \leq a. \text{ Now in } \bar{G}, d(v) = p - 1 - b$$

$$\Rightarrow \delta(\bar{G}) \leq p - 1 - b$$

$$\Rightarrow \delta(G) + \delta(\bar{G}) \leq a + p - 1 - b < p - 1 \text{ as } b - a > 0$$

which is a contradiction to our assumption.

Theorem 2.2.8: For every fuzzy graph $G(\mu, \rho)$, there exists a fuzzy regular graph H containing G as an induced fuzzy sub graph

Proof: Suppose G is not fuzzy regular let G' be another copy of G . We construct G_1 from G and G' by adding the edges $v_i v_i'$ for all vertices for which $d(v_i) < \Delta(G)$. Now assign the weight to edge $v_i v_i'$ by defining

$$\rho(v_i, v_i') = \min \{ 1, \Delta(G) - d(v_i) \}$$

Now G is an induced subgraph of G_1 with

$$\delta(G_1) = \min \{ 1, \min \{ 1, \Delta(G) - d(v_i) \} \}$$

Clearly, $\delta(G) < \delta(G_1) \leq \Delta(G)$.

If G_1 is fuzzy regular then G_1 is the desired graph. Otherwise, we continue this procedure till we get the required fuzzy regular graph. This process will stop in at most n steps.

§2.3 Adjacency matrices of fuzzy regular graphs

Definition 2.3.1: Let $G = (V, \mu, \rho)$ be a fuzzy regular Graph. The adjacency matrix of a such a fuzzy graph with ' n ' vertices is a $n \times n$ matrix, whose $(i, j)^{th}$ entry is $\rho(v_i, v_j)$.

2.3.2: Fuzzy regular graphs and doubly stochastic matrices.

Let $G = (V, \mu, \rho)$ be a fuzzy regular Graph. By dividing the weight of each edge by the common degree (i.e) we normalize the weights of the edges we get a fuzzy regular graph in which $d(v)=1$ for each vertex v . . Since $d(v) = 1$ for each vertex v , the sum of elements in each row and in each column of the adjacency matrix becomes one. Therefore, the adjacency matrix of such a fuzzy regular graph becomes a doubly stochastic matrix in which all the elements along the principal diagonal are zero. (i.e) Corresponding to each fuzzy regular graph on ' n ' vertices we have a doubly stochastic matrix in which all the elements along the principal diagonal are zero. Conversely, given a doubly stochastic matrix in which all the elements along the principal diagonal are zero, we can associate a

fuzzy regular graph in which the degree of each vertex is one. The following results are obvious for fuzzy regular graphs in which, $d(v)=1$ for each vertex v .

Result 2.3.3: An odd cycle is fuzzy regular if and only if the weights of all edges are equal to $\frac{1}{2}$.

Result 2.3.4: An even cycle is fuzzy Regular if and only if the weights of alternate edges are equal and the sum of weights of any two adjacent edges is one.

Result 2.3.5: A cycle which is fuzzy regular is a fuzzy cycle.

§2. 4: Product of fuzzy regular graphs

In this section, we show that the products of fuzzy regular graphs are also fuzzy regular. We assume $\mu(x) = 1$ for all vertices x . We will represent (G_1, μ_1, ρ_1) and (G_2, μ_2, ρ_2) simply as (G_1, ρ_1) and (G_2, ρ_2) . Let them have p_1, p_2 vertices. Then we have the following theorem.

Theorem 2.4.1: If (G_1, ρ_1) and (G_2, ρ_2) are fuzzy regular graphs of degree k_1, k_2 respectively then,

- (i) Their Cartesian product is also fuzzy regular of degree $k_1 + k_2$
- (ii) Their composition is also fuzzy regular of degree $k_2 + p_2 k_1$
- (iii) Their Tensor product is also fuzzy regular of degree $k_1 k_2$

Proof : i) Let $u_1, v_1 \in V_1$ and $u_2, v_2 \in V_2$ For a fixed $(u_1, u_2) \in V_1 \times V_2$

$$\sum (\rho_1 \times \rho_2) ((u_1, u_2), (v_1, v_2)) = \sum_{v_2} (\rho_1 \times \rho_2) ((u_1, u_2), (u_1, v_2)) +$$

$$\sum_{v_1} (\rho_1 \times \rho_2) ((u_1, u_2), (v_1, u_2))$$

$$= \sum_{v_2} \rho_2(u_2, v_2) + \sum_{v_1} \rho_1(u_1, v_1)$$

$$= d(u_2) \text{ in } G_2 + d(u_1) \text{ in } G_1$$

$$= k_2 + k_1$$

$$\text{ii) } \sum (\rho_1 \bullet \rho_2) ((u_1, u_2), (v_1, v_2)) = \sum_{v_2} (\rho_1 \bullet \rho_2) ((u_1, u_2), (u_1, v_2)) +$$

$$\sum_{u_1 \neq v_1} (\rho_1 \bullet \rho_2) \rho_2((u_1, u_2), (v_1, u_2))$$

$$= \sum_{v_2} \rho_2(u_2, v_2) + \sum_{v_1} \sum_{v_2} \rho_1(u_1, v_1)$$

$$= \sum_{v_2} \rho_2(u_2, v_2) + \sum_{v_1} p_2 \rho_1(u_1, v_1)$$

$$= d(u_2) + p_2 \cdot d(u_1)$$

$$= k_2 + p_2 k_1$$

$$\text{iii) } \sum (\rho_1 \otimes \rho_2) ((u_1, u_2), (v_1, v_2)) = \sum_{v_1} \sum_{v_2} \rho_1(u_1, v_1) \cdot \rho_2(u_2, v_2)$$

$$= \left(\sum_{v_1} \rho_1(u_1, v_1) \right) \cdot \left(\sum_{v_2} \rho_2(u_2, v_2) \right)$$

$$= d(u_1) d(u_2)$$

$$= k_1 k_2$$

CHAPTER 3

MATCHING IN FUZZY GRAPHS

§ 3.1 Introduction.

A fuzzy matching involving the vertex weight, the edge weight and the incidence of edges on a vertex. We define a fuzzy matching in a fuzzy graph and the fuzzy matching number of a fuzzy graph. The determination of fuzzy matching number is based on the solution of a 0,1 linear programming problem associated with it. We also define fuzzy perfect matching number of a fuzzy graph and derive three results for perfect fuzzy matching. We also define a new weight function for the edges of a fuzzy graph. This weight function depends on both the vertex weight and the edge weight.

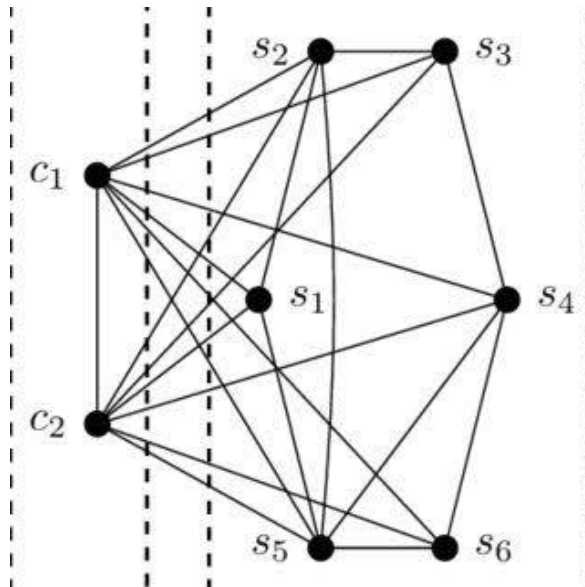


Fig :13

\$ 3.2 Fuzzy Matching sets

Definition 3.2.1: Let $G = (V, \mu, \rho)$ be a fuzzy graph with vertex set V . Let E denote the set of edges of nonzero weights. A subset M of E is called a fuzzy

Matching if for each vertex u , we have $\sum_{\substack{v \in V \\ (u,v) \in M}} \rho(u,v) \leq \mu(u)$

Definition 3.2.2: Let $G = (V, \mu, \rho)$ be a fuzzy graph. We define the fuzzy matching Number $F(G)$ as

$$F(G) = \underset{M}{\text{Max}} \left\{ \sum_{(u,v) \in M} \rho(u,v) \mid M \text{ is a Fuzzy Matching in } G \right\} \text{ and the edge set for}$$

which the maximum is attained as fuzzy matching set.

Example 3.2.3 :

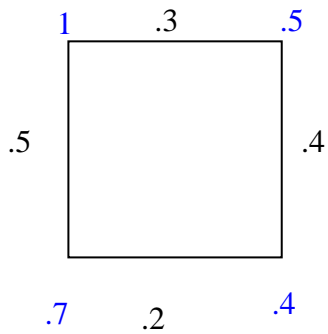


Fig: 14

The Fuzzy matching set is

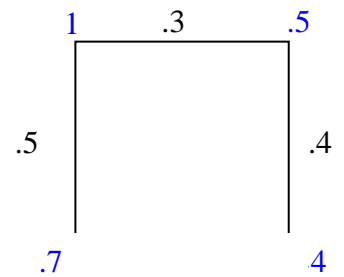


Fig: 15

$$F(G) = 1.2$$

\$ 3.3 The Linear programming formulation

We can formulate the finding of the fuzzy matching number as a 0,1 programming problem. Let E denote the incidence matrix of the fuzzy graph $G = (V, \mu, \rho)$ having n vertices and m edges. Then E is a $n \times m$ matrix whose $(i, j)^{\text{th}}$ entry is $\rho(e_j)$ if the j^{th} edge is incident on v_i and 0 otherwise.

If $X = (x_1, x_2, \dots, x_n)^T$ and $V = (\mu(v_1), \mu(v_2), \dots, \mu(v_n))^T$ and

$W = (\rho(e_1), \rho(e_2), \dots, \rho(e_m))$ then the Linear 0,1 programming problem is

Maximize WX

subject to

$$EX \leq V$$

Where, each x_i is either 1 or 0.

This is a zero-one Programming problem which can be solved by Additive Algorithm or by Branch and Bound Algorithm. We can also consider the dual of the above linear programming problem as a Vertex cover problem for the given fuzzy graph. For illustration consider the following graph.

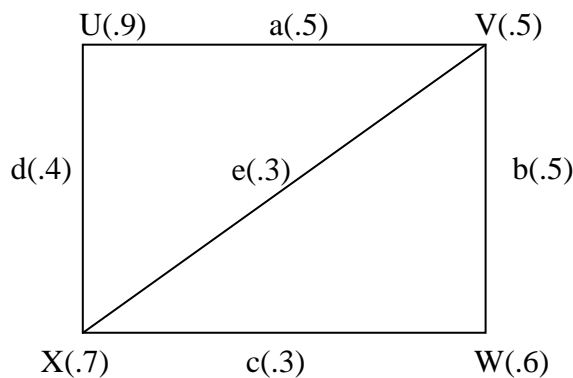


Fig: 16

The incidence matrix of the above graph is $\begin{pmatrix} .5 & 0 & 0 & .4 & 0 \\ .5 & .5 & 0 & 0 & .3 \\ 0 & .5 & .3 & 0 & 0 \\ 0 & 0 & .3 & .4 & .3 \end{pmatrix}$ where the edges

a,b,c,d and e represent the columns and the vertices U,V,W,X represent the edges.

The fuzzy matching problem is Maximize $(.5 \ .5 \ .3 \ .4 \ .3) \begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix}$ subject to

$$\begin{pmatrix} .5 & 0 & 0 & .4 & 0 \\ .5 & .5 & 0 & 0 & .3 \\ 0 & .5 & .3 & 0 & 0 \\ 0 & 0 & .3 & .4 & .3 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \\ e \end{pmatrix} \leq \begin{pmatrix} .9 \\ .5 \\ .6 \\ .7 \end{pmatrix}$$

The fuzzy matching corresponds to the solution $a=1, c=1, d=1$ and the fuzzy matching number is 1.2. Now comes the interesting part. Consider the dual of the above problem. The dual can be put in the form

$$\text{Minimize } (.9 \ .5 \ .6 \ .7) \begin{pmatrix} u \\ v \\ w \\ x \end{pmatrix} \text{ subject to } \begin{pmatrix} .5 & .5 & 0 & 0 \\ 0 & .5 & .5 & 0 \\ 0 & 0 & .3 & .3 \\ .4 & 0 & 0 & .4 \\ 0 & .3 & 0 & .3 \end{pmatrix} \begin{pmatrix} u \\ v \\ w \\ x \end{pmatrix} \leq \begin{pmatrix} .5 \\ .5 \\ .3 \\ .4 \\ .3 \end{pmatrix}. \text{ The solution}$$

is $u=1, w=1$. Note that the constraint can be put in the form

$$\begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ w \\ x \end{pmatrix} \leq \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \text{ which, is the corresponding L.P.P of finding the}$$

independence number for the crisp graph.

§ 3.4 Perfect Matching Fuzzy Set

Definition 3.4.1: A fuzzy matching M is called a perfect fuzzy matching if for each vertex u , we have

$$\sum_{\substack{v \in V \\ (u,v) \in M}} \rho(u,v) = \mu(u)$$

Example 3.4.2 :

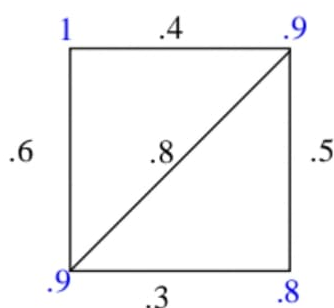


Fig :17

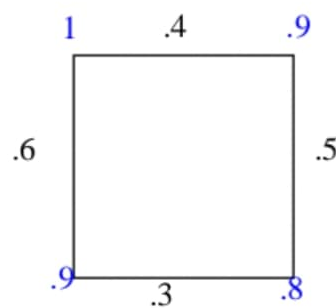


Fig: 18

The perfect Fuzzy Matching with $F(G) = 1.8$

Theorem 3.4.3: Let a fuzzy graph $G = (V, \mu, \rho)$ have a perfect fuzzy matching.

Then

$$F(G) = \frac{1}{2} \sum_{u \in V} \mu(u)$$

Proof: We have $\sum_{\substack{v \in V \\ (u,v) \in M}} \rho(u,v) = \mu(u)$. Since at every vertex we attain the Maximum

possible value for $\sum_{\substack{v \in V \\ (u,v) \in M}} \rho(u,v)$, this sum is the maximum over all possible fuzzy

matchings for that vertex. Taking the sum over all the vertices on both sides we get the result.

Theorem 3.4.4: Let $G = (V, \mu, \rho)$ be a M -strong graph. Let G have a perfect

fuzzy matching M . Then the Components of the induced crisp

subgraph of M are either K_2 or $K_{1,s}$

Proof : Let M be a perfect fuzzy matching. Take an edge e from M . If no other edges of M is incident with e than we have a component K_2 . If not, let its end vertices be u and v . Then either u is saturated by e or v is saturated by e . Without loss of generality, take u . Then no other edge of M is incident with u .

Now we may have edges of M incident at v other than e . However in this case the edges of M , incident to v will have weights which are less than weight of the vertex v . Therefore such vertices will have weight of the respective edges in M

as G is M -strong. In such cases, these vertices will be saturated by the edges of M and we cannot have edges in M incident to these vertices. Therefore In this case we have $K_{1,s}$. Hence, the result folloes..

Theorem 3.4.5: Let a fuzzy graph $G = (V, \mu, \rho)$ have a perfect fuzzy matching

M . Suppose $\mu(v)$ is the same for all the vertices v then the components of M are K_2 or disjoint cycles.

Proof : If a component of M is not K_2 then the edges of M will grow at both ends of such edges. But $\mu(v)$ is same for all the vertices v will ensure that the growth is at both ends and we get a cycle. Again these cycles will be disjoint as $\mu(v)$ is the same for all vertices.

\$ 3.5 Weighted matching in fuzzy graph:

Given a fuzzy graph we define a new function $W : E \rightarrow [0, 1]$ by

$$W(x, y) = \frac{2\rho(x, y)}{\mu(x) + \mu(y)}$$

In case $\mu(x) = 1$ for all vertices x then this reduces to just the weight of the edge. This weight function involves both the weight of the vertex and the weight of the edge. Then we can find the maximum weighted matching for the corresponding weighted graph. In case of fuzzy bipartite graphs we can find a

maximum weighted matching by solving the corresponding assignment problem.

CHAPTER 4

s-MORPHISM IN FUZZY GRAPHS

§ 4.1: Introduction

In this chapter, we see another concept of morphism, based on the strength of connectedness between two vertices. The strength of a path plays a crucial role in determining the connectedness between two vertices. Here, we define an s-morphism between two fuzzy graphs, which is based on the strength of connectedness between two vertices. This s-morphism preserves the strength between two vertices, and two graphs which are s-morphic need not have the same number of edges, even though they must have same number of vertices. This s-morphism can also be expressed in terms of matrices, which give rise to various interpretations of a fuzzy graph. After defining the s-morphism, we prove that it is an equivalence relation among the class of all fuzzy graphs with n vertices. We also prove that this s-morphism preserves tree structures, bridges and strong edges.

§ 4.2: s-Morphism in fuzzy graph

Definition 4.2.1: Let (G_1, μ_1, ρ_1) and (G_2, μ_2, ρ_2) be two fuzzy graphs.

They are said to be s-morphic if there exists a bijection $f: V_1 \rightarrow V_2$ such that

$$\rho_1^\infty(u, v) = \rho_2^\infty(f(u), f(v)) \quad \forall u, v \in V_1.$$

Example

Example 4.1 and 4.2

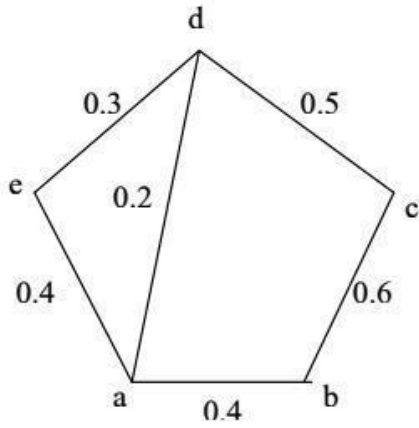


Fig (21)

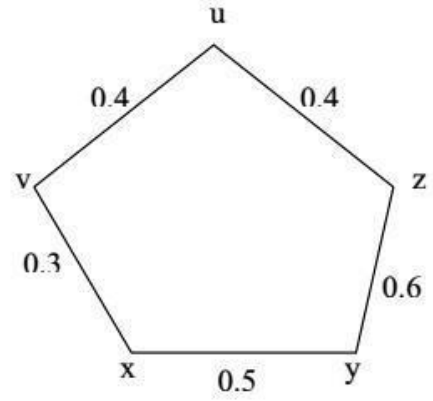


Fig (22)

The map defined by. $f(a) = u$

$$f(b) = z$$

$$f(c) = y$$

$$f(d) = x$$

$$f(e) = v$$

is a s-morphism.

Given a fuzzy graph G we can compute its strength matrix as follows. We

define the product of two adjacency matrices A and B as $C = AB$

where $C_{ij} = \max_k \{ (a_{ik}, b_{kj}) \}$

It has been proved that ,if M is the adjacency matrix of a fuzzy graph of order p then there exists a positive integer $q \leq p-1$ such that $M^q = M^{q+1} = M^{q+2} = \dots$. It has also been proved that the strength of the path connecting V_i and V_j is the $(i,j)^{th}$ entry of M^q .

The strength matrices of the above two graphs example 4.3 and 4.4 are respectively

Example

$$\begin{pmatrix} \infty & .4 & .4 & .4 & .4 \\ .4 & \infty & .5 & .5 & .4 \\ .4 & .6 & \infty & .5 & .4 \\ .4 & .6 & .5 & \infty & .4 \\ .4 & .4 & .4 & .4 & \infty \end{pmatrix}$$

and

$$\begin{pmatrix} \infty & .5 & .5 & .4 & .4 \\ .5 & \infty & .6 & .4 & .4 \\ .5 & .6 & \infty & .4 & .4 \\ .4 & .4 & .4 & \infty & .4 \\ .4 & .4 & .4 & .4 & \infty \end{pmatrix}$$

respectively

§ 4.3: Equivalent condition for s-morphism

We give an equivalent condition for two fuzzy graphs to be s-morphic.

“The fuzzy graphs(G_1, μ_1, ρ_1) and (G_2, μ_2, ρ_2) are s-isomorphic iff there exists a permutation matrix P such that $AP=PB$ (or $A= PBP^{-1}$) where A, B are strength matrices of the fuzzy graphs G_1, G_2 respectively”.

Here, the multiplication is the usual multiplication of matrices.

In the above example 4.4, the permutation matrix is $P=$

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Remark 4.3.1: Multiplying a matrix by a permutation matrix on the right interchanges the column, while multiplying on left interchanges rows. Also,

Permutation matrices are real orthogonal matrices. (i.e) $P^t = P^{-1}$

Remark 4.3.2: If two fuzzy graphs are s-isomorphic then they must have equal number of vertices but they need not have the same number of edges.

Refer Example 4.4.

Theorem 4.3.3: For a given positive integer n , “s-isomorphism” is an equivalence relation in the class of fuzzy graphs on n vertices.

Proof :

We use the condition the fuzzy graphs(G_1, μ_1, ρ_1) and (G_2, μ_2, ρ_2) are s-isomorphic

if and only if there exists a permutation matrix P such that $AP=PB$ or $A= PBP^{-1}$ where A, B are strength matrices of the graphs G_1, G_2 respectively. Clearly G_1 is s-morphic to itself as we can take P to be the identity matrix.

Let G_1 be s-morphic to G_2 . Then $A= PBP^{-1}$

$$= P^{-1} B P \text{ as } P = P^{-1} .$$

Hence G_2 is s-morphic to G_1 .

Let G_1, G_2 and G_3 be three fuzzy graphs with strength matrices A, B, C respectively. Let G_1 be s-morphic to G_2 and G_2 be s-morphic to G_3 .Then there exists a permutation matrix P such that $AP=PB$ and there exists a permutation matrix Q such that $BQ=QC$.

Now, $AP = PB \Rightarrow APQ = PBQ$

$$=PQC$$

But PQ is again a permutation matrix which implies that G_1 is s-morphic to G_3 .

Hence, the result follows.

Theorem 4.3.4: If two trees are s-morphic then they are isomorphic in the crisp sense also.

Proof : Let T_1, T_2 be two trees which are s-morphic. Then they must have equal

number of vertices. In a tree there exists only one path between any two vertices. Also, the strength of two adjacent vertices is equal to the weight of the edge joining them. Hence adjacency will be preserved. Therefore, T_1, T_2 must be isomorphic in the crisp sense also.

Note: However the result is not true in case of fuzzy trees, as given by the example 4.3

Theorem 4.3.5: Let two fuzzy graphs (G_1, μ_1, ρ_1) and (G_2, μ_2, ρ_2) be s-isomorphic

.Then there exists a bijection $f : V_1 \rightarrow V_2$ such that $\rho_1^\infty(u, v) = \rho_2^\infty(f(u), f(v))$

$\forall u, v \in V_1$ and let. Let (u, v) be a bridge. If $(f(u), f(v))$ is also an edge, then

$(f(u), f(v))$ is a bridge.

Proof : Let $f : V_1 \rightarrow V_2$ be an s-Morphism from G_1 to G_2

Let (u, v) be a bridge in G_1 . Then there exists two vertices x and y such that $\rho_1^\infty(x, y) < \rho_1^\infty(u, v)$ for some pair of vertices x and y and where $\rho_1^\infty(u, v) = 0$ and $\rho_1^\infty = \rho$ for all other vertices.

Now, $\rho_1^\infty(x, y) < \rho_1^\infty(u, v) \Rightarrow \rho_2^\infty(f(x), f(y)) < \rho_2^\infty(f(u), f(v))$ as f is an s-morphism
 $\Rightarrow (f(u), f(v))$ is a bridge in G_2 .

Theorem 4.3.6: Let two fuzzy graphs (G_1, μ_1, ρ_1) and (G_2, μ_2, ρ_2) be s-isomorphic.

If (u, v) is a strong edge in G_1 and if $(f(u), f(v))$ is an edge in G_2 then $(f(u), f(v))$ is also a strong edge in G_2 .

Proof: Let (u, v) be a strong edge in $G_1 \Rightarrow \rho_1^\infty(u, v) = \rho_1(u, v)$

$\Rightarrow \rho_2(f(u), f(v)) = \rho_1(u, v)$ as strength is preserved.

Also, $\rho_2(f(u), f(v)) \geq \rho_2(f(u), f(v))$

$\Rightarrow \rho_1^\infty(u, v) \geq \rho_2(f(u), f(v))$

$\Rightarrow (f(u), f(v))$ is strong edge.

APPLICATIONS OF FUZZY GRAPH IN VARIOUS FIELDS

➤ Utility of Fuzzy Graph in Medical Field

Utilizations of Artificial Intelligence Techniques occurred in numerous zones including medication, for example, determination, treatment of sickness, tolerant interest, and expectation of illness chance and so on. Fuzzy logic approach, as opposed to a certain or parallel rationale, utilizes a rationale and decision mechanism which doesn't have certain limits like human rationale. At the point when an individual is given a clinical assessment, a wide assortment of parameters, called side effects in clinical language, can be found out and estimated. Because of the intricacy of the human body, it is beyond the realm of imagination to expect to give a sensible utmost for the quantity of built up criteria. Fuzzy set theory rationale is a scientific control that we use every day and causes us to arrive at the structure in which we decipher our own practices. Fuzzy set theory in which esteems among genuine and bogus that is halfway valid and in part bogus are resolved. Fuzzy set theory express the vulnerabilities of life, for example, warm and cool which are in the middle of hot and cold scientifically. At the point when a specialist begins treatment of a patient he utilizes his own understanding, information from books, and mental capacity.

➤ Exploit of Fuzzy Graph in Traffic Light Control

The control strategy of the traffic light relies generally upon the quantity of vehicles in the crossing point line. On the off chance that the traffic stream in the crossing point line is high, at that point there is a chance of mishap. At the point when the quantity of vehicles in the crossing point line is low then there might be less chance of mishap. The idea of mishap and number of vehicles in each line could be fuzzy. This shouldn't be numerical, is related to the ideal security level for the traffic. Here we describe each traffic stream with a fuzzy edge whose enrolment esteem relies upon the quantity of vehicles in that way. Two fuzzy nodes are neighbouring on the off chance that the relating traffic streams cross one another; at that point there is a chance of mishap. Plausibility of mishap worth will rely upon node enrolment esteem. The most extreme security level is achieved when all paths are viewed as in crossing point with one another and the quantity of vehicles in each line is likewise high. So Graph will be a complete graph. Right now, chromatic number is the quantity of paths and the control approach of the lights guarantee that just a single development is permitted in any space of the cycle. Then again, the base security level is achieved when the crossing point edge set is unfilled, right now, chromatic number is 1 and all developments are permitted at any moment.

➤ Utilize of Fuzzy Graph in Neural Networks

Neural systems are disentangled models of the organic sensory system and in this way have drawn their motivation from the sort of registering performed by a human mind. Neural systems exhibit trademark, for example, mapping abilities or example affiliation, speculation, vigor, adaptation to internal failure, and resemble and rapid data handling. Fuzzy neural systems and neural fuzzy frameworks are ground-breaking procedures for different computational and control applications. The region is still under an extraordinary deluge from both hypothetical and applied research. There is no orderly or brought together methodology for fusing the ideas of fuzziness and neural handling. Fuzzy sets can be utilized to delineate different parts of Neural Computing. That is, fuzziness might be presented at the info yield signals, synaptic loads, and collection activity and actuation capacity of individual neurons to make it fluffy neuron. Applying fuzzy techniques into the activities of neural systems establishes a significant push of neuron-fuzzy computing. A fuzzy neuron has a similar fundamental structure as the counterfeit neuron with the exception of that its segments and parameters are depicted through the arithmetic of fuzzy logic.

CONCLUSION:

The study of fuzzy graphs made in this report is far from being complete. We sincerely hope that the wide ranging applications of graph theory and the interdisciplinary nature of fuzzy set theory, if properly blended together could pave a way for a substantial growth of fuzzy graph theory. Research on the theory of fuzzy sets has been witnessing an exponential growth; both within mathematics and in its applications. This ranges from traditional mathematical subjects like logic, topology, algebra, analysis etc. to pattern recognition, information theory, artificial intelligence, operations research, neural networks, planning etc. Consequently, fuzzy set theory has emerged as a potential area of interdisciplinary research. We hope that the growth of fuzzy graph theory will be further accelerated by the development of fuzzy software and fuzzy hardware.

BIBLIOGRAPHY

- [1] Abraham kandel, Fuzzy Mathematical Techniques with Applications, Addison Wesley Publishing Company.
- [2] Bhattacharya P."Some Remarks on Fuzzy Graphs", Pattern Recognition Letters 6:297-302,1987.
- [3] Bhutani K.R and Rosenfeld A, Strong Arcs in Fuzzy Graphs, Information Sciences 152 (2003), 319-322.
- [4] Craine W.L., Characterization of Fuzzy interval Graphs., Fuzzy Sets and Systems 68:181-193, 1994.
- [5] Goetschel R., Jr., Introduction to Fuzzy hypergraphs and hebbian Structures, Fuzzy Sets and Systems 84: 235-254, 1996.
- [6] Kaufmann A., Introduction to the Theory of Fuzzy Sets, Academic Press, NewYork, 1975.
- [7] Klir G.J. and Yuan B, Fuzzy Sets and Fuzzy Logic: Theory and Applications, Prentice Hall of India, 2002.
- [8] Mordeson J.N. and Peng, C.S, Operations on Fuzzy Graphs, Information Sciences 79:159-170, 1994 .
- [9] Rosenfeld A., Fuzzy Graphs. In: Zadeh K.S, Fu and Shimura M, Eds., Fuzzy Sets and Their Applications, Academic Press, New York, 77-95, 1975.
- [10] Vaidyanathan M, Ramakrishnan P.V, Matching in Fuzzy graphs, Proceedings of National Conference on Discrete Mathematics and its Applications, NCDMA, sep 27-29,2007 Thiagarajar College of Engineering, Madurai .

A STUDY ON FUZZY THEORIES

Project Report submitted to

ST. MARY'S COLLEGE (AUTONOMOUS), THOOTHUKUDI

Affiliated to

MANONMANIAM SUNDARANAR UNIVERSITY, TIRUNELVELI

In partial fulfillment of the requirement for the award of degree of

Bachelor of Science in Mathematics

Submitted by

NAME

REG.NO

AROCKIA CATHRINE SHARUBALA. E

19AUMT04

CAROLIN BELCIA. R

19AUMT08

MAHALAKSHMI. M

19AUMT19

PAULIN PACKIAM. B

19AUMT33

SHIBANIA. C

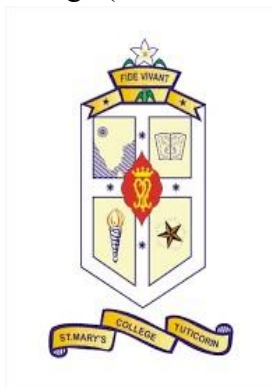
19AUMT43

Under the Guidance of

Dr. V. L. STELLA ARPUTHA MARY, M.Sc., B.Ed., M.Phil., Ph.D.

Head of the department of Mathematics

St. Mary's College (Autonomous), Thoothukudi.



Department of Mathematics

St. Mary's College (Autonomous), Thoothukudi

(2021 - 2022)

CERTIFICATE

We hereby declare that the project report entitled "A STUDY ON FUZZY THEORIES " being submitted to **St. Mary's College (Autonomous), Thoothukudi** affiliated to **Manonmaniam Sundaranar University, Tirunelveli** in partial fulfilment for the award of degree of **Bachelor of Science in Mathematics** and it is a record of work done during the year 2021 - 2022 by the following students:

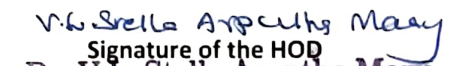
NAME

REG. NO.

AROCKIA CATHRINE SHARUBALA. E
CAROLIN BELCIA. R
MAHALAKSHMI. M
PAULIN PACKIAM. B
SHIBANIA. C

19AUMT04
19AUMT08
19AUMT19
19AUMT33
19AUMT43


Signature of the Guide


Signature of the HOD
Dr. V.L. Stella Arputha Mary
M.Sc., M.Phil., B.Ed., Ph.D.,
Head & Asst Professor of Mathematics
St. Mary's College (Autonomous)
Thoothukudi-628 001.


Signature of the Examiner


Signature of the Principal
Principal
St. Mary's College (Autonomous)
Thoothukudi - 628 001.

DECLARATION

We hereby declare that the project reported entitled "A STUDY ON FUZZY THEORIES", is our original work. It has not been submitted to any university for any degree or diploma.

E. Arockia Cathrine Sharubala
(AROCKIA CATHRINE SHARUBALA. E)

P. Carolin Belcia .
(CAROLIN BELCIA. R)

M. Mahalakshmi
(MAHALAKSHMI. M)

B. Paulin Packiam
(PAULIN PACKIAM. B)

C. Shibania
(SHIBANIA. C)

A STUDY ON FUZZY THEORIES

INTRODUCTION

The term 'FUZZY' refers to 'lacking in clarity' or 'vagueness'. Fuzziness occurs when the boundary of a piece of information is not clear-cut. For example, moving the camera causes fuzzy photos.

Fuzzy set theory: Fuzzy set theory was proposed by Prof. Lotfi A. Zadeh in 1965 as an extension of the classical notion of a set. With the proposed methodology, Zadeh introduced a mathematic method with which decision making using fuzzy descriptions of some information becomes possible. Fuzzy set theory is at once a generalization as well as extension of Crisp set theory. Thus, the basic theme and ideas of Crisp set theory will be reflected in Fuzzy set theory.

Fuzzy relation: From a historical perspective, the first fuzzy relation was mentioned in the year 1971 by Lotfi A. Zadeh. Fuzzy relation can be utilized in databases.

Fuzzy matrix: Fuzzy matrices were introduced for the first time by Thomason who discussed the convergence of power of fuzzy matrix. Fuzzy matrices play a vital role in scientific development.

Fuzzy logic: Fuzzy logic is a logic. Logic refers to the study of methods and principles of human reasoning. Any event that changes continuously we cannot define it as a true or false in such cases we can solve it by fuzzy logic. It deals with vagueness and imprecise information. It was proposed by Lotfi A. Zadeh in his paper 'Fuzzy Logic and Approximate Reasoning, Synthes, 30,1975'.

CONTENT

1. Fuzzy Set Theory	
1.1. Fuzzy Set.....	8
1.2. Operations on Fuzzy Sets.....	10
1.3. Certain Numbers Associated with a Fuzzy Sets.....	13
1.4. The Power of Fuzzy set.....	17
1.5. Extension Principle.....	18
2. Fuzzy Relation	
2.1. Definition.....	23
2.2. Operations on Fuzzy Relations.....	25
2.3. α -Cut of a Fuzzy Relations.....	26
2.4. Composition of Fuzzy Relations.....	27
2.5. Projections of Fuzzy Relations.....	30
3. Fuzzy Matrix	
3.1. Definition	32
3.2. Addition of matrices.....	32
3.3. Max - Min Composition of Matrices.....	34
3.4. Matrix Multiplication.....	35
4. Fuzzy Logics	
4.1. Logic Connectives.....	38
4.2. Three Valued Logics.....	40
4.3. N- Valued Logic for $N \geq 4$	42
4.4. Infinite Valued Logics.....	42
4.5. Fuzzy Logics.....	43
5. Conclusion.....	45
6. Reference.....	46

1. FUZZY SET THEORY

The concept of set is the building block of mathematics. In fact, the whole edifice of mathematics is constructed out of it. This concept is so fundamental and all-pervading that it is absolutely essential to have a firm and clear understanding of the theory of sets.

A set means any well-defined collection of objects. This, however, is not at all defined it is what is called an undefined term. Another undefined term is a

member or an element of a set. We express the relation between an object and a set to which it belongs by writing $a \in A$. The symbol \in is read as “belongs to”, “lies in” etc.

Crisp Set:

Crisp set is a collection of unordered distinct elements, which are derived from Universal set. In the context of fuzzy sets theory, we often refer crisp sets.

Characteristic function:

Crisp Set Theory can also be studied via characteristic function.

Definition:

Let U be a fixed non-empty set, to be called the universal set or universe of discourse or simply domain. Define,

$$F: U \rightarrow \{0,1\}$$

f is called characteristic function on U . The set of all such functions is denoted as $CH(U)$. Each element of $CH(U)$ is called a CH on U .

1.1. FUZZY SETS:

The concept of a fuzzy set is an extension of the concept of a crisp set. Just as a crisp set on a universal is defined by its characteristic function from U to $\{0,1\}$, a fuzzy set on a domain U is defined by its membership function from U to $[0,1]$. Let U be a non - empty set to be called the universal set or the universe of discourse or simply a domain. Then by a fuzzy set on U is meant a function,

$$A: U \rightarrow [0,1].$$

‘ A ’ is called the membership function, $A(x)$ is called the membership grade of x . $A = \{(x, A(x)): x \in U\}$. We represent the unit interval $[0,1]$ by I .

Membership Function:

The membership function fully defines the fuzzy set. A membership function provides a measure of the degree of similarity of an element to a fuzzy set. It can be either be chosen by the user arbitrarily, based on the user's experience or be designed using machine learning methods.

Example:

Consider $U = \{a, b, c, d\}$ and $A: U \rightarrow I$ defined by

$$A(a) = 0.0, \quad A(b) = 0.7,$$

$$A(c) = 0.4, \quad A(d) = 1$$

Then A is a fuzzy set on U .

$$A = \{(a,0), (b,0.7), (c,0.4), (d,1)\}$$

Fuzzy Power Set:

Let U be a domain. The set of all fuzzy sets on U is denoted by $PF(U)$ is called the Fuzzy Power Set of U .

$$PF(U) = \{A \mid A: U \rightarrow I\}$$

1.1.1 RELATION BETWEEN FUZZY SETS:

Let U be a domain and A, B be fuzzy sets on U .

Containment or Inclusion:

A is said to be included or contained in B if and only if $A(x) \leq B(x)$ for all x in U . We write as $A \subseteq B$. We also say that A is a subset of B .

Equality:

A is said to be equal to B or same as B if and only if $A \subseteq B$ and $B \subseteq A$,

$$\text{i.e.) } A(x) = B(x), \text{ for all } x \in U.$$

We write as $A=B$.

These two relations satisfy the following properties:

Let A, B, C be fuzzy sets on U . Then,

$$1. A \subseteq A.$$

2. $A \subseteq B$ and $B \subseteq A$ imply $A = B$.
3. $A \subseteq B$ and $B \subseteq A$ imply $A = B$.
4. $A = A$
5. $A = B$ imply $B = A$.
6. $A = B$ and $B = C$ imply $A = C$.

1.2. OPERATIONS ON FUZZY SETS:

Let U be a domain and A, B be fuzzy sets on U . Then,

Union:

Union of A and B , denoted by $A \cup B$, is defined as that fuzzy set on U for which,

$$(A \cup B)(x) = \max(A(x), B(x)), \text{ for every } x \in U.$$

Intersection:

Intersection of A and B , denoted by $A \cap B$ is defined as that fuzzy set on U for which,

$$(A \cap B)(x) = \min(A(x), B(x)), \text{ for every } x \in U.$$

Complement:

Complement of A , denoted by A' , defined as the fuzzy set on U for which,

$$(A')x = 1 - A(x), \text{ for every } x \text{ in } U.$$

Example:

- 1) Let $U = \{a, b, c, d\}$ be the domain and A and B be fuzzy sets on U as given a

	a	b	c	d
A	0.5	0.8	0.0	0.3
B	0.2	1.0	0.1	0.7

For $A \cup B$,

$$\begin{aligned} (A \cup B)(a) &= \max[A(a), B(a)] \\ &= \max[0.5, 0.2] \\ &= 0.5 \end{aligned}$$

$$(A \cup B)(b) = \max[A(b), B(b)]$$

$$= \max [0.8, 1.0]$$

$$= 1.0$$

$$(A \cup B)(c) = \max [A(c), B(c)]$$

$$= \max [0.0, 0.1]$$

$$= 0.1$$

$$(A \cup B)(a) = \max [A(c), B(c)]$$

$$= \max [0.3, 0.7]$$

$$= 0.7$$

Thus,

	a	b	c	d
AUB	0.5	1.0	0.1	0.7

For $A \cap B$,

$$(A \cap B)(a) = \min [A(a), B(a)]$$

$$= \min [0.5, 0.2]$$

$$= 0.2$$

$$(A \cap B)(b) = \min [A(b), B(b)]$$

$$= \min [0.8, 1.0]$$

$$= 0.8$$

$$(A \cap B)(c) = \min [A(c), B(c)]$$

$$= \min [0.0, 0.1]$$

$$= 0.0$$

$$(A \cap B)(d) = \min [A(d), B(d)]$$

$$= \min [0.3, 0.7]$$

$$= 0.3$$

Thus,

	a	b	c	d
$A \cap B$	0.2	0.8	0.0	0.3

For A' ,

$$A'(a) = 1 - A(a)$$

$$= 1 - 0.5$$

$$= 0.5$$

$$A'(b) = 1 - A(b)$$

$$= 1 - 0.8$$

$$= 0.2$$

$$A'(c) = 1 - A(c)$$

$$= 1 - 0.0$$

$$= 1.0$$

$$A'(d) = 1 - A(d)$$

$$= 1 - 0.3$$

$$= 0.7$$

Thus,

	a	b	c	d
A'	0.5	0.2	1.0	0.7

For B' ,

$$B'(a) = 1 - B(a)$$

$$= 1 - 0.2$$

$$= 0.8$$

$$B'(b) = 1 - B(b)$$

$$= 1 - 1.0$$

$$= 0.0$$

$$B'(c) = 1 - B(c)$$

$$= 1 - 0.1$$

$$= 0.9$$

$$B'(d) = 1 - B(d)$$

$$= 1 - 0.7$$

$$= 0.3$$

Thus,

	a	b	c	d
B'	0.2	1.0	0.1	0.3

1.3. CERTAIN NUMBERS ASSOCIATED WITH A FUZZY SETS:

Let A be a fuzzy set on U. Then by the scalar cardinality of A, we mean then number $\sum A(x)$ where the summation is over all the elements of U or more generally, the summation is over the support of A [supp(A)]. This makes sense only when U is a finite set or more generally, the support of A is finite. This number is denoted by |A| or SC (A).

Example:

$$1) A = (0.1, 0.8, 0.2)$$

$$SC(A) = 1.1$$

i) Height of a fuzzy set:

Let A be a fuzzy set on U. Then the height of A is defined to be that number ht (A) which is such that:

$$i) A(x) \leq \text{ht} (A), \text{ for all } x \text{ in } \text{supp}(A).$$

$$ii) A(x) = \text{ht} (A) \text{ for all least one } x \text{ in } \text{supp}(A).$$

This can be compactly expressed as follows:

$$\text{ht} (A) = \max \{A(x) | x \text{ in } \text{supp}(A)\}.$$

When supp(A) is finite. When supp (A) is not finite, we write,

$$\text{ht} (A) = \text{supremum} \{A(x) | x \text{ in } \text{supp}(A)\}.$$

And include the condition $A(x) = \text{ht}(A)$ for at least one x, explicitly in order

to exclude certain pathological cases like the following fuzzy set:

$$A(x) = 1 - e^{-x}, \text{ for } x \geq 0$$

$$A(x) = 0, \text{ for } x \leq 0.$$

For this fuzzy set, height is 1, but there is no value of x for which $A(x)=1$ and also that $ht(A)$ always lies between 0 and 1.

Example:

$$1. \text{ If } A = (0.0, 0.2, 0.8)$$

$$\text{Then } ht(A)=0.8$$

$$2. \text{ If } A = (0.0, 0.2, 0.6)$$

$$\text{Then } ht(A)=0.6$$

ii) Normal fuzzy set:

Let A be a fuzzy set on U . Then A is said to be normal if $A(x) = 1$ for at least one x in U . In other word, $ht(A) = 1$.

Example:

$$1. \text{ All non-empty crisp sets are normal.}$$

Certain fuzzy set can be converted into a normal fuzzy set. This procedure is called normalization of a fuzzy set.

iii) Normalization of a fuzzy set:

Let A be a non-empty fuzzy set on U . Let $A_N(x) = A(x)/ht(A)$ for all x in U . Then A_N is a fuzzy set on U , called the normalized version of A . Note that $ht(A_N)=1$, so that A_N is normal. Note that A_N is always a fuzzy set. This process associates a fuzzy set with a given fuzzy set. Note also that if A is normal, then $A_N = A$, i.e., an already normal fuzzy set is not affected by normalization.

Example:

$$1. \text{ For } A = (0.0, 0.2, 0.8), ht(A)=0.8 \text{ and hence } A_N = (0.0, 0.25, 1.0)$$

$$2. \text{ For } A = (0.0, 0.2, 1.0), ht(A)=1 \text{ and hence } A_N = (0.0, 0.2, 1.0)$$

iv) Support of a fuzzy set:

Let A be a fuzzy set on U . The set $\{x \in U \mid A(x) > 0\}$ is called the support of A and is denoted by $supp(A)$.

Remark:

$$1. \text{ } supp(A) \text{ is a crisp set on } U, \text{ for all fuzzy set } A.$$

$$2. \text{ } supp(A) = A \text{ for any crisp set } A.$$

3. for a genuine fuzzy set A, $A \subset \text{supp}(A)$

Before giving the definition of α -cuts of a fuzzy set, we deal with level set associated with a fuzzy set.

v) Level set associated with a fuzzy set:

With every fuzzy set A on U, we associate $L(A)$, a crisp subset on $I=[0,1]$ called its level set. $L(A)$ is defined as follows:

$$L(A) = \{ \alpha \in I / A(x) = \alpha, \text{ for some } x \in U \}.$$

Example:

1. Let $A = \{0.8, 0.0, 1.0, 0.4\}$

Then $L(A) = \{0.4, 0.8, 1.0\}$.

2. Let A be given by

$$A(x) = 1 - e^{-x}, \text{ for } x \geq 0$$

$$A(x) = 0, \text{ for } x \leq 0.$$

Then, $L(A) = [0,1]$

vi) α -Cuts of a fuzzy set:

Given a fuzzy set A on U and a number α in I, such that $0 < \alpha \leq 1$. We can associate crisp set with A, denoted by A_α and defined as

$A_\alpha = \{x \in U \mid A(x) \geq \alpha\}$. A_α is called the α -cuts of A. Thus, for each α , we obtain an α -cuts of A.

Example:

Let U be the set $\{a, b, c, d\}$ and A be given by $A = (0.8, 1.0, 0.3, 0.1)$. Then,

$$A_{1.0} = (0, 1, 0, 0) = \{b\}$$

$$A_{0.8} = (1, 1, 0, 0) = \{a, b\}$$

$$A_{0.3} = (1, 1, 1, 0) = \{a, b, c\}$$

$$A_{0.1} = (1, 1, 1, 1) = U$$

More generally,

$$\text{When } 0 < \alpha \leq 0.1, A_\alpha = A_{0.1}$$

$$\text{When } 0.1 < \alpha \leq 0.3, A_\alpha = A_{0.3}$$

When $0.3 < \alpha \leq 0.8$, $A_\alpha = A_{0.8}$

When $0.8 < \alpha \leq 1.0$, $A_\alpha = A_{1.0}$

This happens when the domain U is a countable set.

vii) Fuzzy cardinality of a fuzzy set:

We now introduce an important concept associated with a given fuzzy set namely, its fuzzy cardinality.

Let A be a non-empty fuzzy set on U and $\text{supp}(A)$ be finite. Its fuzzy cardinality, denoted by $\text{FC}(A)$, is defined as the fuzzy set on N ((the set of all-natural numbers.) given by $\sum \alpha / \text{SC}(A_\alpha)$ that is

$\text{FC}(A) = \sum \alpha / n_\alpha$ where $n_\alpha = \text{SC}(A_\alpha)$ and \sum runs over all α in $L(A)$.

Example:

1. For $A = (0, 0.3, 0.2, 0.8, 0.1)$, $L(A) = \{0.1, 0.2, 0.3, 0.8\}$. Thus the α -Cuts of A are $A_{0.1} = \{0, 1, 1, 1, 1\}$; $A_{0.2} = \{0, 1, 1, 1, 0\}$;

$A_{0.3} = \{0, 1, 0, 1, 0\}$; $A_{0.8} = \{0, 0, 0, 1, 0\}$;

Further, $\text{SC}(A_{0.1}) = 4$

$\text{SC}(A_{0.2}) = 3$

$\text{SC}(A_{0.3}) = 2$

$\text{SC}(A_{0.8}) = 1$

Thus, $\text{FC}(A) = \frac{0.8}{1} + \frac{0.3}{2} + \frac{0.2}{3} + \frac{0.1}{4}$

2. If $A = \{a\}$ is a crisp singleton set, then $L(A) = \{1\}$. Hence, there is only one non-empty α -cut, $A_{1.0} = \{a\}$. Therefore, $\text{SC}(A_{1.0}) = 1$ and $\text{FC}(A) = 1.0/1$. Thus, $\text{FC}(A) = \{1\}$ is a crisp singleton on N. The fuzzy cardinality gives a mapping from $\text{PF}(U)$ to $\text{PF}(N)$.

viii) Fuzzification of a fuzzy set:

We now take up the method of fuzzification of a given fuzzy set. Let U be a domain and for every x in U let a fuzzy set K(x) on U be given, then for any fuzzy set A on U, we define the fuzzification of A,

$$F(A) = \sum A(x)K(x)$$

Here, \sum stands for the union over elements of U and $A(x)F(x)$ stands for usual product of numbers. Note, that $F(A)$ is a fuzzy set on U . The collection $\{K(x) | x \in U\}$ is called the kernel of fuzzification.

Example:

Let U be $\{a,b,c,d\}$,

$$A = \frac{0.3}{a} + \frac{0.6}{b}$$

$$K(a) = \frac{0.7}{a} + \frac{0.4}{b}$$

$$K(b) = \frac{0.4}{a} + \frac{1.0}{b} + \frac{0.4}{c}$$

$$K(c) = \frac{0.2}{b} + \frac{0.8}{c}$$

Then

$$\begin{aligned} K(A) &= A(a)K(a) + A(b)K(b) + A(c)K(c) \\ &= 0.3 \times \left[\frac{0.7}{a} + \frac{0.4}{b} \right] + 0.6 \times \left[\frac{0.4}{a} + \frac{1.0}{b} + \frac{0.4}{c} \right] + 0 \\ &= \frac{0.24}{a} + \frac{0.6}{b} + \frac{0.24}{c} \end{aligned}$$

1.4.THE POWER OF A FUZZY SET:

For a fuzzy set A on U , and a positive real number α , we define the α -th power of A (denoted by A^α),

$$A^\alpha(x) = [A(x)]^\alpha \text{ for all } x \text{ in } U$$

The following special cases are quite important in applications:

i)Concentration of A:

It is denoted by $\text{con}(A)$ and is given by

$$\text{con}(A)(x) = [A(x)]^2 \text{ for } x \text{ in } U$$

$$\text{con}(A)(x) = A^2 \text{ for } x \text{ in } U$$

That is, $\text{con}(A) = A^2$

ii) Dilation of A:

This is denoted by $\text{Dil}(A)$ and is given by $\text{Dil}(A) = [A(x)]^{0.5}$ for all x in U . That is,

$$\text{Dil}(A) = A^{0.5}$$

iii) Contrast Intensification of a Fuzzy set:

The contrast intensification of a fuzzy set A , denoted by $\text{Int}(A)$, is defined as:

$$\begin{aligned} \text{Int}(A)(x) &= 2[A(x)]^2 \text{ for } 0 \leq A(x) \leq 0.5 \\ &= 1 - 2[1 - A(x)]^2 \text{ for } 0.5 \leq A(x) \leq 1 \end{aligned}$$

1.5. EXTENSION PRINCIPLE:

The extension principle is a basis principle by means of which certain mathematical concepts pertaining to the crisp side can be generalized to the fuzzy framework (Notable exception are union, intersection and complement operators of crisp sets.)

The extension principle was introduced by Zadeh in his paper ‘The Concept of a Linguistic Variable and its Application to Approximate Reasoning’. A further elaboration of this principle was presented by R.R. Yager in ‘A Characterization of the Extension Principle’. The details of this principle are as follows:

Let f be a function from $U_1 \times U_2 \times U_3 \times \cdots \times U_n$ (a Cartesian product of n domains) to V . Let $A_1, A_2, A_3, \dots, A_n$ be fuzzy sets on $U_1, U_2, U_3, \dots, U_n$ respectively. Then, extension principle indicates a method of associating a fuzzy set B on V based on the given information or inputs. This fuzzy set B is given by

$$B(v) = 0, \text{ if } f^{-1}(v) \text{ is empty}$$

$= \max [\min \{A_1(u_1), A_2(u_2), \dots, A_n(u_n)\}]$ if $f^{-1}(v)$ is non-empty where the max is taken over all n -tuples $(u_1, u_2, u_3, \dots, u_n)$ in $U_1 \times U_2 \times U_3 \times \cdots \times U_n$ whose image is v under f , i.e., all n -tuples such that $f(u_1, u_2, u_3, \dots, u_n) = v$.

We write,

$$B = f(A) \text{ or } B = f \circ A$$

Where, $A = A_1 \times A_2 \times A_3 \times \cdots \times A_n$ and

$A(u_1, u_2, u_3, \dots, u_n) = \min [A_1(u_1), A_2(u_2), \dots, A_n(u_n)]$ is a fuzzy set on

$$U = U_1 \times U_2 \times U_3 \times \cdots \times U_n.$$

A is called the cartesian product of the fuzzy sets $A_1, A_2, A_3, \dots, A_n$. The fuzzy set B is called the image of A under f.

We close this section with two examples illustrating the extension principle. The first one is a straightforward example which helps to clarify and fix the concepts. The second one illustrates the idea of extending the addition of real numbers to ‘addition’ of fuzzy sets.

Example 1

Consider the three domains U, V, and W where $U = \{a, b, c\}$, $V = \{x, y, z\}$ and $W = \{p, q, r\}$. Consider the function $f: U \times V \rightarrow W$

where $f(a, x) = f(a, y) = f(c, y) = p$,

$f(a, z) = f(b, x) = f(b, z) = q$ and

$f(b, y) = f(c, x) = f(c, z) = r$. This can be expressed compactly in the form of a table:

U \ V	x	y	z
a	p	p	q
b	q	r	q
c	r	p	r

Consider the fuzzy sets A and B on U and V respectively, where

$$A = \frac{0.2}{a} + \frac{0.7}{b} + \frac{0.5}{c}$$

$$B = \frac{0.5}{x} + \frac{0.3}{y} + \frac{1.0}{z}$$

Then, the values of $C(p)$, $C(q)$ and $C(r)$, where $C = f(A \times B)$ the image fuzzy set on W, are given by

$$i) \quad f^{-1}(p) = \{(a, x), (a, y), (c, y)\}$$

$$C(p) = \max [\min \{A(a), B(x)\}, \min \{A(a), B(y)\}, \min \{A(c), B(y)\}]$$

$$= \max [\min \{0.2, 0.5\}, \min \{0.2, 0.3\}, \min \{0.5, 0.3\}]$$

$$= \max [0.2, 0.2, 0.3]$$

$$C(p) = 0.3$$

$$\text{ii) } f^{-1}(q) = \{(a, z), (b, x), (b, z)\}$$

$$C(q) = \max [\min \{A(a), B(z)\}, \min \{A(b), B(x)\}, \min \{A(b), B(z)\}]$$

$$= \max [\min \{0.2, 1.0\}, \min \{0.7, 0.5\}, \min \{0.7, 1.0\}]$$

$$= \max [0.2, 0.5, 0.7]$$

$$C(q) = 0.7$$

$$\text{iii) } f^{-1}(r) = \{(b, y), (c, x), (c, z)\}$$

$$C(r) = \max [\min \{A(b), B(y)\}, \min \{A(c), B(x)\}, \min \{A(c), B(z)\}]$$

$$= \max [\min \{0.7, 0.3\}, \min \{0.5, 0.5\}, \min \{0.5, 1.0\}]$$

$$= \max [0.3, 0.5, 0.5]$$

$$C(r) = 0.5$$

Thus,

$$C = \frac{0.3}{p} + \frac{0.7}{q} + \frac{0.5}{r}$$

Example 2

Consider the three domains U, V and W where $U=V=W=N$, the set of natural numbers.

Let $f: U \times V \rightarrow W$ be given by $f(m, n) = m + n$, i.e., f is the addition operation on natural numbers.

Let A and B be fuzzy sets on U and V respectively, given by

$$A = \frac{0.2}{2} + \frac{0.8}{3} + \frac{0.7}{4} \quad \text{and}$$

$$B = \frac{0.7}{3} + \frac{0.6}{4} + \frac{0.5}{5}$$

Observe that $\text{supp}(A) = \{2, 3, 4\}$ and $\text{supp}(B) = \{3, 4, 5\}$. Thus, the various pairs in $U \times V$ that are to be considered lie in $\text{supp}(A) \times \text{supp}(B)$. These pairs and their images under f are given in the following table:

f	3	4	5
2	5	6	7
3	6	7	8
4	7	8	9

$$i) f^{-1}(7) = \{(2, 5), (3, 4), (4, 3)\}$$

$$\begin{aligned} C(7) &= \max [\min \{A(2), B(5)\}, \min \{A(3), B(4)\}, \min \{A(4), B(3)\}] \\ &= \max [\min \{0.2, 1.0\}, \min \{0.7, 0.5\}, \min \{0.7, 1.0\}] \\ &= \max [0.2, 0.5, 0.7] \end{aligned}$$

$$C(7) = 0.7$$

$$ii) f^{-1}(5) = \{(2, 3)\}$$

$$\begin{aligned} C(5) &= \max [\min \{A(2), B(3)\}] \\ &= \max [\min \{0.2, 0.7\}] \\ &= \max [0.2] \end{aligned}$$

$$C(5) = 0.2$$

$$iii) f^{-1}(6) = \{(2, 4), (3, 3)\}$$

$$\begin{aligned} C(6) &= \max [\min \{A(2), B(4)\}, \min \{A(3), B(3)\}] \\ &= \max [\min \{0.2, 0.6\}, \min \{0.8, 0.7\}] \\ &= \max [0.2, 0.7] \end{aligned}$$

$$C(6) = 0.7$$

$$iv) f^{-1}(8) = \{(4, 4), (3, 5)\}$$

$$\begin{aligned} C(8) &= \max [\min \{A(4), B(4)\}, \min \{A(3), B(5)\}] \\ &= \max [\min \{0.7, 0.6\}, \min \{0.8, 0.5\}] \\ &= \max [0.6, 0.5] \end{aligned}$$

$$C(8) = 0.6$$

$$v) f^{-1}(9) = \{(4, 5)\}$$

$$\begin{aligned}
C(9) &= \max [\min \{A(4), B(5)\}] \\
&= \max [\min \{0.7, 0.5\}] \\
&= \max [0.5]
\end{aligned}$$

$$C(9) = 0.5$$

Therefore,

$$C = \frac{0.2}{5} + \frac{0.7}{6} + \frac{0.7}{7} + \frac{0.6}{8} + \frac{0.5}{9}$$

Since f represents addition, $C=f(A, B)$ can be written as:

$$C = A+B$$

Where ‘+’ denotes addition of fuzzy sets or if we prefer, can be called fuzzy addition.

2.FUZZY RELATION

2.1 DEFINITION:

Fuzzy relation defines the mapping of variables from one **fuzzy set** to another. Like **crisp relation**, we can also define the relation over fuzzy set.

Let A be a fuzzy set on universe X and B be a fuzzy set on universe Y, then the Cartesian product between fuzzy sets A and B will result in a fuzzy relation R which is contained with the full Cartesian product space or it is subset of cartesian product of fuzzy subsets. Formally, we can define fuzzy relation as,

$$R = A \times B$$

And

$$R \subset (X \times Y)$$

where the relation R has membership function,

$$\mu_R(x, y) = \mu_{A \times B}(x, y) = \min(\mu_A(x), \mu_B(y))$$

An **n-ary** fuzzy relation R is a fuzzy set on $U_1 \times U_2 \times \dots \times U_n$ where U_1, U_2, \dots, U_n are domains.

A **2-ary fuzzy relation** is also called a **binary fuzzy relation**. A binary fuzzy relation (BFR) looks like

$$R = \sum_{(u,v)} \frac{R(u,v)}{(u,v)}$$

where (u, v) varies over $U \times V$.

We say that R is from U to V and is indicated by $R: U \rightarrow V$

A **3-ary fuzzy relation** is also called a **ternary fuzzy relation**. A 3-ary fuzzy relation looks like

$$T = \sum_{(u,v,w)} \frac{t(u,v,w)}{(u,v,w)}$$

where the triplets (u, v, w) vary over $U \times V \times W$.

Example

1. Let $U = \{a, b, c\}$ and $V = \{x, y\}$. Then a binary fuzzy relation on $U \times V$ is given by

R	x	y
a	0.6	1.0
b	0.3	0.5
c	0.4	0.2

This is called the tabular or matrix representation of R and it is very useful when dealing with binary fuzzy relations.

2. $U = \{a, b, c\}$, $V = \{x, y\}$ and $W = \{\&, *\}$. Then a fuzzy relation on $U \times V \times W$ is given by

$$T = \frac{0.21}{(a, x, \&)} + \frac{0.38}{(b, y, \&)} + \frac{0.9}{(a, y, *)}$$

We can express T which is a ternary fuzzy relation in the tabular form: one matrix for & and one for *.

&	x	y
a	0.21	0
b	0	0.38
c	0	0

*	x	y
a	0	0.9
b	0	0
c	0	0

3. U and V be the set of real numbers. Then the relation 'y is smaller than x' is a **binary fuzzy relation** on $U \times V$. A representation of this fuzzy relation is given by

$$R(x, y) = \begin{cases} 0 & \text{if } y \geq x \\ \frac{1}{1 + (x - y)^{-2}} & \text{if } y < x \end{cases}$$

2.2. OPERATIONS ON FUZZY RELATIONS :

Let U_1, U_2, \dots, U_n be domains and let $U = U_1 \times U_2 \times \dots \times U_n$. Then U is also a domain, by definition of cartesian product and $u \in U$ looks like $u = (u_1, u_2, \dots, u_n)$, an n -tuple. We thus have $PF(U) = PF(U_1 \times U_2 \times \dots \times U_n)$. This equality shows that every n -ary fuzzy relation (FR) on $U_1 \times U_2 \times \dots \times U_n$, is a fuzzy set on U and vice-versa.

Let U be $U_1 \times U_2 \times \dots \times U_n$.

1.Equality:

For R, S in $PF(U)$, we say $R = S$ if and only if $R(u) = S(u)$ for all u in U

2. Containment:

For R, S in $PF(U)$, we say $R \subseteq S$ if and only if $R(u) \leq S(u)$ for all u in U .

3.Union:

For R, S in $PF(U)$, the union of R and S , denoted by $R \cup S$, is defined by

$(R \cup S)(u) = \max[R(u), S(u)]$ for every u in U .

Example:

$$\begin{aligned} \text{Let } A &= \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ B &= \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ (A \cup B)(u) &= \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

4.Intersection:

For R, S in $PF(U)$, the intersection of R and S , denoted by $R \cap S$, is defined by $(R \cap S)(u) = \min[R(u), S(u)]$ for every u in U .

Example:

$$\text{Let } A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$(A \cap B)(u) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

5.Complement:

For R in $P(FU)$. R' is defined by $R'(u) = 1-R(u)$ for every w in U .

Example:

$$\text{Let } A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$A'(u) = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2.3. α -CUTS OF A FUZZY RELATION:

Let R be a fuzzy relation (FR) on $U \times V$ and α be such that $0 < \alpha \leq 1$. Then, the α -cut of R , denoted by R_α is defined by

$$R_\alpha = \{ (u, v) \mid R(u, v) \geq \alpha \}$$

Note that R_α is a crisp set on $U \times V$ and hence is a crisp (binary) relation on $U \times V$.

The α -cuts of R satisfy the following property, called the decomposition theorem or resolution form of R . Let R be a fuzzy relation on $U \times V$. Then $R = \sum(\alpha R_\alpha)$ where

\sum is taken over all α . The following example illustrates the above point.

Let R be a fuzzy relation on $U \times V$ given by the matrix

$$R = \begin{bmatrix} 0.7 & 0.4 \\ 0.4 & 0.0 \end{bmatrix}$$

$$\text{Then, } R_{0.4} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \quad \text{and} \quad R_{0.7} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

$$(0.4 \times R_{0.4}) \cup (0.7 \times R_{0.7}) = \begin{bmatrix} 0.4 & 0.4 \\ 0.4 & 0 \end{bmatrix} \cup \begin{bmatrix} 0.7 & 0 \\ 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0.7 & 0.4 \\ 0.4 & 0.0 \end{bmatrix}$$

This verifies the above theorem.

Remark:

The α -cut decomposition can be obtained directly by applying the maximum principle. Again, consider R as above. Consider the largest entry 0.7 of R and write R as:

$$R = 0.7 \times \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \cup \begin{bmatrix} 0.4 & 0.4 \\ 0.4 & 0 \end{bmatrix}$$

Now, apply this principle again to obtain

$$R = 0.7 \times \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \cup 0.4 \times \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

which is the α -cut decomposition of R.

2.4. COMPOSITION OF FUZZY RELATIONS:

Composition of two relations can be defined in several ways.

- Max-min composition
- Max product composition

Max-min Composition of Two Fuzzy Relations:

Let R be a binary fuzzy relation (BFR) on $U \times V$ and S be a BFR on $V \times W$. Then, the max-min composition of R and S (that is, composition of R followed by S) is a BFR on $U \times W$, denoted by SoR and is given by

$$(SoR)(u, w) = \max [\min \{R(u, v), S(v, w)\}]$$

where the maximum is taken over all v in V .

2. More generally, let R be in $PF(U \times V)$ and S be in $PF(V \times W)$. where now

$$U = U_1 \times U_2 \times \dots \times U_k$$

$$V = V_1 \times V_2 \times \dots \times V_m$$

and

$$W = W_1 \times W_2 \times \dots \times W_n$$

Then, the max-min composition of R and S, denoted by SoR , is a fuzzy relation on $U \times W$ and is given by

$$(SoR)(u, w) = \max \{ \min (R(u, v), S(v, w)) \}$$

Where now

$$u = (u_1, u_2, \dots, u_k)$$

$$v = (v_1, v_2, \dots, v_m)$$

$$w = (w_1, w_2, \dots, w_n)$$

The maximum is taken over all v in V .

Note that R is a $(k + m)$ -ary fuzzy relation, S is an $(m + n)$ -ary fuzzy relation and V is the common domain (or, called the linking domain) of R and S . This is called the compatibility condition for composition. And finally, SoR is a $(k + n)$ -ary fuzzy relation.

Examples:

Consider the fuzzy relations R on $U \times V$ and S on $V \times W$, where $U = \{a, b, c\}$, $V = \{x, y, z\}$ and $W = \{\&, *\}$ given in matrix form by

$$R = \begin{bmatrix} 1.0 & 0.4 & 0.5 \\ 0.3 & 0.0 & 0.7 \\ 0.6 & 0.8 & 0.2 \end{bmatrix} \quad S = \begin{bmatrix} 0.7 & 0.1 \\ 0.2 & 0.9 \\ 0.8 & 0.4 \end{bmatrix}$$

Then SoR can be defined and it is fuzzy relation on $U \times W$. Now

$$\begin{aligned} (SoR)(a, \&) &= \max [\min \{R(a, v), S(v, \&)\}], \text{ for every } v \text{ in } V \\ &= \max [\min \{R(a, x), S(x, \&)\}, \min \{R(a, y), S(y, \&)\}, \min \{R(a, z), S(z, \&)\}] \\ &= \max [\min (1, 0.7), \min (0.4, 0.2), \min (0.5, 0.8)] \\ &= \max (0.7, 0.2, 0.5) \\ &= 0.7 \end{aligned}$$

$$\begin{aligned} (SoR)(a, *) &= \max [\min \{R(a, v), S(v, *)\}], \text{ for every } v \text{ in } V \\ &= \max [\min \{R(a, x), S(x, *)\}, \min \{R(a, y), S(y, *)\}, \min \{R(a, z), S(z, *)\}] \\ &= \max [\min (1.0, 1.0), \min (0.4, 0.9), \min (0.5, 0.4)] \\ &= \max [1, 0.4, 0.4] \\ &= 0.4 \end{aligned}$$

$$\begin{aligned} (SoR)(b, \&) &= \max [\min \{R(b, v), S(v, \&)\}], \text{ for every } v \text{ in } V \\ &= \max [\min \{R(b, x), S(x, \&)\}, \min \{R(b, y), S(y, \&)\}, \min \{R(b, z), S(z, \&)\}] \\ &= \max [\min (0.3, 0.2), \min (0.0, 0.2), \min (0.7, 0.8)] \\ &= \max [0.2, 0.0, 0.7] \\ &= 0.7 \end{aligned}$$

$$\begin{aligned}
(\mathbf{SoR}) (\mathbf{b}, *) &= \max [\min \{R (\mathbf{b}, v), S (v, *)\}], \text{ for every } v \text{ in } V \\
&= \max [\min \{R (\mathbf{b}, x), S (x, *)\}, \min \{R (\mathbf{b}, y), S (y, *)\}, \min \{R (\mathbf{b}, z), S (z, *)\}] \\
&= \max [\min (0.3, 0.1), \min (0.0, 0.9), \min (0.7, 0.4)] \\
&= \max [0.1, 0.0, 0.4] \\
&= 0.4
\end{aligned}$$

$$\begin{aligned}
(\mathbf{SoR}) (\mathbf{c}, \&) &= \max [\min \{R (\mathbf{c}, v), S (v, \&)\}], \text{ for every } v \text{ in } V \\
&= \max [\min \{R (\mathbf{c}, x), S (x, \&)\}, \min \{R (\mathbf{c}, y), S (y, \&)\}, \min \{R (\mathbf{c}, z), S (z, \&)\}] \\
&= \max [\min (0.6, 0.7), \min (0.8, 0.2), \min (0.2, 0.8)] \\
&= \max [0.6, 0.2, 0.2] \\
&= 0.6
\end{aligned}$$

$$\begin{aligned}
(\mathbf{SoR}) (\mathbf{c}, *) &= \max [\min \{R (\mathbf{c}, v), S (v, *)\}], \text{ for every } v \text{ in } V \\
&= \max [\min \{R (\mathbf{c}, x), S (x, *)\}, \min \{R (\mathbf{c}, y), S (y, *)\}, \min \{R (\mathbf{c}, z), S (z, *)\}] \\
&= \max [\min (0.6, 0.1), \min (0.8, 0.9), \min (0.2, 0.4)] \\
&= \max [0.1, 0.8, 0.2] \\
&= 0.8
\end{aligned}$$

$$\mathbf{SoR} = \begin{bmatrix} 0.7 & 0.4 \\ 0.7 & 0.4 \\ 0.6 & 0.8 \end{bmatrix}$$

The Max-product composition:

It can be defined as:

If R is a fuzzy relation on $U \times V$ and S is a fuzzy relation on $V \times W$ then the max product composition of R followed by S, denoted again by

$$(\mathbf{SoR}) (u, w) = \max [R (u, v) * S (v, w)]$$

Where ‘*’ is the ordinary product of real numbers and ‘max’ is taken over all elements v in V

This special case deals with the composition, of a fuzzy relation, as explained in the following definition.

Let A be a fuzzy set on U and R be a fuzzy relation on $U \times V$, where

$V = V_1 \times V_2 \times \dots \times V_n$. Then the composition of A followed by R, also called the image of A under R, denoted by RoA and defined as

$$(RoA)(v) = \max [\min \{A(u), R(u,v)\}]$$

Where ‘max’ is taken over all u in U. RoA is an n-ary fuzzy relation on V (in case $n=1$ it is a fuzzy set on V). We can in a similar way, define the max – product composition of A and R

2.5. PROJECTIONS OF FUZZY RELATION:

Definition:

Let R be a BFR on $U \times V$. Then, by the first projection of R or the projection of R on U or the shadow of R on U, we mean the fuzzy set on U given by

$$\max \{R(u, v) \mid \text{for all } v \text{ in } V\}.$$

We denote this projection by $\text{Proj}[R : U]$ or by $[R \uparrow U]$ or simply R_1 , (that is. R projected on to the first domain). Similarly, we can talk about the projection of R on V, defined by $\max \{R(u, v) \mid \text{all } u \text{ in } U\}$. This is denoted by $\text{Proj}[R : V]$ or $[R \downarrow V]$ or R_2 . (that is, projection on the second domain).

EXAMPLE:

Let $U = \{a, b, c\}$ and $V = \{x, y\}$. Here R is given by

$$R = \begin{bmatrix} 0.6 & 1.0 \\ 0.3 & 0.5 \\ 0.4 & 0.2 \end{bmatrix}$$

To determine R_1 , we need to compute $R_1(a)$, $R_1(b)$, $R_1(c)$.

$$\begin{aligned} R_1(a) &= \max \{R(a, v) \mid \text{for all } v \text{ in } V\} \\ &= \max \{R(a, x), R(a, y)\} \\ &= \max \{0.6, 1\} \\ &= 1 \end{aligned}$$

$$\begin{aligned} R_1(b) &= \max \{R(b, v) \mid \text{for all } v \text{ in } V\} \\ &= \max \{R(b, x), R(b, y)\} \\ &= \max \{0.3, 0.5\} \\ &= 0.5 \end{aligned}$$

$$\begin{aligned} R_1(c) &= \max \{R(c, v) \mid \text{for all } v \text{ in } V\} \\ &= \max \{R(c, x), R(c, y)\} \end{aligned}$$

$$= \max \{0.4, 0.2\}$$

$$= 0.4$$

$$\mathbf{R}_1 = (1, 0.5, 0.4)$$

Similarly, we have by looking at column maxima

$$R_2(x) = \max \{R(x,u) | \text{for all } u \text{ in } U\}$$

$$= \max \{R(x,a), R(x,b), R(x,c)\}$$

$$= \max \{0.6, 0.3, 0.4\}$$

$$= 0.6$$

$$R_2(y) = \max \{R(y,u) | \text{for all } u \text{ in } U\}$$

$$= \max \{R(y,a), R(y,b), R(y,c)\}$$

$$= \max \{1.0, 0.5, 0.2\}$$

$$= 1.0$$

$$\mathbf{R}_2 = (0.6, 1)$$

3. FUZZY MATRICES

Fuzzy matrix, we mean a matrix over a fuzzy algebra. We confine with matrices over the fuzzy algebra $\mathcal{F} = [0,1]$ under the max-min operations and with the usual ordering on real numbers. Fuzzy matrices have quite different properties from matrices over a field, due to fact that addition in a fuzzy algebra does not form a group, every fuzzy linear transformation on V_n can be represented by a unique fuzzy matrix. One of the most important ways to study a fuzzy matrix is to consider its row space that subspace of V_n spanned by its rows.

3.1. DEFINITION :

Let \mathcal{F}_{mn} denote the set of all $m \times n$ matrices over \mathcal{F} . If $m=n$ in shorts we write \mathcal{F}_n elements of \mathcal{F}_{mn} are called membership value matrices, binary fuzzy relation matrices (or) in short, fuzzy matrices. Matrices over the Boolean algebra $\{0,1\}$ are special type of fuzzy matrices.

Let $A = (a_{ij}) \in \mathcal{F}_{mn}$. Then the element a_{ij} is called (i,j) entry of A . Let A_i^* (A^*_j) denote the i^{th} row (column) of A . The row space $\mathbb{R}(A)$ of A is the subspace of V_n generated by the rows $\{A_i^*\}$ of A . The column space $\wp(A)$ of A is the subspace of V_m generated by the columns $\{A^*_j\}$ of A . The null space or Kernel of A is the $\{x/xA = 0\}$. Note that a row (column) vector is just an element of V_n (\mathbb{F}^n).

The $n \times m$ zero matrix O is the matrix all of whose entries are zero. The $n \times n$ identity matrix I is the matrix (δ_{ij}) such $\delta_{ij} = 1$ if $i=j$ and $\delta_{ij}=0$ if $i \neq j$. Then $n \times m$ universal matrix J is the matrix all of whose entries are 1.

Since the order of a matrix is clear from the context, most of the time suppress the order of the matrix.

3.2. ADDITION OF MATRICES:

Let $A = (a_{ij}) \in \mathcal{F}_{mn}$ and $B = (b_{ij}) \in \mathcal{F}_{mn}$. Then the $A + B = (\sup\{a_{ij}, b_{ij}\}) \in \mathcal{F}_{mn}$ is called the sum of A and B .

Example:

$$\text{If } A = \begin{bmatrix} 0.5 & 0 & 1 \\ 0.8 & 0.2 & 0.3 \\ 0 & 0.6 & 0.1 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.2 & 0.4 & 0.6 \\ 0.5 & 0.3 & 0.3 \\ 0.7 & 0.8 & 0 \end{bmatrix}$$

$$A+B = (\sup \{a_{ij}, b_{ij}\})$$

$$A+B = \begin{bmatrix} 0.5 & 0 & 1 \\ 0.8 & 0.2 & 0.3 \\ 0 & 0.6 & 0.1 \end{bmatrix} + \begin{bmatrix} 0.2 & 0.4 & 0.6 \\ 0.5 & 0.3 & 0.3 \\ 0.7 & 0.8 & 0 \end{bmatrix}$$

$$A+B = \begin{bmatrix} \sup \{a_{11}, b_{11}\} & \sup \{a_{12}, b_{12}\} & \sup \{a_{13}, b_{13}\} \\ \sup \{a_{21}, b_{21}\} & \sup \{a_{22}, b_{22}\} & \sup \{a_{23}, b_{23}\} \\ \sup \{a_{31}, b_{31}\} & \sup \{a_{32}, b_{32}\} & \sup \{a_{33}, b_{33}\} \end{bmatrix}$$

$$= \begin{bmatrix} \sup \{0.5, 0.2\} & \sup \{0, 0.4\} & \sup \{1, 0.6\} \\ \sup \{0.8, 0.5\} & \sup \{0.2, 0.3\} & \sup \{0.3, 0.3\} \\ \sup \{0, 0.7\} & \sup \{0.6, 0.8\} & \sup \{0.1, 0\} \end{bmatrix}$$

$$A+B = \begin{bmatrix} 0.5 & 0.4 & 1 \\ 0.8 & 0.3 & 0.3 \\ 0.7 & 0.8 & 1 \end{bmatrix}$$

Let $A = (a_{ij}) \in \mathcal{F}_{mn}$ and $C \in \mathcal{F}$ then the fuzzy multiplication, that is scalar multiplication with scalars restricted to \mathcal{F} is defined as

$$CA = (\inf \{c, a_{ij}\}) \in \mathcal{F}_m$$

For the universal matrix J , $CJ = (\inf \{C, 1\})$ is the constant matrix all of whose entries are C . Further under component wise multiplication.

$$CJ \odot A = (\inf \{c, a_{ij}\}) = CA$$

PREPOSITION:

The set \mathcal{F}_{mn} is a fuzzy vector space under the operations defined as $A+B = (\sup \{a_{ij}, b_{ij}\})$ and $CA = (\inf \{c, a_{ij}\})$ for $A = (a_{ij})$, $B = (b_{ij}) \in \mathcal{F}_{mn}$

Proof :

For ,

$$A, B, C \in \mathcal{F}_{mn} ,$$

$$A+B = B+A \in \mathcal{F}_{mn} \quad (\text{Commutativity})$$

$$A+(B+C) = (A+B) + C \quad (\text{Associativity})$$

For all $A \in \mathcal{F}_{mn}$, there exists an element $0 \in \mathcal{F}_{mn}$ such that $A+0 = A$.

For $C \in \mathcal{F}$,

$$\begin{aligned} c(A+B) &= cJ \odot (A+B) \\ &= (cJ \odot A) + (cJ \odot B) \\ &= cA + cB \end{aligned}$$

For $c_1, c_2 \in \mathcal{F}$,

$$\begin{aligned} (c_1+c_2) A &= (c_1 + c_2) J \odot A \\ &= (c_1J + c_2J) \odot A \\ &= (c_1 J \odot A) + (c_2J \odot A) \\ &= c_1A + c_2A \end{aligned}$$

Hence \mathcal{F}_{mn} is a vector space over \mathcal{F} . In particular for $m=1$

3.3. MAX - MIN COMPOSITION OF MATRICES:

For $A = (a_{ij}) \in \mathcal{F}_{mp}$ and $B = (b_{ij}) \in \mathcal{F}_{pn}$, the max-min produce

$$AB = (\sup \inf \{a_{ik}, b_{jk}\}) \in \mathcal{F}_{mn}.$$

The product AB defined if and only if the number of columns of A is the same as the number of rows of B . A are said to be comfortable for multiplication.

Example:

$$A = \begin{bmatrix} 0.8 & 0.1 \\ 0.2 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.6 & 0.5 \\ 0.4 & 0.3 \end{bmatrix}$$

$$AB = \begin{bmatrix} 0.8 & 0.1 \\ 0.2 & 1 \end{bmatrix} \begin{bmatrix} 0.6 & 0.5 \\ 0.4 & 0.3 \end{bmatrix}$$

$$= \begin{bmatrix} [0.8 \ 0.1] \begin{pmatrix} 0.6 \\ 0.4 \end{pmatrix} & [0.8 \ 0.1] \begin{pmatrix} 0.5 \\ 0.3 \end{pmatrix} \\ [0.2 \ 1] \begin{pmatrix} 0.6 \\ 0.4 \end{pmatrix} & [0.2 \ 1] \begin{pmatrix} 0.5 \\ 0.3 \end{pmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} \sup\{\inf\{0.8, 0.6\}, \inf\{0.8, 0.5\}\} & \sup\{\inf\{0.8, 0.5\}, \inf\{0.1, 0.3\}\} \\ \sup\{\inf\{0.2, 0.6\}, \inf\{1, 0.4\}\} & \sup\{\inf\{0.2, 0.5\}, \inf\{1, 0.3\}\} \end{bmatrix}$$

$$= \begin{bmatrix} \sup\{0.6, 0.1\} & \sup\{0.5, 0.1\} \\ \sup\{0.2, 0.4\} & \sup\{0.2, 0.3\} \end{bmatrix}$$

$$AB = \begin{bmatrix} 0.6 & 0.5 \\ 0.4 & 0.3 \end{bmatrix}$$

3.4. MATRIX MULTIPLICATION:

Matrix multiplication is not in general commutative, that is, $AB \neq BA$. Further $AB = 0$ need not imply $A = 0$ (or) $B = 0$ as in the case of real matrices

Example:

$$A = \begin{bmatrix} 0.8 & 0.1 \\ 0.2 & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} 0.6 & 0.5 \\ 0.4 & 0.3 \end{bmatrix}$$

$$C = \begin{bmatrix} 0.6 & 0.2 \\ 0.7 & 0.3 \end{bmatrix}$$

$$AB = (\sup\{\inf\{a_{ik}, b_{kj}\}\}) \in \mathcal{F}_{mn}$$

$$AB = \begin{bmatrix} 0.8 & 0.1 \\ 0.2 & 1 \end{bmatrix} \begin{bmatrix} 0.6 & 0.5 \\ 0.4 & 0.3 \end{bmatrix}$$

$$\begin{aligned}
&= \begin{bmatrix} [0.8 & 0.1]_{(0.4)}^{(0.6)} & [0.8 & 0.1]_{(0.3)}^{(0.5)} \\ [0.2 & 1]_{(0.4)}^{(0.6)} & [0.2 & 1]_{(0.3)}^{(0.5)} \end{bmatrix} \\
&= \begin{bmatrix} \sup\{\inf\{0.8, 0.6\}, \inf\{0.1, 0.4\}\} & \sup\{\inf\{0.8, 0.5\}, \inf\{0.1, 0.3\}\} \\ \sup\{\inf\{0.2, 0.6\}, \inf\{1, 0.4\}\} & \sup\{\inf\{0.2, 0.5\}, \inf\{1, 0.3\}\} \end{bmatrix} \\
&= \begin{bmatrix} \sup\{0.6, 0.1\} & \sup\{0.5, 0.1\} \\ \sup\{0.2, 0.4\} & \sup\{0.2, 0.3\} \end{bmatrix} \\
AB &= \begin{bmatrix} 0.6 & 0.5 \\ 0.4 & 0.3 \end{bmatrix} \\
BA &= \begin{bmatrix} [0.6 & 0.5]_{(0.2)}^{(0.8)} & [0.6 & 0.5]_{(1)}^{(0.1)} \\ [0.4 & 0.3]_{(0.2)}^{(0.8)} & [0.4 & 0.3]_{(1)}^{(0.1)} \end{bmatrix} \\
&= \begin{bmatrix} \sup\{\inf\{0.6, 0.8\}, \inf\{0.5, 0.2\}\} & \sup\{\inf\{0.6, 0.1\}, \inf\{0.5, 1\}\} \\ \sup\{\inf\{0.4, 0.8\}, \inf\{0.3, 0.2\}\} & \sup\{\inf\{0.4, 0.1\}, \inf\{0.3, 1\}\} \end{bmatrix} \\
&= \begin{bmatrix} \sup\{0.6, 0.2\} & \sup\{0.1, 0.5\} \\ \sup\{0.4, 0.2\} & \sup\{0.1, 0.3\} \end{bmatrix} \\
BA &= \begin{bmatrix} 0.6 & 0.5 \\ 0.4 & 0.3 \end{bmatrix} \\
BC &= \begin{bmatrix} 0.6 & 0.5 \\ 0.4 & 0.3 \end{bmatrix} \begin{bmatrix} 0.6 & 0.2 \\ 0.7 & 0.3 \end{bmatrix} \\
&= \begin{bmatrix} [0.6 & 0.5]_{(0.7)}^{(0.6)} & [0.6 & 0.5]_{(0.3)}^{(0.2)} \\ [0.4 & 0.3]_{(0.7)}^{(0.6)} & [0.4 & 0.3]_{(0.3)}^{(0.2)} \end{bmatrix} \\
&= \begin{bmatrix} \sup\{\inf\{0.6, 0.6\}, \inf\{0.5, 0.7\}\} & \sup\{\inf\{0.6, 0.2\}, \inf\{0.5, 0.3\}\} \\ \sup\{\inf\{0.4, 0.6\}, \inf\{0.3, 0.7\}\} & \sup\{\inf\{0.4, 0.2\}, \inf\{0.3, 0.3\}\} \end{bmatrix} \\
&= \begin{bmatrix} \sup\{0.6, 0.5\} & \sup\{0.2, 0.3\} \\ \sup\{0.4, 0.3\} & \sup\{0.2, 0.3\} \end{bmatrix} \\
BC &= \begin{bmatrix} 0.6 & 0.3 \\ 0.4 & 0.3 \end{bmatrix}
\end{aligned}$$

$$CB = \begin{bmatrix} 0.6 & 0.2 \\ 0.7 & 0.3 \end{bmatrix} \begin{bmatrix} 0.6 & 0.5 \\ 0.4 & 0.3 \end{bmatrix}$$

$$= \begin{bmatrix} [0.6 & 0.2]_{(0.4)}^{(0.6)} & [0.6 & 0.2]_{(0.3)}^{(0.5)} \\ [0.7 & 0.3]_{(0.4)}^{(0.6)} & [0.7 & 0.3]_{(0.3)}^{(0.5)} \end{bmatrix}$$

$$= \begin{bmatrix} \sup\{\inf\{0.6, 0.6\}, \inf\{0.2, 0.4\}\} & \sup\{\inf\{0.6, 0.5\}, \inf\{0.2, 0.3\}\} \\ \sup\{\inf\{0.7, 0.6\}, \inf\{0.3, 0.4\}\} & \sup\{\inf\{0.7, 0.5\}, \inf\{0.3, 0.3\}\} \end{bmatrix}$$

$$= \begin{bmatrix} \sup\{0.6, 0.2\} & \sup\{0.5, 0.2\} \\ \sup\{0.6, 0.3\} & \sup\{0.5, 0.3\} \end{bmatrix}$$

$$CB = \begin{bmatrix} 0.6 & 0.5 \\ 0.4 & 0.5 \end{bmatrix}$$

$$BC = \begin{bmatrix} 0.6 & 0.3 \\ 0.4 & 0.3 \end{bmatrix} \neq \begin{bmatrix} 0.6 & 0.5 \\ 0.4 & 0.5 \end{bmatrix} = CB$$

4. FUZZY LOGIC

Any event that changes continuously we cannot define it as a true or false in such cases we can solve it by fuzzy logic. It deals with vagueness and imprecise information.

4.1. LOGIC CONNECTIVES (Negation, Conjunction, Disjunction)

Truth table for Negation:

P	$\neg P$
T	F
F	T

Truth table for conjunction:

P	Q	$P \wedge Q$
T	T	T
T	F	F
F	T	F
F	F	F

Truth table for Disjunction:

P	Q	$P \vee Q$
T	T	T
T	F	T
F	T	T
F	F	F

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$
T	T	F	T	T
F	T	T	F	T
T	F	F	F	T
F	F	T	F	F

Conditional or Bi conditional:

Let P and Q be any two statements. Then the statement $P \Rightarrow Q$ which is read as if P then Q or P implies Q is called a conditional statement. The truth tables of $P \Rightarrow Q$ is F when Q has truth values F and P the truth values F and P the truth values T; in all other cases $P \Rightarrow Q$ has truth values T. P is called antecedent and Q is called consequent in P Q. The truth table for $P \Rightarrow Q$ is given as follows:

P	Q	$P \Rightarrow Q$
T	T	T
T	F	F
F	T	T
F	F	T

For any two statement P and Q the statement $P \Leftrightarrow Q$ is called a Biconditional. This is read as P if and only if Q and abbreviated as P iff Q this is also called P is necessary and sufficient for Q. $P \Leftrightarrow Q$ has the truth values T whenever both P and Q have identical truth values. The table is given as follows:

P	Q	$P \Leftrightarrow Q$
T	T	T
T	F	F
F	T	F
F	F	T

4.2. THREE-VALUED LOGICS:

The classical logic is two-valued: the values being True and False. Aristotle, who was the founder of the two-valued logic, felt that this assumption is not justified and raised doubts about this assumption of two truth values.

To tackle such situations. Lukasiewicz suggested in 1920. a 3-valued logic. In this logic everything is same as in the 2-valued logic, except that there are three truth values: The May be and False. These linguistic values are usually represented by $1, \frac{1}{2}$ and 0.2 respectively. In this 3-valued logic, denoted by L_3 . the truth value of any statement can be either 1 or $\frac{1}{2}$ or 0. i.e. $T(p) = 1$ or $\frac{1}{2}$ or 0. We define three operations on the statements p, q, r denoted by $p \vee q, p \wedge q$ and $\neg p$ analogous to the three operations OR, AND and NOT of classical logic. These are defined by their truth values as follows

$$T(p \vee q) = \max \{T(p), T(q)\}$$

$$T(p \wedge q) = \min \{T(p), T(q)\}$$

$$T(\neg p) = 1 - T(p)$$

Lukasiewicz also defined the implication operation by

$$T(p \rightarrow q) = 1 - T(p) + T(q), \text{ if } T(p) > T(q)$$

Or simply as: $= 1$, if $T(p) \leq T(q)$

Using this formula, we can write down the truth table of \rightarrow . This is given in the table,

\rightarrow	1	$\frac{1}{2}$	0
1	1	$\frac{1}{2}$	0
$\frac{1}{2}$	1	$\frac{1}{2}$	$\frac{1}{2}$
0	1	1	1

This 3-valued logic has many distinguishing and surprising features. One of them is that the WFF $[p \vee (\neg p)]$ is NOT a tautology (Note, however, that $[p \vee (\neg p)]$ is a tautology in the classical logic). This can be seen by writing the truth table of this WFF as given in Table

P	$\neg p$	$[p \vee (\neg p)]$
1	1	1
$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$
0	0	0

Thus, we see that the last column of the table does not have all the entries equal to 1

A	0	0	0	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1	1	1
B	0	$\frac{1}{2}$	1	0	$\frac{1}{2}$	1	0	$\frac{1}{2}$	1
\wedge	0	$\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$	1
\vee	0	$\frac{1}{2}$	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1	$\frac{1}{2}$	1
\rightarrow	1	$\frac{1}{2}$	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$	1

A	0	0	0	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1	1	1
B	0	$\frac{1}{2}$	1	0	$\frac{1}{2}$	1	0	$\frac{1}{2}$	1
\wedge	0	0	0	0	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$	1
\vee	0	$\frac{1}{2}$	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	1	1	1

\rightarrow	1	1	1	0	1	1	0	$\frac{1}{2}$	1
---------------	---	---	---	---	---	---	---	---------------	---

One thing is common for Lukasiewicz and Bochvar logics, namely, the definition of $T(\neg p)$

$$T(\neg p) = 1 - T(p)$$

But, this is not true for Heyting's logic.

4.3. N-VALUED LOGICS FOR $N \geq 4$:

Once the 3-valued logics were accepted and their usefulness realized, further generalizations took place. Several n-valued logics for $n \geq 4$ were developed in the 1930's .

For a given value of n, consider the set $T(n)$ called the truth value set, where

$$T(n) = \left\{ 0, \frac{1}{(n-1)}, \frac{2}{(n-1)}, \dots, \frac{n-2}{(n-1)}, 1 \right\}$$

Using this set. Lukasiewicz proposed the first generalization of L, denoted by L_n using the following equations as definitions:

$$T(\neg p) = 1 - T(p)$$

$$T(p \vee q) = \max \{T(p), T(q)\}$$

$$T(p \wedge q) = \min \{T(p), T(q)\}$$

$$T(p \rightarrow q) = \min \{1, 1 - (p) + T(q)\}$$

$$T(p \leftrightarrow q) = 1 - |T(P) - T(q)|$$

Note that, for $n = 2$, $T(n) = (0, 1)$ and the above definitions reduce to the truth table of the classical logic. Similarly, for $n = 3$, $T(n) = (0, \frac{1}{2}, 1)$ and L_n reduces to L_3 .

Thus, L_n is an appropriate generalization of both the 2-valued classical logic L, and the 3-valued logic L_3

4.4. INFINITE-VALUED LOGICS

The natural generalization of n-valued logics is the infinite-valued logics, wherein infinite sets are used as truth value sets. Two of the commonly used infinite sets are:

1. $T(\text{infinity}) =$ all rational numbers in the unit interval $[0, 1]$
2. $T(1) =$ all real numbers in the unit interval $[0, 1]$

Of course, there are other infinite sets, which are subsets of $[0, 1]$ and which can be used as truth value sets. For example.

$$S_1 = \left\{ 0, 1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \dots, \frac{2}{3}, \frac{3}{4}, \frac{4}{5}, \dots \right\}$$

$$S_2 = \left\{ 0, 1, \frac{1}{2}, \left(\frac{1}{2}\right), \left(\frac{1}{2}\right), \dots, \left(\frac{1}{2}\right), 1 - \left(\frac{1}{2}\right), \dots \right\}$$

See that we can also express S_1 and S_2 in a compact way as follows:

$$S_1 = \left\{ 0, 1, \frac{1}{n}, 1 - \frac{1}{n}, n = 2, 3, \dots \right\}$$

$$S_2 = \left\{ 0, 1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \dots, \frac{2}{3}, \frac{3}{4}, \frac{4}{5}, \dots \right\}$$

Lukasiewicz proposed two infinite valued logics, denoted by $L(\infty)$ and $L(1)$. $L(\infty)$ is based on \mathbb{N} and $L(1)$ is based on \mathbb{Z} as their truth value sets respectively. Both of them are based on the same sets of definitions (Refer Eq. (1)). Note that $L(1)$ is a genuine generalization of $L(\infty)$ and $L(\infty)$ is a genuine generalization of $L(1)$.

4.5. FUZZY LOGICS:

In its widest sense, fuzzy logic encompasses the logics developed so far and even more. In fact, L_2 is a fuzzy logic in this sense. But for the present, it is enough to consider logic in the following narrower sense: fuzzy logic is any logic having $\mathbb{Z} = [0, 1]$ as its truth value set, we have already come across an example of fuzzy logic in the previous section, namely, $L(1)$. Most of the fuzzy logics (including $L(1)$) are based on the following definitions for the logical connectives \vee , \wedge and \neg

$$T(P \vee q) = \max [T(p), T(q)]$$

$$T(P \wedge q) = \min [T(p), T(q)]$$

$$T(\neg p) = 1 - T(p)$$

Where p, q are fuzzy propositions and $T(p), T(q)$ take values in \mathbb{Z} . Note that we have not specified the connectives \rightarrow each specification of \rightarrow gives rise to a different fuzzy logic. That is these fuzzy logics differ in the definition of \rightarrow . In $L(1)$, we have

$$T(P \rightarrow q) = \min [1, 1 - T(p), T(q)]$$

Zadeh proposed the following definition:

$$T(p \rightarrow q) = \max \{ \min [T(p), T(q)], 1 - T(p) \}$$

There are plenty of definition available in the literature for \rightarrow some of them are:

Let $a = T(p)$ and $b = T(q)$. then

1. $T(p \rightarrow q) = 1$, if $a \leq b$ and 0, otherwise
2. $T(P \rightarrow q) = 1$, if $a \leq b$ and b , otherwise
3. $T(P \rightarrow q) = \min \{1, b/a\}$
4. $T(p \rightarrow q) = \min \{1, [b(1-a)] / [a(1-b)]\}$

Each one of the above definitions give rise to different fuzzy logic.

Each one of the fuzzy logics is to be considered as a model for real life situation and the choice will depend on the characteristic of the problem considered and the intuition experience and the ingenuity of the problem solver

Given A in $PF(U)$, consider the proposition p where p : x is a member of A . Then

$$T(P) = A(x)$$

Similarly, if B is in $PF(U)$, then we get the proposition q where q : x is a member of B and $T(q) = B(x)$. Then

$$\begin{aligned} T(p \vee q) &= \max[T(p), T(q)] \\ &= \max[A(x), B(x)] \\ &= (A \cup B)(x) \end{aligned}$$

Thus, the proposition corresponding to $A \cup B$ is $p \vee q$. Similarly, we can show that $A \cap B$ corresponds to $p \wedge q$ and A' corresponds to $\neg p$.

CONCLUSION:

Fuzzy theories have been used in day-to-day life. Fuzzy set theory has been shown to be a useful tool to describe situations in which the data are imprecise or vague. Fuzzy sets handle such situations by attributing a degree to which a certain object belongs to a set. Fuzzy logic has been successfully used in numerous fields such as control systems engineering, image processing, power optimization. Fuzzy relation equations, which are obtained by the composition of binary fuzzy relations, are used in this work as a tool for evaluating student mathematical modelling skills. Fuzzy matrix frame work have been utilized in several different approaches to model the medicine diagnostic process and decision-making process.

REFERENCES

1. Zadeh, L.A. Fuzzy Sets, Info. and Control, 8, 1965.
2. Klir and Bo Yuvan, Fuzzy Sets: Theory and Applications, PHI< 1997.
3. Ross, T., Fuzzy Logic with Engineering Applications, McGraw Hill, 1995.
4. Kruse, R., Gebhardt J. and Klawonn, F., Foundations of Fuzzy Systems, John Wiley and Sons, 1994.
5. Bezdek. J. Spillman. B and Spillman. R. (1978). 'A fuzzy relation space for group decision theory', Fuzzy Sets Sys. 1: 255 - 268
6. Blin.J.M. (1974). 'Fuzzy relations in group decision theory.' J. Cyberntics.4 17-22.
7. Cho. H.H. (1993). 'Regular matrices in the semigroup of Hall matrices. Lin. Alg. Appl. 191: 151-163
8. Cho. H.H. (1993). 'Regular fuzzy matrices and fuzzy equations. 'Fuzzy Sets Sys, 105: 445 - 451.

INTRODUCTION TO R SOFTWARE

Project Report submitted to

ST. MARY'S COLLEGE (AUTONOMOUS), THOOTHUKUDI

Affiliated to

MANONMANIAM SUNDARANAR UNIVERSITY, TIRUNELVELI

In partial fulfillment of the requirement for the award of degree of

Bachelor of Science in Mathematics

Submitted by

NAME

REG.NO.

BLESSY JEBARANI. A

19AUMT07

EMIMA LUKRAGI. J

19AUMT11

MUTHU YOGESWARI. J

19AUMT30

STERLY. B

19AUMT46

VARSHA. S

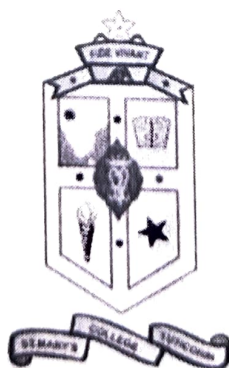
19AUMT47

Under the Guidance of

Dr. Tmt. A. PUNITHA THARANI, M.Sc., M.Phil., Ph.D.

Associate Professor of Mathematics and COE

St. Mary's College (Autonomous), Thoothukudi.



Department of Mathematics

St. Mary's College (Autonomous), Thoothukudi

(2021 - 2022)

CERTIFICATE

We here by declare that the project report entitled "INTRODUCTION TO R SOFTWARE" being submitted to **St. Mary's College (Autonomous), Thoothukudi** affiliated to **Manonmaniam Sundaranar University, Tirunelveli** in partial fulfillment for the award of degree of **Bachelor of Science in Mathematics** and it is a record of work done during the year 2021 - 2022 by the following students:

NAME	REG.NO.
BLESSY JEBARANI. A	19AUMT07
EMIMA LUKRAGI. J	19AUMT11
MUTHU YOGESWARI. J	19AUMT30
STERLY. B	19AUMT46
VARSHA. S	19AUMT47

Signature of the Guide

Dr. A. Punitha Tharani

M.Sc., M.Phil., Ph.D.

Associate Professor,

Dept. of Mathematics,

St. Mary's College (Autonomous),

Thoothukudi - 628 001.

Signature of the HOD

Dr. V.L. Stella Arputha Mary

M.Sc., M.Phil., B.Ed., Ph.D.

Head & Asst. Professor of Mathematics

St. Mary's College (Autonomous)

Thoothukudi-628 001.

Signature of the Examiner

Signature of the Principal

Principal

St. Mary's College (Autonomous)

Thoothukudi - 628 001.

DECLARATION

We hereby declare that the project reported entitled “**INTRODUCTION TO R SOFTWARE**”, is our original work. It has not been submitted to any university for any degree or diploma.

A. Blessy Jebarani
(BLESSY JEBARANI. A)

J. Emima Lukragi
(EMIMA LUKRAGI. J)

J. Muthu Yogeswari
(MUTHU YOGESWARI. J)

B. Sterly
(STERLY. B)

S. Varsha
(VARSHA. S)

ACKNOWLEDGEMENT

First of all, we thank Lord Almighty for showering his blessings to undergo this project.

With immense pleasure, we register our deep sense of gratitude to our guide **Dr. Tmt. A. PUNITHA THARANI, M.Sc., M.Phil., Ph.D.** for guiding and supporting us in every part of this project.

We are thankful to **Dr. V. L. Stella Arputha Mary M.Sc., M.Phil., B.Ed., Ph.D.,** Head of the Department, for having imparted necessary guidelines throughout the period of our studies.

We thank our beloved Principal, **Rev. Dr. Sr. A.S.J. Lucia Rose M.Sc., M.Phil., Ph.D., PGDCA** for providing us the help to carry out our project work successfully.

Finally, we thank all those who extended their helping hands regarding this project.

INTRODUCTION TO R SOFTWARE

CONTENT

1. INTRODUCTION	7
2. R AS A CALCULATOR	10
3. FUNCTIONS AND MATRIX OPERATIONS	14
4. MISSING DATA AND LOGICAL OPERATORS	19
5. CONDITIONAL EXECUTION AND LOOPS	22
6. DATA MANAGEMENT	25
7. STRINGS – DISPLAY AND FORMATTING	35
8. DATA FRAMES	42
9. STATISTICAL FUNCTIONS	49
10. PROGRAMMING IN R	62
11. APPLICATIONS	64
12. CONCLUSION	66
13. REFERENCES	67

1. INTRODUCTION

R is another emerging name in the Programming world!

R is an open-source programming language that is widely used as a statistical software and data analysis tool. R generally comes with the Command-line interface. R is available across widely used platforms like Windows, Linux, and macOS. Also, the R programming language is the latest cutting-edge tool.

It was designed by **Ross Ihaka and Robert Gentleman** at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team. R programming language is an implementation of the S programming language. It also combines with lexical scoping semantics inspired by Scheme. Moreover, the project conceives in 1992, with an initial version released in 1995 and a stable beta version in 2000.

Is it worth learning R in 2022?

In our opinion, absolutely **YES!** This is still an awesome programming language to learn. With the increasing demand for machine learning and data science, it is worth learning the R programming language. Various big tech companies like Facebook, Google, Uber, etc. are using the R language for their businesses. Learning the R programming language is surely worthwhile for future career endeavors.

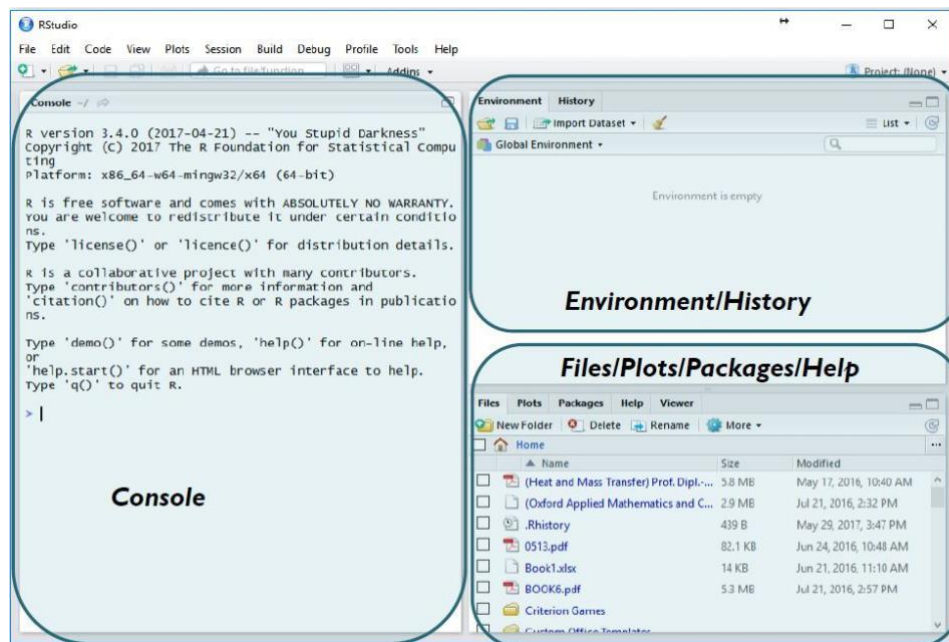
Programming in R

Since R is much similar to other widely used languages syntactically, it is easier to code and learn in R. Programs can be written in R in any of the widely used IDE like **R Studio, Rattle, Tinn-R**, etc. After writing the program save the file with the extension **.r**.

Introduction to R studio

R Studio is an integrated development environment (IDE) for R. IDE is a GUI, where you can write your codes, see the results and also see the variables that are generated during the course of programming.

R Studio can be downloaded from its Official Website (<https://rstudio.com/>)



After the installation process is over, the R Studio interface looks like this:

- The console panel (left panel) is the place where R is waiting for you to tell it what to do, and see the results that are generated when you type in the commands.
- To the top right, you have the Environmental/History panel. It contains 2 tabs:
 - **Environment tab:** It shows the variables that are generated during the course of programming in a workspace that is temporary.
 - **History tab:** In this tab, you'll see all the commands that are used till now from the start of usage of R Studio.
- To the right bottom, you have another panel, which contains multiple tabs, such as files, plots, packages, help, and viewer.
 - The **Files tab** shows the files and directories that are available within the default workspace of R.
 - The **Plots tab** shows the plots that are generated during the course of programming.
 - The **Packages tab** helps you to look at what are the packages that are already installed in the R Studio and it also gives a user interface to install new packages.
 - The **Help tab** is the most important one where you can get help from the R Documentation on the functions that are in built-in R.
 - The final and last tab is that the **Viewer tab** which can be used to see the local web content that's generated using R.

Getting help with R:

Before asking others for help, it's generally a good idea for you to try to help yourself. R includes extensive facilities for accessing documentation and searching for help. There are also specialized search engines for accessing information about R on the internet, and general internet search engines can also prove useful.

- If you need help with a function, then type question mark followed by the name of the function. For example, `?read.table` to get help for function `read.table`.
- Sometimes, you want to search by the subject on which we want help (e.g., data input). In such a case, type `help.search("data input")`
- `'help()'` for on-line help, or `'help.start()'` for an HTML browser interface to help.
- The `find` function tells us what package something is in.

For example:

```
> find("lowess")
```

```
[1] "package:stats"
```

- The `apropos` returns a character vector giving the names of all objects in the search list that match your enquiry.

For example:

```
> apropos("lm")
```

```
[1] ".colMeans"      ".lm.fit"         "colMeans"        "confint.lm"
```

```
[5] "contr.helmert"  "dummy.coef.lm"   "glm"             "glm.control"
```

```
[9] "glm.fit"        "KalmanForecast"  "KalmanLike"      "KalmanRun"
```

- To see a worked example just type the function name, e.g., `lm` for linear models:

```
> example(lm)
```

and we see the printed and graphical output produced by the `lm` function.

Libraries in R:

R provides many functions and one can also write own. Functions and datasets are organised into libraries. To use a library, simply type the library function with the name of the library in brackets. For example: `>library(MASS)`

Examples of libraries that come as a part of base package in R:

- **MASS:** package associated with Venables and Ripley's book entitled Modern Applied Statistics using S-Plus.
- **mgcv** : generalized additive models.

2. R AS A CALCULATOR

R can be used as a powerful calculator by entering equations directly at the prompt in the command console. R will evaluate the expressions and respond with the result. While this is a simple interaction interface, there could be problems if you are not careful. R will normally execute your arithmetic expression by evaluating each item from left to right, but generally it follows the BEDMAS order: **Brackets ()**, **Exponents ^**, **Division / and Multiplication ***, **Addition + and Subtraction -**. Let's start with some simple expressions as examples.

Simple Arithmetic Expressions:

The operators R uses for basic arithmetic are:

- + Addition
- Subtraction
- * Multiplication
- / Division
- ^ Exponentiation

Examples:

```
> 2+3
[1] 5
> 2-3
[1] -1
> 2*3
[1] 6
> 3/2
[1] 1.5
> 2^3
[1] 8
> 2*3-4+5/6
[1] 2.8333
```

Integer Division

Division in which the fractional part(remainder) is discarded

Usage:

%%

Example:

```
> c(2,3,5,7) %% 2  
[1] 1 1 2 3
```

Modulo Division

$x \bmod y$: Modulo operation finds the remainder after division of one number by another.

Usage:

%%

Example:

```
> c(2,3,5,7) %% 2  
[1] 0 1 1 1
```

Maximum & Minimum

The `max()`, `min()` is a built in R- function. `max()` is used to calculate the maximum of vector elements or maximum of a particular column of a data frame. `min()`, is used to calculate the minimum of vector elements or minimum of a particular column of a data frame.

Usage:

- `max(x)`
- `min(x)`, where `x` is a numeric or character arguments.

Example:

```
> max(1.2, 3.4, -7.8)  
[1] 3.4  
> min(1.2, 3.4, -7.8)  
[1] -7.8
```

Absolute Value

To calculate the absolute value in R, use the `abs()` method. The `abs()` function takes a real number or numeric value as a vector, matrix, or data frame and returns the absolute value.

Usage:

abs(x), where x is a numeric or character arguments.

Example:

```
> abs(c(-1,-2,-3,4,5))
```

```
[1] 1 2 3 4 5
```

Square Root

sqrt() function in R Language is used to calculate the mathematical square-root of the value passed to it as argument.

Usage:

sqrt(x), where x is a numeric or character arguments.

Example:

```
> sqrt(c(4,9,16,25))
```

```
[1] 2 3 4 5
```

Sum

sum() function in R is used to calculate the sum of vector elements.

Usage:

sum(x), where x is a numeric or character arguments.

Example:

```
> sum(c(2,3,5,7))
```

```
[1] 17
```

Product

prod() function in R Language is used to return the multiplication results of all the values present in its arguments.

Usage:

prod(x), where x is a numeric or character arguments.

Example:

```
> prod(c(2,3,5,7))
```

```
[1] 210
```

Round

round() function in R Language is used to round off values to a specific number of decimal values.

Usage:

round(x), where x is a numeric or character arguments.

Example:

```
> round(1.23)
```

```
[1] 1
```

round(), floor(), ceiling()	Rounding, up and down
log()	Logarithms
exp()	Exponential function
sin(), cos(), tan(),	Trigonometric functions
sinh(), cosh(), tanh(),	Hyperbolic functions

Assignments

Assignment operator “=” can be used to assign the value to a variable in an environment.

Example:

```
> x1 = c(1,2,3,4)
```

```
> x2 = x1^2
```

```
> x2
```

```
[1] 1 4 9 16
```


3. FUNCTIONS & MATRIX OPERATIONS

Functions

A function is a set of statements organized together to perform a specific task. R has a large number of in-built functions and the user can create their own functions. In R, a function is an object so the R interpreter is able to pass control to the function, along with arguments that may be necessary for the function to accomplish the actions. The function in turn performs its task and returns control to the interpreter as well as any result which may be stored in other objects.

Usage:

```
Name = function(Argument1, Argument2, ...)
```

```
{  
expression  
}
```

where expression is a single command or a group of commands

Function (Single Variable)

```
> abc = function(x){  
x^2  
}  
> abc(3)  
[1] 9
```

Function (Two Variables)

```
> abc = function(x,y){  
x^2+y^2  
}  
> abc(-2,-1)  
[1] 5
```

Function (Other Variables)

```
> abc = function(x){  
sin(x)^2+cos(x)^2 + x
```

```
}
```

```
> abc(8)
```

```
[1] 9
```

Matrix

In R, a matrix is a collection of elements of the same data type (numeric, character, or logical) arranged into a fixed number of rows and columns. It is a rectangular array with p rows and n columns. An element in the i -th row and j -th column is denoted by X_{ij} or $X[i, j]$. A Matrix is created using the `matrix()` function.

In R, a 4×2 -matrix X can be created with a following command:

```
> x = matrix(nrow=4, ncol=2, data=c(1,2,3,4,5,6,7,8))
```

```
> x
```

```
  [,1] [,2]
```

```
[1,]  1   5
```

```
[2,]  2   6
```

```
[3,]  3   7
```

```
[4,]  4   8
```

Note:

- The parameter `nrow` defines the row number of a matrix.
- The parameter `ncol` defines the column number of a matrix.
- The parameter `data` assigns specified values to the matrix elements.
- The values from the parameters are written column-wise in matrix.
- One can access a single element of a matrix with `x[i,j]`:

```
> x[3,2]
```

```
[1] 7
```

- In case, the data has to be entered row wise, then a 4×2 -matrix X can be created with

```
> x = matrix( nrow=4, ncol=2, data=c(1,2,3,4,5,6,7,8), byrow = TRUE)
```

```
> x
```

```
  [,1] [,2]
```

```
[1,]  1   2
```

```
[2,]  3   4
```

```
[3,]  5   6
```

```
[4,] 7 8
```

Properties of a Matrix

`dim()` function in R Language is used to get the dimension of the specified matrix, array or data frame. `nrow()` function is used to return the number of rows of the specified matrix. `ncol()` function is used to return the number of columns of the specified matrix. `mode()` function informs the type of an object in the matrix.

Usage:

- `dim(x)`, where x is an R object, for example a matrix, array or data frame.
- `nrow(x)`, where x is a vector, array, data frame.
- `ncol(x)`, where x is a vector, array, data frame.
- `mode(x)`, where x is any R object.

Example:

```
> dim(x)
[1] 4 2
> nrow(x)
[1] 4
> ncol(x)
[1] 2
> mode(x)
[1] "numeric"
```

Assigning a specified number to all matrix elements:

```
> x = matrix(nrow=4, ncol=2, data=2)
> x
[,1] [,2]
[1,] 2 2
[2,] 2 2
[3,] 2 2
[4,] 2 2
```

Diagonal Matrix

diag() function in R Language is used to construct a diagonal matrix. t() function in R Language is used to calculate transpose of a matrix or Data Frame.

Example:

```
> d = diag(1, nrow=2, ncol=2)
> d
[,1] [,2]
[1,] 1 0
[2,] 0 1
```

Transpose of a Matrix

t() function in R Language is used to calculate transpose of a matrix or Data Frame.

Example:

```
> x = matrix(nrow=4, ncol=2, data=1:8, byrow=T)
> x
      [,1] [,2]
[1,]  1    2
[2,]  3    4
[3,]  5    6
[4,]  7    8
> xt = t(x)
> xt
      [,1] [,2] [,3] [,4]
[1,]  1    3    5    7
[2,]  2    4    6    8
```

Matrix Operations

There are multiple matrix operations that you can perform in R. The most basic matrix operations are addition and subtraction. Addition and subtraction of matrices of same dimensions can be executed with the usual operators + and -.

Matrix Multiplication

In R, the operator `%*%` is used for matrix multiplication satisfying the condition that the number of columns in the first matrix is equal to the number of rows in second.

Example:

```
> xtx = t(x) %*% x
> xtx
[1,] [,2]
[1,] 84 100
[2,] 100 120
```

Inverse of a Matrix

In order to calculate the inverse of a matrix in R, `solve()` function is used. It finds the inverse of a positive definite matrix.

Example:

```
> y=matrix( nrow=2, ncol=2, byrow=T, data=c(84,100,100,120))
> y
[1,] [,2]
[1,] 84 100
[2,] 100 120
> solve(y)
[1,] [,2]
[1,] 1.50 -1.25
[2,] -1.25 1.05
```

4. MISSING DATA & LOGICAL OPERATORS

Missing Data

R represents missing observations through the data value NA. We can detect missing values using is.na. NA is a placeholder for something that exists but is missing. NULL stands for something that never existed at all.

Example:

```
> x = NA
> is.na(x)
[1] TRUE
> x = c(11, NA, 13)
> is.na(x)
[1] FALSE TRUE FALSE
```

Logical Operators and Comparisons

The following table shows the operations and functions for logical comparisons (True or False). TRUE and FALSE are reserved words denoting logical constants.

Operator	Executions
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal
==	Exactly equal to
!=	Not equal to
!	Negation(not)
&, &&	and
,	or
Xor()	either...or(exclusive)
isTRUE(x)	Test if x is TRUE
TRUE	true
FALSE	false

- The shorter form performs element-wise comparisons in almost the same way as arithmetic operators.

- The longer form evaluates left to right examining only the first element of each vector. Evaluation proceeds only until the result is determined.
- The longer form is appropriate for programming control-flow and typically preferred in if clauses (conditional).

Examples:

```
> 8 > 7
```

```
[1] TRUE
```

- Is 8 less than 6?

```
> isTRUE(8<6)
```

```
[1] FALSE
```

```
> x = 5
```

```
> (x < 10) && (x > 2)
```

```
[1] TRUE
```

- Is x greater than 10 or x is greater than 5?

```
> (x > 10) || (x > 5)
```

```
[1] FALSE
```

- Is x equal to 10 and is y equal to 20?

```
> x = 10
```

```
> y = 20
```

```
> (x == 10) & (y == 20)
```

```
[1] TRUE
```

Examples using & and |

```
> x = 1:6
```

```
> (x > 2) | (x < 5)
```

```
[1] TRUE TRUE TRUE TRUE TRUE TRUE
```

```
[1] 1 2 3 4 5 6
```

```
> x = 1:6
```

```
> (x > 2) & (x < 5)
```

```
[1] FALSE FALSE TRUE TRUE FALSE FALSE
```

```
> x[(x > 2) & (x < 5)]
```

```
[1] 3 4
```

The longer form evaluates left to right examining only the first element of each vector

```
> x = 1:6
```

```
> (x > 2) && (x < 5)
```

```
[1] FALSE
```

is equivalent to:

```
> (x[1] > 2) & (x[1] < 5)
```

```
[1] FALSE
```

Note: `x[1]` is only the first element in `x`

Truth Table

A truth table is a display of the inputs to, and the output of a Boolean function organized as a table where each row gives one combination of input values and the corresponding value of the function.

Statement 1 :: (x)	Statement 2 :: (y)	Outcomes :: x and y	Outcomes :: x or y
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

5. CONDITIONAL EXECUTION & LOOPS

Control structures in R:

- Control statements,
- Loops,
- Functions.

Conditional Execution

Conditional execution controls whether or not the core will execute an instruction. If they match then the instruction is executed, otherwise the instruction is ignored. The condition attribute is postfixed to the instruction mnemonic, which is encoded into the instruction. Conditionals are expressions that perform different computations or actions depending on whether a predefined Boolean condition is TRUE or FALSE. Conditional statements include `if()`, the combination `if()/else()`, and `ifelse()`.

Usage:

`if (condition) {executes commands if condition is TRUE}`

`if (condition) {executes commands if condition is TRUE} else { executes commands if condition is FALSE }`

Example:

```
> x = 5
```

```
> if ( x==3 ) { x = x-1 } else { x = 2*x }
```

```
> x [1] 1
```

Interpretation:

- If $x = 3$, then execute $x = x - 1$.
- If $x \neq 3$, then execute $x = 2 * x$.

In this case, $x = 5$, so $x \neq 3$. Thus $x = 2 * 5$

ifelse Execution

The `ifelse` function is used to assign one object or another depending on whether the first argument, `test`, is TRUE or FALSE.

Usage: `ifelse(test, yes, no)`

Example:

```
> x = 1:10
```

```
>x
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
> ifelse( x<6, x^2, x+1 )
```

```
[1] 1 4 9 16 25 7 8 9 10 11
```

Interpretation:

- If $x < 6$ (TRUE), then $x = x^2$ (YES).
- If $x \geq 6$ (FALSE), then $x = x + 1$ (NO).
- So, for $x = 1, 2, 3, 4, 5$, we get $x = x^2 = 1, 4, 9, 16, 25$
- For $x = 6, 7, 8, 9, 10$, we get $x = x + 1 = 7, 8, 9, 10, 11$

Loops

Repetitive commands are executed by loops

- for loop
- while loop
- repeat loop

For Loop

For loop in R Programming Language can be used to execute a group of statements repeatedly depending upon the number of elements in the object. It is an entry-controlled loop. In this loop the test condition is tested first, then the body of the loop is executed, the loop body would not be executed if the test condition is false.

Usage:

```
for (var in vector) {commands to be executed}
```

Here, var takes on each value of vector during the loop.

Example:

```
> for ( i in c(2,4,6,7) ) { print( i^2 ) }
```

```
[1] 4
```

```
[1] 16
```

```
[1] 36
```

```
[1] 49
```

While Loop

A "While" Loop is used to repeat a specific block of code an unknown number of times, until a condition is met.

Usage: while(condition){ commands to be executed as long as condition is TRUE }

Example:

```
> i = 1
> while (i<5) {
+ print(i^2)
+ i = i+2
+}
[1] 1
[1] 9
```

Repeat Loop

Repeat loop in R is used to iterate over a block of code multiple number of times. And also, it executes the same code again and again until a break statement is found. Additionally, the command next is available, to return to the beginning of the loop (to return to the first command in the loop).

Usage: repeat{ commands to be executed }

Example:

```
> i = 1
> repeat{
+ i = i+1
+ if (i < 10) next
+ print(i^2)
+ if (i >= 13) break
+}
[1] 100
[1] 121
[1] 144
[1] 169
```

6. DATA MANAGEMENT

Sequences

seq() function in R Language is used to create a sequence of elements in a Vector. It takes the length and difference between values as optional argument.

Usage:

```
seq(from,to,by)
```

Example:

```
> seq(from=-4, to=4)
```

```
[1] -4 -3 -2 -1 0 1 2 3 4
```

Sequence with constant increment:

Example:

```
> seq(from=20, to=10, by=-2)
```

```
[1] 20 18 16 14 12 10
```

Downstream sequence with constant increment:

Example:

```
> seq(from=3, to=-2, by=-0.5)
```

```
[1] 3.0 2.5 2.0 1.5 1.0 0.5 0.0 -0.5 -1.0 -1.5 -2
```

Sequences with a predefined length:

Sequences with a predefined length with default increment +1.

Examples:

```
> seq(to=10, length=10)
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
> x=50
```

```
> seq(0, x, x/10)
```

```
[1] 0 5 10 15 20 25 30 35 40 45 50
```

Index-Vector

Vector elements are accessed using indexing vectors, which can be numeric, character or logical vectors. You can access an individual element of a vector by its position (or "index"), indicated using square brackets.

Example:

```
> x = c(9,8,7,6)
> ind = seq(along=x)
> ind
[1] 1 2 3 4
> x[ ind[2] ]
[1] 8
```

Generating Sequence of Alphabets

letters are used to find sequence of lowercase alphabets. To create a sequential uppercase alphabet in R, use the LETTERS constant. The LETTERS is a character constant in R that generates an uppercase alphabet, and you can use it with different functions to extract the result as per your requirement.

Usage:

- letters[from_index:to_index]
- LETTERS[from_index:to_index]

Examples:

```
> letters
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n"
[15] "o" "p" "q" "r" "s" "t" "u" "v" "w" "x" "y" "z"
> letters[1:3]
[1] "a" "b" "c"
> LETTERS
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N"
[15] "O" "P" "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
> LETTERS[21:23]
[1] "U" "V" "W"
```

Repeats

Command `rep()` is used to replicate the values in a vector.

Usage:

- `rep(x, times=n)`
- `rep(x, each=n)`

Example:

```
> rep(1:4, each = 2, times = 3)
```

```
[1] 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4
```

Repetition of elements in a matrix:

```
> x = matrix(nrow=2, ncol=2, data=1:4, byrow=T)
```

```
> x
```

```
  [,1] [,2]
```

```
[1,] 1    2
```

```
[2,] 3    4
```

```
> rep(x, 3)
```

```
[1] 1 3 2 4 1 3 2 4 1 3 2 4
```

Repetition of characters:

```
> rep(c("a", "b", "c"), 2)
```

```
[1] "a" "b" "c" "a" "b" "c"
```

Sorting

To sort a data frame in R, use the `order()` function. By default, sorting is ASCENDING. Prepend the sorting variable by a minus sign to indicate DESCENDING order.

Usage:

`sort(x, decreasing = FALSE, ...)`, where `x` is a sequence of numeric, complex, character or logical vectors.

Example:

```
> y = c(8,5,7,6)
```

```
> y
```

```
[1] 8 5 7 6
> sort(y)
[1] 5 6 7 8
> sort(y, decreasing = TRUE)
[1] 8 7 6 5
```

Ordering

The `order()` function in R is very useful in sorting a particular data value according to a specific variable. It will arrange the data or orders the data based on given parameters.

Usage:

`order(x, decreasing = FALSE, ...)`, where `x` is a sequence of numeric, complex, character or logical vectors.

Example:

```
> y = c(8,5,7,6)
> y
[1] 8 5 7 6
> order(y)
[1] 2 4 3 1
> order(y, decreasing = TRUE)
[1] 1 3 4 2
```

Lists

Lists are the R objects which contain elements of different types like – numbers, strings, vectors and another list inside it. A list can also contain a matrix or a function as its elements. List is created using `list()` function. Lists can be indexed by position.

For example: `x[[5]]` refers to the fifth element of `x`.

Difference between a vector and a list:

- In a vector, all elements must have the same mode.
- In a list, the elements can have different modes.

List can contain any kind. An example of a list that contains different object types:

```
> z1 = list(c("water", "juice", "lemonade"), rep(1:4, each=2), matrix(data=5:8, nrow=2, ncol=2, byrow=T))
```

```
> z1
```

```
[[1]]
```

```
[1] "water" "juice" "lemonade"
```

```
[[2]]
```

```
[1] 1 1 2 2 3 3 4 4
```

```
[[3]]
```

```
[,1] [,2]
```

```
[1,] 5 6
```

```
[2,] 7 8
```

Access the elements of a list using the operator [[]]

Following commands work.

```
> z1[[1]]
```

```
[1] "water" "juice" "lemonade"
```

Suppose we want to extract "juice".

```
z1[[1]][2]
```

```
[1] "juice"
```

Mode

Every object has a mode.

The mode indicates how the object is stored in memory: as a

- number,
- character string,
- list of pointers to other objects,
- function etc.

OBJECT	EXAMPLE	MODE
Number	1.234	numeric
Vector of numbers	c(5, 6, 7, 8)	numeric
Character string	"India"	character
Vector of character strings	c("India", "USA")	character

Factor	factor(c("UP", "MP"))	numeric
List	list("India", "USA")	list
Data frame	data.frame(x=1:2, y=c("India", "USA"))	list
Function	print	function

Usage:

mode(x), where x is a numeric or character arguments.

Example:

```
> mode(c(5,6,7,8))
[1] "numeric"
> mode(c("India", "USA"))
[1] "character"
> mode(list("India", "USA"))
[1] "list"
> mode(print)
[1] "function"
```

Vector Indexing

Vector elements are accessed using indexing vectors, which can be numeric, character or logical vectors. You can access an individual element of a vector by its position (or "index"), indicated using square brackets.

Example:

```
> x = 1:10
>x
[1] 1 2 3 4 5 6 7 8 9 10
> x[ (x > 5) ]
[1] 6 7 8 9 10
> x[ (x%%2==0) ]
[1] 2 4 6 8 10
```

Logical Vector

A logical vector is a vector that only contains TRUE and FALSE values. In R, true values are designated with TRUE, and false values with FALSE. When you index a vector with a logical vector, R will return values of the vector for which the indexing vector is TRUE.

Example:

```
> x[5] = NA
> x
[1] 1 2 3 4 NA 6 7 8 9 10
> y = x[ !is.na(x) ]
> y
[1] 1 2 3 4 6 7 8 9 10
```

Vector of Negative integers

A negative of a vector represents the direction opposite to the reference direction. It means that the magnitude of two vectors is same but they are opposite in direction.

```
> x = 1:10
> x
[1] 1 2 3 4 5 6 7 8 9 10
> x[-(1:5)]
[1] 6 7 8 9 10
```

String Vector

names() function in R Language is used to get or set the name of an Object. This function takes object i.e., vector, matrix or data frame as argument along with the value that is to be assigned as name to the object. The length of the value vector passed must be exactly equal to the length of the object to be named.

Usage:

names(x), where x is an R object.

Example:

```
> z = list(a1 = 1, a2 = "c", a3 = 1:3)
> z
```

```

$a1
[1] 1
$a2
[1] "c"
$a3
[1] 1 2 3
> names(z)
[1] "a1" "a2" "a3"

```

Empty Index

In R Programming Language an empty index can be created by simply not passing any value while creating a regular index using the `x[]` function.

Example:

```

> x = 1:10
>x
[1] 1 2 3 4 5 6 7 8 9 10
> x[]
[1] 1 2 3 4 5 6 7 8 9 10

```

Matrix created from Lists

List can be heterogeneous (mixed modes). We can start with a heterogeneous list, give it dimensions, and thus create a heterogeneous matrix that is a mixture of numeric and character data.

Example:

```

> ab = list(1, 2, 3, "X", "Y", "Z")
> dim(ab) = c(2,3)
> print(ab)
      [,1] [,2] [,3]
[1,]  1   3  "Y"
[2,]  2  "X"  "Z"

```

Factors

Factors in R Programming Language are data structures that are implemented to categorize the data or represent categorical data and store it on multiple levels. They can be stored as integers with a corresponding label to every unique integer. Factors are the data objects which are used to categorize the data and store it as levels. They can store both strings and integers. They are useful in data analysis for statistical modeling.

Usage:

- `factor(x)`
- `factor(x,levels)`, where x is a numeric or character arguments.

Example:

```
> x = factor(c("juice", "juice", "lemonade",  
"juice", "water"))
```

```
>x
```

```
[1] juice juice lemonade juice water
```

```
Levels: juice lemonade water
```

The single levels are ordered alphabetically:

```
juice --- lemonade --- water
```

Unclass Function

All objects in R have a *class*, reported by the function `class`. For simple vectors this is just the mode, for example "numeric", "logical", "character" or "list", but "matrix", "array", "factor" and "data.frame" are other possible values. `unclass()` is used to temporarily remove the effects of class. The command `unclass` shows, an integer is assigned to every factor level.

Usage:

`unclass(x)`, where x is an R object.

Example:

```
> x = factor(c("juice", "juice", "lemonade", "juice", "water"))
```

```
> unclass(x)
```

```
[1] 1 1 2 1 3
```

```
attr("levels")
```

```
[1] "juice" "lemonade" "water"
```

Ordered Factor

The levels of factors are stored in alphabetical order, or in the order they were specified to factor if they were specified explicitly. Sometimes the levels will have a natural ordering that we want to record and want our statistical analysis to make use of. The `ordered()` function creates such ordered factors but is otherwise identical to `factor`.

Example:

```
> income = ordered(c("high", "high", "low", "medium", "medium"), levels=c("low", "medium", "high"))
```

```
> income
```

```
[1] high high low medium medium
```

```
Levels: low < medium < high
```

Turning a vector into a factor:

A vector can be turned into a factor with the command `as.factor()`. However, it converts a vector into a factor and uses value labels as factor levels.

Usage:

`as.factor(x)`, where `x` is a vector of data, taking a small number of distinct values.

Example:

```
> x = c(4, 5, 1, 2, 3, 3, 4, 4, 5, 6)
```

```
> x = as.factor(x)
```

```
> x
```

```
[1] 4 5 1 2 3 3 4 4 5 6
```

```
Levels: 1 2 3 4 5 6
```

7. STRINGS - DISPLAY & FORMATTING

Any value written within a pair of single quote or double quotes in R is treated as a string. Internally R stores every string within double quotes, even when you create them with single quote.

Formatting and Display of strings

A common task when working with character strings involves printing and displaying them on the screen or on a file. R provides a series of functions for printing strings. Some of the printing functions are

- `print(x)`, where `x` is an object used to select a method.
- `format(x)`, where `x` is any R object, typically numeric.
- `cat(x)`, where `x` is an R object
- `paste(x)`, where `x` is a one or more R objects, to be converted to character vectors.

Print Function

In R there are various methods to print the output. Most common method to print output in R program, is the `print()` function.

Example:

```
> print(sqrt(2), digits=16)
[1] 1.414213562373095
```

Limitations:

- The print function has a significant limitation that it prints only one object at a time. Trying to print multiple items gives error message:

Example:

```
> print("The zero occurs at", 2*pi, "radians.")
Error in print.default("The zero occurs at", 2 * pi, "radians."): invalid 'quote' argument
```

- The only way to print multiple items is to print them one at a time

Example:

```
> print("The zero occurs at"); print(2*pi);
print("radians")
[1] "The zero occurs at"
[1] 6.283185
[1] "radians"
```

Format Function

The function `format()` allows you to format an R object for pretty printing. Essentially, `format()` treats the elements of a vector as character strings using a common format. This is especially useful when printing numbers and quantities under different formats.

Usage:

```
format(x, trim = FALSE, digits = NULL, nsmall = 0L, justify = c("left", "right", "centre", "none"),
width = NULL, ...)
```

Example:

```
> print(format( 0.5, digits=10, nsmall=15 ))
[1] "0.5000000000000000"
```

Cat Function

The function `cat()` converts its arguments to character strings, concatenates them, separating them by the given `sep=` string, and then prints them. `cat` puts a space between each item by default. One must provide a newline character (`\n`) (newline) to terminate the line. `cat` is useful for producing output in user defined functions.

Example:

```
> cat( 1:10, sep = "_" )
1_2_3_4_5_6_7_8_9_10
```

The `cat` function is an alternative to `print` that lets you combine multiple items into a continuous output as well as it can also print simple vectors.

```
> cat("The zero occurs at", 2*pi, "radians.", "\n")
```

The zero occurs at 6.283185 radians.

```
> evenno = c(2,4,6,8,10)
```

```
> evenno
```

```
[1] 2 4 6 8 10
```

```
> cat("The first few even numbers are:", evenno, "... \n")
```

The first few even numbers are: 2 4 6 8 10 ...

Paste Function

The `paste()` function concatenates several strings together. It creates a new string by joining the given strings end to end. The result of `paste()` can be assigned to a variable. `paste` inserts a single space between pairs of strings. A desired line break can be achieved with `"\n"`

(newline). The collapse parameter defines a top-level separator and instructs paste to concatenate the generated strings using that separator:

Usage:

```
paste(..., sep = " ", collapse = NULL).
```

Example:

```
> x = paste("Ex", 1:5, sep="_", collapse="")
```

```
> x[1]
```

```
[1] "Ex_1Ex_2Ex_3Ex_4Ex_5"
```

```
> names = c("Prof. Singh", "Mr. Venkat", "Dr. Jha")
```

```
> paste(names, "is", "a good", "person.", collapse=" and ")
```

```
[1] "Prof. Singh is a good person., and Mr. Venkat is a good person., and Dr. Jha is a good person."
```

Splitting

The strsplit() in R programming language function is used to split the elements of the specified character vector into substrings according to the given substring taken as its parameter.

Usage:

```
strsplit(x, split, fixed = FALSE, ...)
```

Example:

```
> x = "The&!syntax&!of&!paste&!is!&available!&inthe online-help"
```

```
> x
```

```
[1] "The&!syntax&!of&!paste&!is!&available!
```

```
&inthe online-help"
```

```
> abc = strsplit(x,"!&")
```

```
> abc
```

```
[[1]]
```

```
[1] "The&!syntax&!of&!paste&!is" "available!&inthe online-help"
```

Note: To access single components:

```
> abc[[1]][1]
```

```
[1] "The&!syntax&!of&!paste&!is"
```


String Manipulation Functions

Here are the functions available for string manipulation in R:

- `nchar(x)`
- `tolower(x)`
- `toupper(x)`, where `x` is a character vector, or a vector to be coerced to a character vector

nchar Function

With the help of this `nchar()` function, we can count the characters. This function consists of a character vector as its argument which then returns a vector comprising of different sizes of the elements of `x`. `nchar()` is the fastest way to find out if elements of a character vector are non-empty strings or not.

Example:

```
> x = "R course 24.07.2017"
```

```
> nchar(x)
```

```
[1] 19
```

tolower and toupper Functions

The `tolower()` function is used to convert the string characters to the lower case. The `toupper()` function is used to convert the string characters to upper case.

Usage:

- `tolower(x)`
- `toupper(x)`, where `x` is a character vector, or a vector to be coerced to a character vector

Example:

```
> x = "R course will start from 24.07.2022"
```

```
> tolower(x)
```

```
[1] "r course will start from 24.07.2022"
```

```
> toupper(x)
```

```
[1] "R COURSE WILL START FROM 24.07.2022"
```

Operations with Strings

R has various functions for regular expression-based match and replaces. Some functions (e.g., `grep`, `grepl`, etc.) are used for searching for matches and functions whereas `sub` and `gsub` are used for performing replacement.

- `sub()`
- `gsub()`
- `grep()`

sub and gsub Functions

The `sub()` and `gsub()` function in R is used for substitution as well as replacement operations. The `sub()` function will replace the first occurrence leaving the other as it is. On the other hand, the `gsub()` function will replace all the strings or values with the input strings.

Usage:

```
sub(old, new, string)
```

```
gsub(old, new, string)
```

Example:

```
> y = "Mr. Singh is the smart one. Mr. Singh is funny, too."
> y
[1] "Mr. Singh is the smart one. Mr. Singh is funny, too."
> sub("Mr. Singh", "Professor Jha", y)
[1] "Professor Jha is the smart one. Mr. Singh is funny, too."
> gsub("Mr. Singh", "Professor Jha", y)
[1] "Professor Jha is the smart one. Professor Jha is funny, too."
```

grep Function

`grep()` function in R Language is used to search for matches of a pattern within each element of the given string. If its value is `TRUE`, it returns the matching elements vector, else return the indices vector. `value = FALSE` is default.

Usage:

```
grep(pattern, x, value=TRUE/FALSE)
```

Examples:

`grep(pattern, x, value = TRUE)` returns a character vector containing the selected elements of `x`.

```
> x = c("R Course", "exercises", "include examples of R language")
```

```
> grep("ex", x, value=T)
```

```
[1] "exercises" "include examples of R language"
```

grep(pattern, x, value = FALSE) returns an integer vector of the indices of the elements of x that yielded a match.

```
> x = c("R Course", "exercises", "include examples of R language")
```

```
> grep("ex", x, value=F)
```

```
[1] 2 3
```

Combining two Strings

The c() in R programming language function is used to combine two strings.

Example:

```
> x = "R course 24.07.2022"
```

```
> y = "Number of participants: 25"
```

```
> c(x,y)
```

```
[1] "R course 24.07.2022" "Number of participants: 25"
```

eval Function

eval() function in R Language is used to evaluate an expression passed to it as argument.

Usage:

eval(x), where x is an object to be evaluated

Example:

```
> eval("6+8")
```

```
[1] "6+8"
```

```
> eval(6+8)
```

```
[1] 14
```

```
> eval("6+8 is Fourteen")
```

```
[1] "6+8 is Fourteen"
```

The `eval()` function evaluates an expression, but `"6+8"` is a string, not an expression whereas `6+8` is not an expression so it evaluates.

parse Function

`parse()` function in R language is used to convert an object of character class to an object of expression class. `parse()` with `text=string` is used to change the string into an expression.

Usage:

`parse(x)`, where `x` is an object of character class.

Example:

```
> eval("6+8")
[1] "6+8"
> class("6+8")
[1] "character"
> eval(parse(text="6+8"))
[1] 14
> class(parse(text="6+8"))
[1] "expression"
```

8. DATA FRAMES

A data frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column. Following are the characteristics of a data frame. The column names should be non-empty. The row names should be unique. Data frames contain complete data sets that are mostly created with other programs (spreadsheet-files, software SPSS-files, Excel-files etc.). Variables in a data frame may be numeric (numbers) or categorical (characters or factors).

Example:

An example data frame painters is available in the library MASS.

```
> library(MASS)
```

```
> painters
```

	Composition	Drawing	Colour	Expression	School
Da Udine	10	8	16	3	A
Da Vinci	15	16	4	14	A
Del Piombo	8	13	16	7	A
Del Sarto	12	16	9	8	A
Fr. Penni	0	15	8	0	A

Here, the names of the painters serve as row identifications, i.e., every row is assigned to the name of the corresponding painter.

Row Names

All data frames have a row names attribute, a character vector of length the number of rows with no duplicates nor missing values.

Usage:

rownames(x), where x is an object of class "data.frame".

Example:

```
> rownames(painters)
```

```
[1] "Da Udine"      "Da Vinci"      "Del Piombo"
```

[4] "Del Sarto"	"Fr. Penni"	"Guilio Romano"
[7] "Michelangelo"	"Perino del Vaga"	"Perugino"
[10] "Raphael"	"F. Zucarro"	"Fr. Salviata"
[13] "Parmigiano"	"Primaticcio"	"T. Zucarro"
[16] "Volterra"	"Barocci"	"Cortona"
[19] "Josepin"	"L. Jordaens"	"Testa"
[22] "Vanius"	"Bassano"	"Bellini"
[25] "Giorgione"	"Murillo"	"Palma Giovane"

Column Names

colnames() method in R is used to rename and replace the column names of the data frame in R. The columns of the data frame can be renamed by specifying the new column names as a vector. The new name replaces the corresponding old name of the column in the data frame.

Usage:

colnames(x), where x is an object of class "data.frame".

Example:

```
> colnames(painters)
```

```
[1] "Composition" "Drawing" "Colour" "Expression" "School"
```

Variables

The data set contains four numerical variables (Composition, Drawing, Colour and Expression), as well as one factor variable (School).

Example:

```
> is.numeric(painters$School)
```

```
[1] FALSE
```

```
> is.factor(painters$School)
```

```
[1] TRUE
```

```
> is.numeric(painters$Drawing)
```

```
[1] TRUE
```

```
> is.factor(painters$Drawing)
```

```
[1] FALSE
```

Summary Function

The `summary()` command will provide you with a statistical summary of your data. The output of `summary` command depends on the object you are looking at. It gives the output as the largest value in data, the least value or mean and median and another similar type of information.

Usage:

`summary(x)`, where `x` is an object for which a summary is desired.

Example:

```
> summary(painters)
```

Composition	Drawing	Colour	Expression	School
Min. : 0.00	Min. : 6.00	Min. : 0.00	Min. : 0.000	A :10
1st Qu. : 8.25	1st Qu. : 10.00	1st Qu. : 7.25	1st Qu. : 4.000	D :10
Median :12.50	Median : 13.50	Median :10.00	Median : 6.000	E : 7
Mean :11.56	Mean :12.46	Mean :10.94	Mean : 7.667	G : 7
3rd Qu. :15.00	3rd Qu. :15.00	3rd Qu. :16.00	3rd Qu.: 11.500	B : 6
Max. :18.00	Max. :18.00	Max. :18.00	Max. :18.000	C : 6
				(Other): 8

Test if we are dealing with a data frame:

```
> is.data.frame(painters)
```

```
[1] TRUE
```

Creating data frames

We can create a data frame in R by passing the variable `a,b,c,d` into the `data.frame()` function. We can create data frame and name the columns with `name()` and simply specify the name of the variables.

Example:

```
> x = 1:16
```

```
> y = matrix(x, nrow=4, ncol=4)
```

```
> z = letters[1:16]
```

```
> x
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
```

```
> y
```

```
      [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
```

```
> z
```

```
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p"
```

```
> datafr = data.frame(x, y, z)
```

```
> datafr
```

	x	X1	X2	X3	X4	z
1	1	1	5	9	13	a
2	2	2	6	10	14	b
3	3	3	7	11	15	c
4	4	4	8	12	16	d
5	5	1	5	9	13	e
6	6	2	6	10	14	f
7	7	3	7	11	15	g
8	8	4	8	12	16	h
9	9	1	5	9	13	i
10	10	2	6	10	14	j
11	11	3	7	11	15	k
12	12	4	8	12	16	l
13	13	1	5	9	13	m
14	14	2	6	10	14	n
15	15	3	7	11	15	o
16	16	4	8	12	16	p

Structure of the data

Data structures in R programming are tools for holding multiple values. R's base data structures are often organized by their dimensionality (1D, 2D, or nD) and whether they're homogeneous (all elements must be of the identical type) or heterogeneous (the elements are often of various types).

Usage:

`str(x)`, where `x` is any R object about which you want to have some information.

Example:

```
> str(painters)
'data.frame' :      54 obs. of  5 variables:
 $ Composition:   int      10 15  8 12  0 15  8 15  4 17 ...
 $ Drawing      :   int      8 16 13 16 15 16 17 16 12 18 ...
 $ Colour       :   int     16  4 16  9  8  4  4  7 10 12 ...
 $ Expression   :   int      3 14  7  8  0 14  8  6  4 18 ...
 $ School       :   Factor w/  8 levels "A", "B", "C", "D"..: 1 1 1 1 1 1 1 1 ...
```

Note: int means integer.

How to extract variable from a data frame?

- Extract a variable from data frame using `$`
- Variables can be extracted using the `$` operator followed by the name of the variable.

Suppose we want to extract information on variable `School` from the data set `painters`.

```
> painters$School
[1] A A A A A A A A A A B B B B B B C C C C C C D D D D D
[28] D D D D D E E E E E E E F F F F G G G G G G H H H H
Levels: A B C D E F G H
```

How to extract data from a data frame?

- The data from a data frame can be extracted by using the matrix-style `[row, column]` indexing.

Suppose, if we want to extract information on the first painter `Da Udine` on the variable `Composition` from the data set `painters`.

```
> painters["Da Udine", "Composition"]
```

```
[1] 10
```

Subsets of a Data Frame

A Data Frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. It is the process of selecting a set of desired rows and columns from the data frame.

Usage:

subset(x), where x is an object to be subsetted.

Example:

```
> subset(painters, School=='F')
```

	Composition	Drawing	Colour	Expression	School
Durer	8	10	10	8	F
Holbein	9	10	16	13	F
Pourbus	4	15	6	6	F
Van Leyden	8	6	6	4	F

Splitting of a data frame

Using the 'product' and 'condition' variables, divide the data frame into groups. Use the unsplit() function to restore the original data frame from the split() method. The unsplit() method has the following syntax. Use the split() function in R to split a vector or data frame.

Usage:

split(x), where x is a vector or data frame containing values to be divided into groups.

Example:

Following command splits painters with respect to School (A,B,C,... categories)

```
> splitted = split(painters, painters$School)
```

```
> splitted
```

```
$A
```

	Composition	Drawing	Colour	Expression	School
Da Udine	10	8	16	3	A
Da Vinci	15	16	4	14	A

Del Piombo	8	13	16	7	A
Del Sarto	12	16	9	8	A
Fr. Penni	0	15	8	0	A
Guilio Romano	15	16	4	14	A
Michelangelo	8	17	4	8	A
Perino del Vaga	15	16	7	6	A
Perugino	4	12	10	4	A
Raphael	17	18	12	18	A

\$B

	Composition	Drawing	Colour	Expression	School
F. Zucarro	10	13	8	8	B
Fr. Salviata	13	15	8	8	B
Parmigiano	10	15	6	6	B
Primaticcio	15	14	7	10	B
T. Zucarro	13	14	10	9	B
Volterra	12	15	5	8	B

And so on.

9. STATISTICAL FUNCTIONS

Suppose there are 10 persons coded into two categories as male and female.

M, F, M, F, M, M, M, F, M, M.

Use a1 and a2 to refer to male and female categories. There are 7 male and 3 female persons, denoted as $n_1 = 7$ and $n_2 = 3$. The number of observations in a particular category is called the absolute frequency.

The relative frequencies of a1 and a2 are

$$f_1 = \frac{n_1}{n_1 + n_2} = \frac{7}{10} = 0.7 = 70\%$$

$$f_2 = \frac{n_2}{n_1 + n_2} = \frac{3}{10} = 0.3 = 30\%$$

This gives us information about the proportions of male and female.

Absolute and Relative Frequencies

Table uses the cross-classifying factors to build a contingency table of the counts at each combination of factor levels. In R, the command `table()` creates the absolute frequency of the variable of the data file.

Usage:

- `table(x)`
- `table(x)/length(x)`, where x is one or more objects which can be interpreted as factors or a list.

Example:

- **ABSOLUTE FREQUENCY**

```
> gender = c(1, 2, 1, 2, 1, 1, 1, 2, 1, 1)
```

```
> gender
```

```
[1] 1 2 1 2 1 1 1 2 1 1
```

```
> table(gender)
```

```
1 2
```

```
7 3
```

- **RELATIVE FREQUENCY**

```
> table(gender)/length(gender)
```

1 2
0.7 0.3

Partition Values

The Partition Values are the measures used in statistics for dividing the total number for dividing the total number of observations of a distribution into certain number of equal parts.

Quartile: Divides the data into 4 equal parts.

Decile: Divides the data into 10 equal parts.

Percentile: Divides the data into 100 equal parts.

Quantile

The generic function `quantile` produces sample quantiles corresponding to the given probabilities. The smallest observation corresponds to a probability of 0 and the largest to a probability of 1.

Usage:

- `quantile(x, ...)`
- `quantile(x, probs = seq(0, 1, 0.25),...)`, where `x` is a numeric vector.

Example:

```
> marks = c(68, 82, 63, 86, 34, 96, 41, 89, 29, 51, 75, 77, 56, 59, 42)
```

```
> quantile(marks)
```

```
0% 25% 50% 75% 100%
```

```
29.0 46.5 63.0 79.5 96.0
```

```
> quantile(marks, probs=c(0,0.25,0.5,0.75,1))
```

```
0% 25% 50% 75% 100%
```

```
29.0 46.5 63.0 79.5 96.0
```

Variability

Variability (also known as Statistical Dispersion) is one of the features of descriptive statistics. Variability shows the spread of a data set around a point.

Data: x_1, x_2, \dots, x_n where x is a data vector.

To find the variability of this data:

```
> marks = c(68, 82, 63, 86, 34, 96, 41, 89, 29, 51, 75, 77, 56, 59, 42)
```

- **Variance**

$$\text{var}(x) = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$$

Usage: var(x), where x is a numeric vector, matrix or data frame.

Example:

```
> var(marks)
```

```
[1] 439.3143
```

- **Range**

maximum(x_1, x_2, \dots, x_n) - minimum(x_1, x_2, \dots, x_n)

USAGE: max(x) - min(x), where x is any numeric or character objects.

- **Interquartile Range**

Third quartile(x_1, x_2, \dots, x_n) - First quartile(x_1, x_2, \dots, x_n)

Usage: IQR(x), where x is a numeric vector.

Example:

```
> IQR(marks)
```

```
[1] 33
```

- **Quartile Deviation**

[Third quartile(x_1, x_2, \dots, x_n) - First quartile(x_1, x_2, \dots, x_n)]/2 = Interquartile range/2

Usage: IQR(x)/2, where x is a numeric vector.

Example:

```
> IQR(marks)/2
```

```
[1] 16.5
```

Correlation

Let x, y be the two data vectors

Data: $x = (x_1, x_2, \dots, x_n)$ and $y = (y_1, y_2, \dots, y_n)$

- **Covariance**

$$\text{cov}(x, y) = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$$

Usage: $\text{cov}(x, y)$: covariance between x and y

Example:

```
> cov( c(1,2,3,4), c(1,2,3,4) )
```

```
[1] 1.666667
```

```
> cov( c(1,2,3,4), c(-1,-2,-3,-4) )
```

```
[1] -1.666667
```

- **Correlation Coefficient**

Measures the degree of linear relationship between the two variables.

$$r_{xy} = \frac{\text{cov}(x, y)}{\sqrt{\text{var}(x) \text{var}(y)}} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 \sum_{i=1}^n (y_i - \bar{y})^2}}$$

$$-1 \leq r_{xy} \leq 1$$

Usage: $\text{cor}(x, y)$: correlation between x and y

Example:

- Exact positive linear dependence

```
> cor( c(1,2,3,4), c(1,2,3,4) )
```

```
[1] 1
```

- Exact negative linear dependence

```
> cor( c(1,2,3,4), c(-1,-2,-3,-4) )
```

```
[1] -1
```

Graphics and Plots

The graphics package is an R base package for creating graphs. The plot function is the most basic function to create plots in R. With this plotting function you can create several types of plots, like line charts, bar plots or even boxplots, depending on the input.

Bar Plots

A bar plot is used to display the relationship between a numeric and a categorical variable. Each entity of the categorical variable is represented as a bar. The size of the bar represents its numeric value.

Usage:

barplot(x), where x refers to the vector of values for which the barplot is desired.

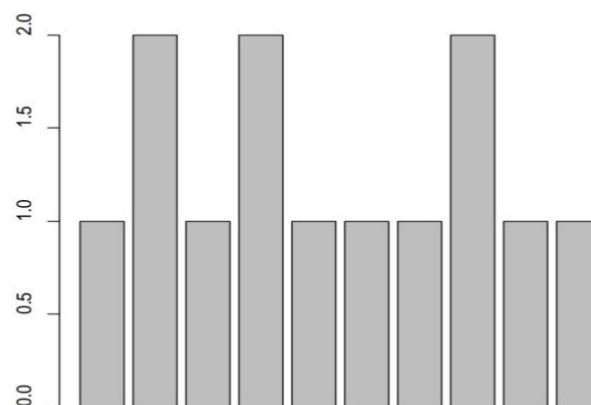
Example:

```
> gender = c(1, 2, 1, 2, 1, 1, 1, 2, 1, 1)
```

```
> gender
```

```
[1] 1 2 1 2 1 1 1 2 1 1
```

```
> barplot(gender)
```



Pie Diagram

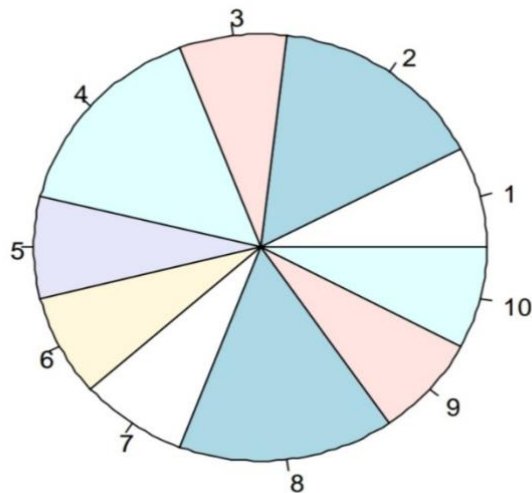
A pie chart is a representation of values as slices of a circle with different colors. The slices are labeled and the numbers corresponding to each slice is also represented in the chart. In R the pie chart is created using the pie() function which takes positive numbers as a vector input.

Usage:

pie(x), where x refers to a vector of values for which the Pie diagram is desired.

Example:

```
> pie(gender)
```



Histogram

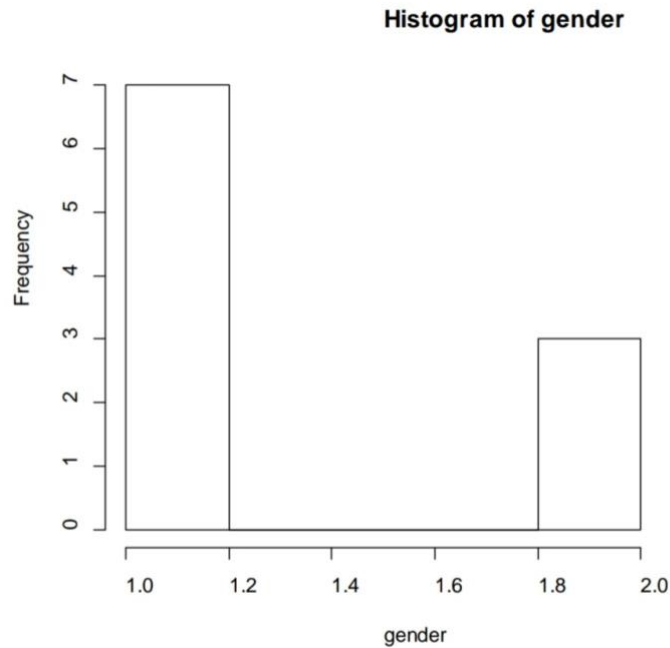
A histogram represents the frequencies of values of a variable bucketed into ranges. Histogram is similar to bar chart but the difference is it groups the values into continuous ranges. Each bar in histogram represents the height of the number of values present in that range. R creates histogram using hist() function.

Usage:

hist(x), where x refers to a vector of values for which the histogram is desired.

Example:

```
> hist(gender)
```



Boxplot

Boxplots are a measure of how well distributed is the data in a data set. It divides the data set into three quartiles. This graph represents the minimum, maximum, median, first quartile and third quartile in the data set.

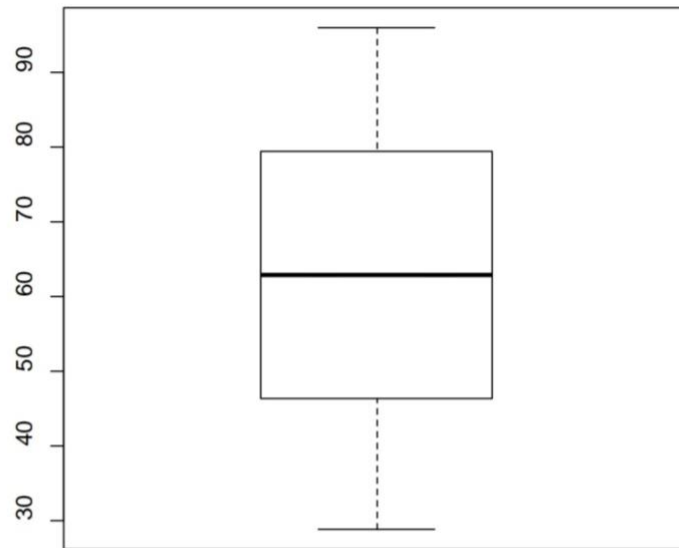
Usage:

`boxplot(x)`, where `x` refers to the vector of values for which the boxplot is desired.

Example:

```
> marks = c(68, 82, 63, 86, 34, 96, 41, 89, 29, 51, 75, 77, 56, 59, 42)
```

```
> boxplot(marks)
```



Scatter Plot

A scatter plot is a set of dotted points to represent individual pieces of data in the horizontal and vertical axis. A graph in which the values of two variables are plotted along X-axis and Y-axis, the pattern of the resulting points reveals a correlation between them.

Usage:

- `plot(x, y)`
- `plot(x, y, type)`

type	
"p" for points	"b" for both
"l" for lines	"c" for the lines part alone of "b"
"o" for both 'overplotted'	"s" for stair steps.
"h" for 'histogram' like (or 'high-density') vertical lines	

Parameters:

- `x`: This parameter sets the horizontal coordinates.
- `y`: This parameter sets the vertical coordinates.
- `xlab`: This parameter is the label for horizontal axis.
- `ylab`: This parameter is the label for vertical axis.
- `main`: This parameter main is the title of the chart.
- `xlim`: This parameter is used for plotting values of `x`.
- `ylim`: This parameter is used for plotting values of `y`.
- `axes`: This parameter indicates whether both axes should be drawn on the plot.

Examples:

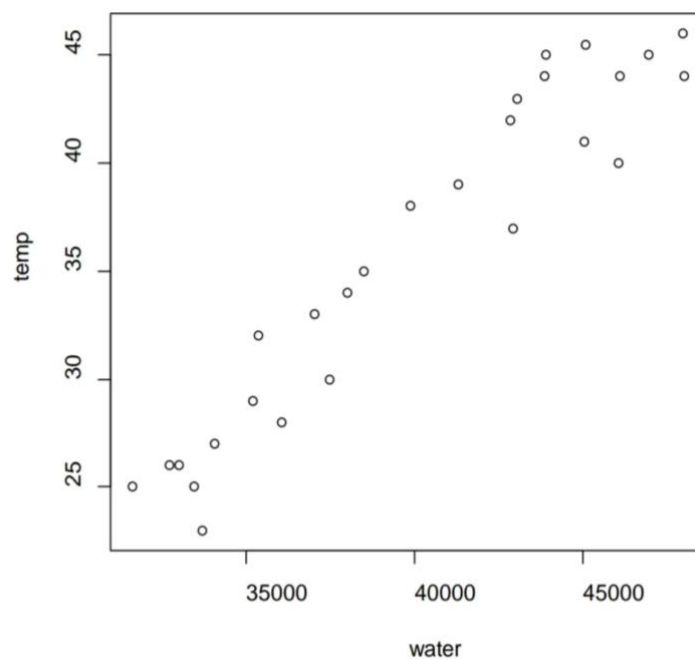
Daily water demand in a city depends upon weather temperature. We know from experience that water consumption increases as weather temperature increases. Data on 27 days is collected as follows:

Daily water demand (in million litres), Temperature (in centigrade)

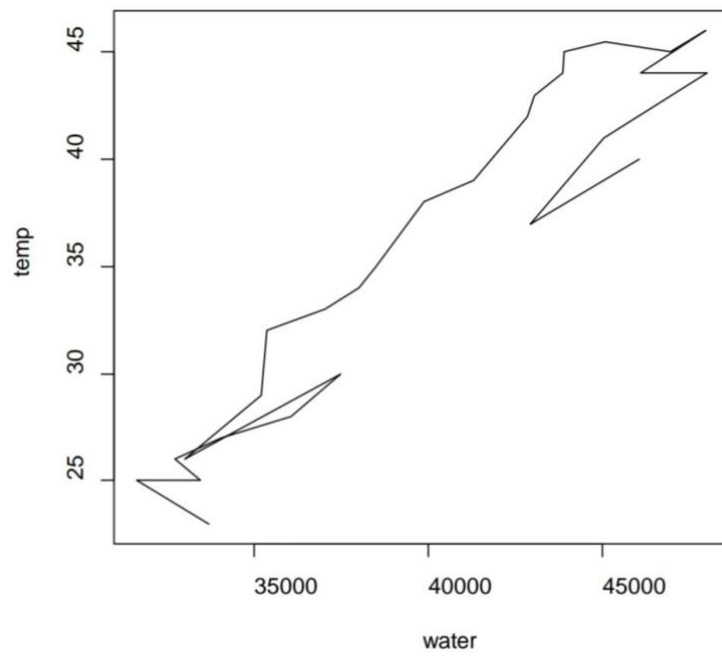
```
> water = c(33710, 31666, 33495, 32758, 34067, 36069, 37497, 33044, 35216, 35383, 37066,  
38037, 38495, 39895, 41311, 42849, 43038, 43873, 43923, 45078, 46935, 47951, 46085,  
48003, 45050, 42924, 46061)
```

```
> temp = c(23,25,25,26,27,28,30,26,29,32,33,34,35,38,39,42,43,44,45,45.5,  
45,46,44,44,41,37,40)
```

```
> plot(water, temp)
```

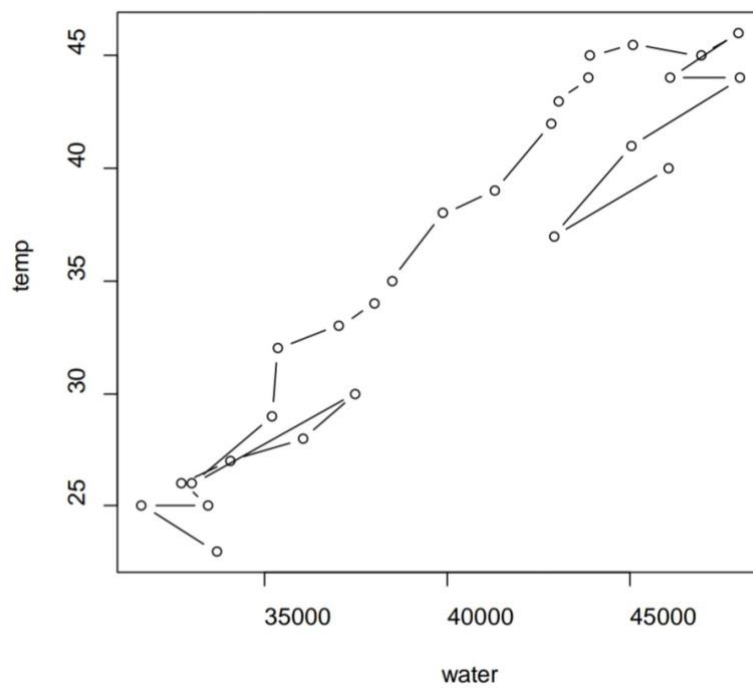


```
> plot(water, temp, "l")
```

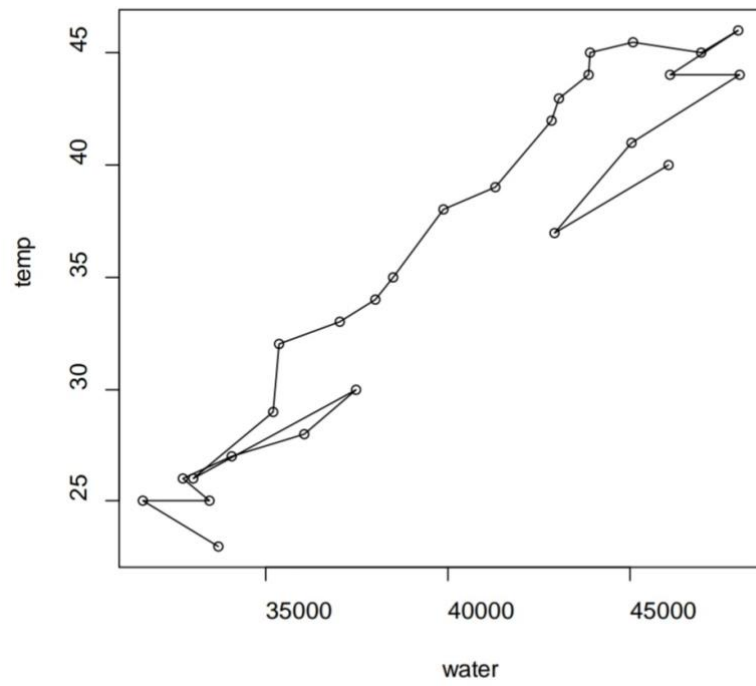


7

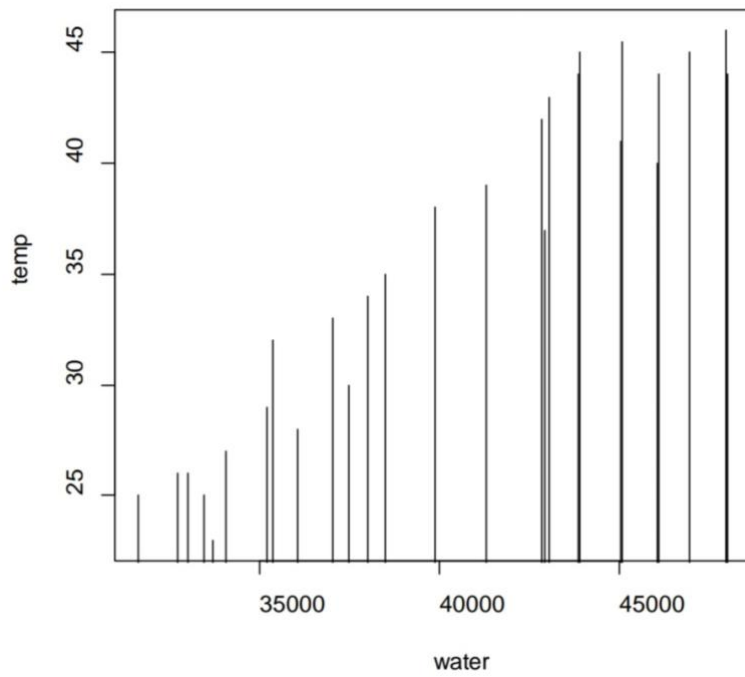
```
> plot(water, temp, "b")
```



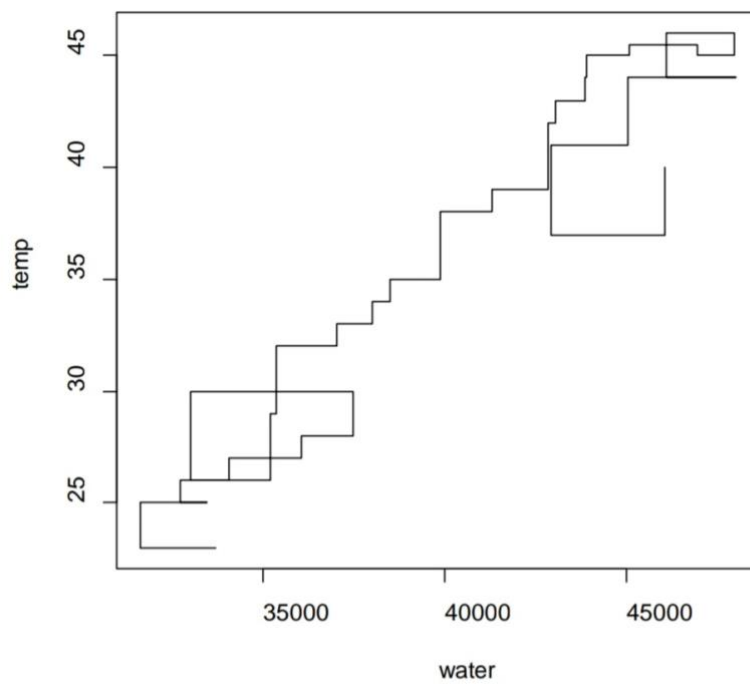
```
> plot(water, temp, "o")
```



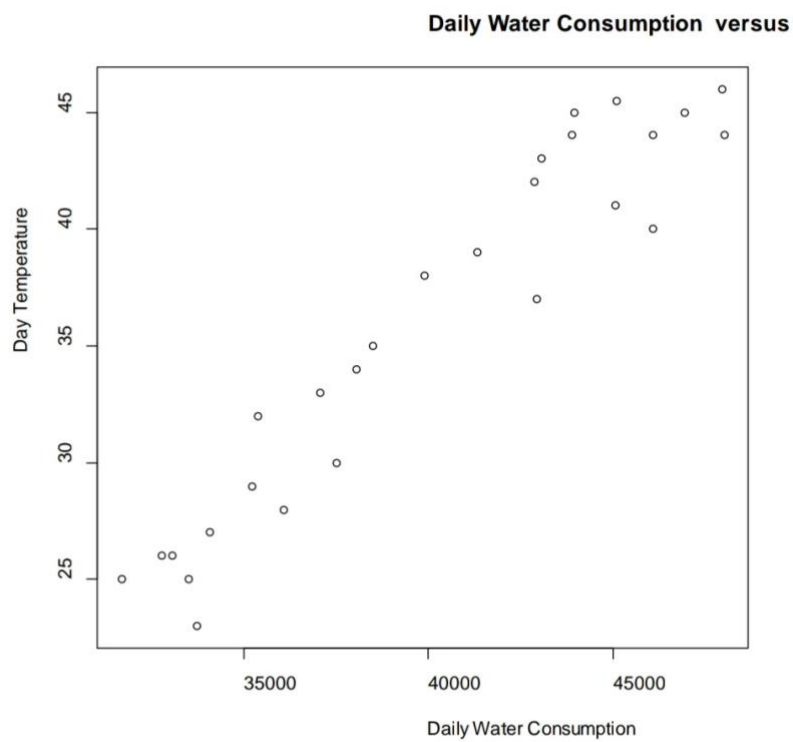
```
> plot(water, temp, "h")
```



```
> plot(water, temp, "s")
```



```
> plot(water, temp, xlab=" Daily Water Consumption ", ylab=" Day Temperature ",
main="Daily Water Consumption versus Day Temperature")
```



Smooth Scatter Plot

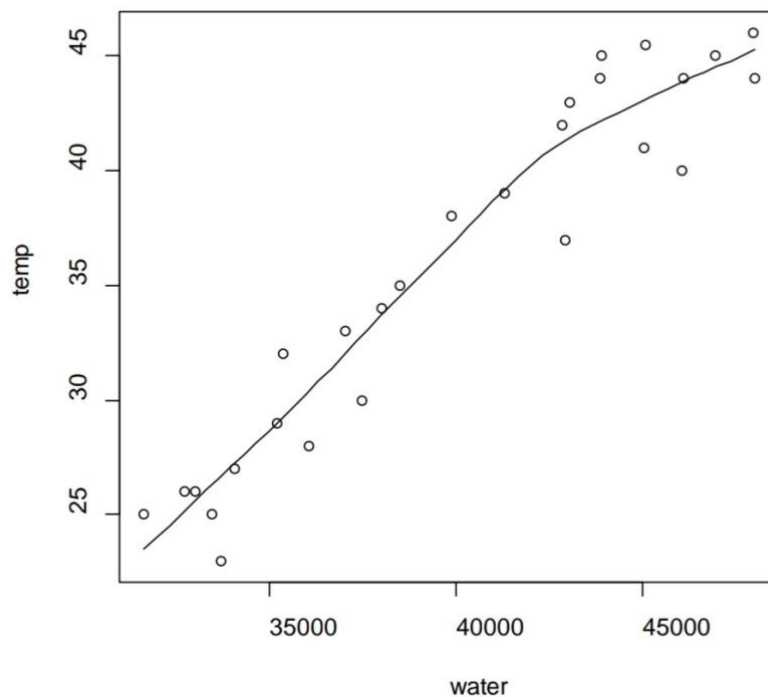
In statistical, several Scatter plot smoothing methods are available to fit a function through the points of a scatterplot to best represent the relationship between the variables.

Usage:

`scatter.smooth(x,y)`, where x, y refers to the arguments that provide the x and y coordinates for the plot.

Example:

```
>scatter.smooth(water,temp)
```



More Functions:

- `contour()` for contour lines
- `dotchart()` for dot charts (replacement for bar charts)
- `image()` pictures with colors as third dimension
- `mosaicplot()` mosaic plot for (multidimensional) diagrams of categorical variables (contingency tables)
- `persp()` perspective surfaces over the x–y plane

10. R PROGRAMMING

Steps to write a programme:

- ◆ A programme is a set of instructions or commands which are written in a sequence of operations i.e., what comes first and what comes after that.
- ◆ The objective of a programme is to obtain a defined outcome based on input variables.
- ◆ The computer is instructed to perform the defined task.
- ◆ Computer is an obedient worker but it has its own language.
- ◆ We do not understand computer's language and computer does not understand our language.
- ◆ The software helps us and works like an interpreter between us and computer.
- ◆ We say something in software's language and software informs it to computer.
- ◆ Computer does the task and informs back to software.
- ◆ The software translates it to our language and informs us.
- ◆ Programme in R is written as a function using function.
- ◆ Write down the objective, i.e., what we want to obtain as an outcome.
- ◆ Translate it in the language of R.
- ◆ Identify the input and output variables.
- ◆ Identify the nature of input and output variables, i.e., numeric, string, factor, matrix etc.
- ◆ Input and output variables can be single variable, vector, matrix or even a function itself.
- ◆ The input variables are the component of function which are reported in the argument of function()
- ◆ The output of a function can also be input to another function.
- ◆ The output of an outcome can be formatted as per the need and requirement.

Tips

- ❖ Loops usually slower the speed of programmes, so better is to use vectors and matrices.
- ❖ Use # symbol to write comment to understand the syntax.
- ❖ Use the variable names which are easy to understand.
- ❖ Don't forget to initialize the variables.

Example:

Suppose we want to compute

$$f(x) = \begin{cases} \exp\left(\frac{x + \ln(1 + x^3)}{x^2}\right) & \text{if } x > 0 \\ 10 & \text{if } x = 0 \\ \frac{2 + x^3}{x} & \text{if } x < 0 \end{cases}$$

At a Glance:

```
> f = function(x)
{
  if(x>0) {exp((x+log(1+x^3))/x^2)}
  else if(x==0) {10}
  else {(2+x^3)/x}
}
```

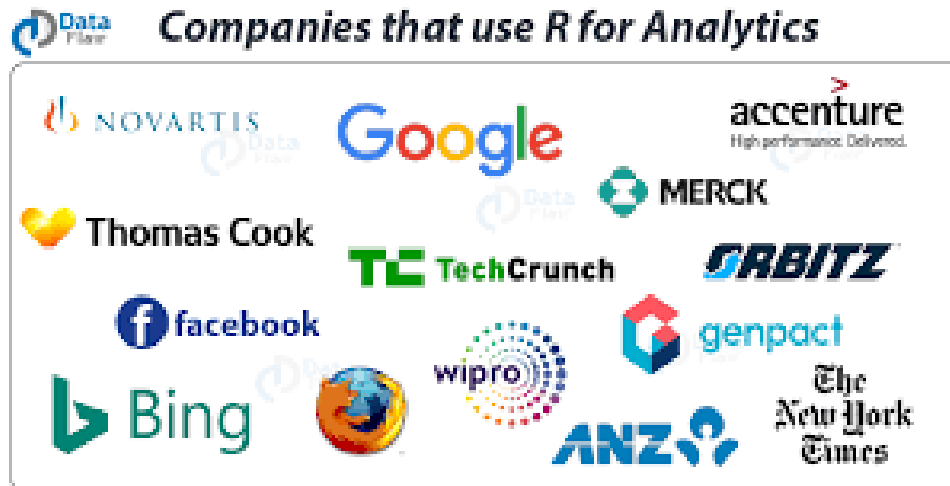
Output:

```
> f(8)
[1] 1.249201
> f(-4)
[1] 15.5
> f(0)
[1] 10
```

APPLICATIONS

Real-Life uses of R

R applications are not enough until you don't know how people/companies are using the R programming language.



1. **Facebook** – Facebook uses R to update status and its social network graph. It is also used for predicting colleague interactions with R.
2. **Ford Motor Company** – Ford relies on Hadoop. It also relies on R for statistical analysis as well as carrying out data-driven support for decision making.
3. **Google** – Google uses R to calculate ROI on advertising campaigns and to predict economic activity and also to improve the efficiency of online advertising.
4. **Foursquare** – R is an important stack behind Foursquare's famed recommendation engine.
5. **Microsoft** – Microsoft uses R for the Xbox matchmaking service and also as a statistical engine within the Azure ML framework.
6. **Mozilla** – It is the foundation behind the Firefox web browser and uses R to visualize web activity.
7. **New York Times** – R is used in the news cycle at The New York Times to crunch data and prepare graphics before they go for printing.
8. **Thomas Cook** – Thomas Cook uses R for prediction and also **Fuzzy Logic Systems** to automate price settings of their last-minute offers.
9. **National Weather Service** – The National Weather Service uses R at its River Forecast Centers. Thus, it is used to generate graphics for flood forecasting.
10. **Twitter** – R is part of Twitter's Data Science toolbox for sophisticated statistical modeling.

Pros and Cons of R

R is one of the most popular languages for statistical modeling and analysis. But like every other programming language, R has its own set of benefits and limitations. R is a continuously evolving language. This means that many of the cons will gradually fade away with the future updates of R.

Advantages of R:

- R is the most comprehensive statistical analysis package. As new technology and concepts often appear first in R.
- As R programming language is an open source. Thus, you can run R anywhere and at any time.
- R programming language is suitable for GNU/Linux and Windows operating system.
- R programming is cross-platform which runs on any operating system.
- In R, everyone is welcome to provide new packages, bug fixes, and code enhancements.

Disadvantages of R:

- In the R programming language, the standard of some packages is less than perfect.
- Although, R commands give little pressure to memory management. So R programming language may consume all available memory.
- In R basically, nobody to complain if something doesn't work.
- R programming language is much slower than other programming languages such as Python and MATLAB.

We got to know the positive aspects of R Language which place us a step ahead towards generating our interest in learning R. We also inferred many of its weaknesses but, most of them are under the correction phase through several upgrades and further development. We believe that many of the limitations will be eradicated in future.

CONCLUSION

As a conclusion, R is the most popular analytic tool for data analysis and statistics, having approximately 2 million users. It is ideal for all data analytics operations.

Being an open-source language, it is continuously expanding, people from all over the world are contributing to its development.

The platform independence, diversity of packages, and robust graphical features add an advantage to this primary tool in the analytics industry.

Due to a shortfall of data analysts, various jobs are available for R programmers in the Data Analyst Industry. Both novice and professionals have a place in this industry.

Apart from the IT industry, several other industries are using data to transform problems into solutions -

- Financial Sectors
- Banks
- Health Organizations
- Manufacturing companies
- Academia
- Governmental departments

Companies like Facebook, Google, Twitter are adopting R to meet their analytical goals. Emerging startups are moving on the same path.

The adoption of R in data-driven companies is increasing rapidly and will flourish in the years to come.

However, these organizations expect their new employees to be up to date with R. They want them to be familiar with R and its use for Data Analytics. With so many advantages, this language will continue to grow in popularity in the world of statistical computing and data analytics.

REFERENCES

1. Introduction to Statistics and Data Analysis - With Exercises, Solutions and Applications in R - By Christian Heumann, Michael Schomaker and Shalabh, Springer, 2016.
2. The R Software-Fundamentals of Programming and Statistical Analysis -Pierre Lafaye de Micheaux, Rémy Drouilhet, Benoit Liquet, Springer 2013.
3. A Beginner's Guide to R (Use R) By Alain F. Zuur, Elena N. Ieno, Erik H.W.G. Meesters, Springer 2009.
4. <https://www.geeksforgeeks.org/r-programming-language-introduction>

BASICS OF PYTHON AND MATHEMATICAL COMPUTATIONS USING PYTHON PROGRAMMING

Project Report submitted to

ST.MARY'S COLLEGE (AUTONOMOUS), THOOTHUKUDI

Affiliated to

MANONMANIAM SUNDARANAR UNIVERSITY, TIRUNELVELI

In partial fulfillment of the requirement for the award of degree of

Bachelor of Science in Mathematics

Submitted by

NAME

ANGEL MERCY. J

HARITHA. S

MARIA ANTONY SNOWBA. J

SELIN PREETHI. A

SHALINI. M

REG.NO.

19AUMT01

19AUMT12

19AUMT22

19AUMT40

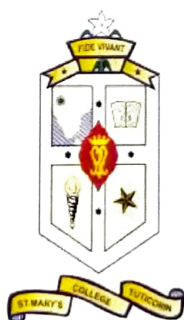
19AUMT42

Under the Guidance of

Dr. Sr. S. KULANDAI THERESE M.Sc., B.Ed., M.Phil., Ph.D.

Assistant Professor of Mathematics

St. Mary's College (Autonomous), Thoothukudi.



Department of Mathematics

St. Mary's College (Autonomous), Thoothukudi

(2021 - 2022)

**BASICS OF PYTHON AND MATHEMATICAL COMPUTATIONS
USING PYTHON PROGRAMMING**

Project Report submitted to

ST.MARY'S COLLEGE (AUTONOMOUS), THOOTHUKUDI

Affiliated to

MANONMANIAM SUNDARANAR UNIVERSITY, TIRUNELVELI

In partial fulfillment of the requirement for the award of degree of

Bachelor of Science in Mathematics

Submitted by

NAME

ANGEL MERCY. J

HARITHA. S

MARIA ANTONY SNOWBA. J

SELIN PREETHI. A

SHALINI. M

REG.NO.

19AUMT01

19AUMT12

19AUMT22

19AUMT40

19AUMT42

Under the Guidance of

Dr. Sr. S. KULANDAI THERESE M.Sc., B.Ed., M.Phil., Ph.D.

Assistant Professor of Mathematics

St. Mary's College (Autonomous), Thoothukudi.



Department of Mathematics

St. Mary's College (Autonomous), Thoothukudi

(2021 - 2022)

CERTIFICATE

We hereby declare that the project report entitled "BASICS OF PYTHON AND MATHEMATICAL COMPUTATIONS USING PYTHON" being submitted to St. Mary's College (Autonomous), Thoothukudi affiliated to Manonmaniam Sundaranar University, Tirunelveli in partial fulfillment for the award of degree of Bachelor of Science in Mathematics and it is a record of work done during the year 2021 - 2022 by the following student :

NAME

REG.NO.

ANGEL MERCY. J

19AUMT01

HARITHA. S

19AUMT12

MARIA ANTONY SNOWBA. J

19AUMT22

SELIN PREETHI. A

19AUMT40

SHALINI. M

19AUMT42



Signature of the Guide

Dr. S. KULANDAI THERESE

M.Sc., B.Ed., M.Phil., Ph.D.,

Assistant Professor,

Department of Mathematics,

St. Mary's College (Autonomous),

Thoothukudi - 628 001.

V.L. Stella Arputha Mary
Signature of the HOD

Dr. V.L. Stella Arputha Mary

M.Sc., M.Phil., B.Ed., Ph.D.,

Head & Asst Professor of Mathematics

St. Mary's College (Autonomous)

Thoothukudi-628 001.



Signature of the Examiner



Signature of the Principal

Principal

St. Mary's College (Autonomous)

Thoothukudi-628 001.

DECLARATION

We hereby declare that the project reported entitled "BASICS OF PYTHON AND MATHEMATICAL COMPUTATIONS USING PYTHON PROGRAMMING", is our original work. It has not been submitted to any university for any degree or diploma.

J. Angel Mercy
(ANGEL MERCY. J)

S. Haritha
(HARITHA. S)

A. Selin Preethi
(SELIN PREETHI. A)

M. Shalini
(SHALINI. M)

J. Maria Antony Snowba
(MARIA ANTONY SNOWBA. J)

ACKNOWLEDGEMENT

First of all, we thank Lord Almighty for showering his blessings to undergo this project.

With immense pleasure, we register our deep sense of gratitude to our guide **Dr. Sr. S. Kulandai Therese M.Sc., B.Ed., M.Phil., Ph.D.** and the Head of the Department, **Dr. V. L. Stella Arputha Mary M.Sc., M.Phil., B.Ed., Ph.D.** for having imparted necessary guidelines throughout the period of our studies.

We thank our beloved Principal, **Rev. Dr. Sr. A.S.J. Lucia Rose M.Sc., M.Phil., Ph.D., PGDCA** for providing us the help to carry out our project work successfully.

Finally, we thank all those who extended their helping hands regarding this project.

BASICS OF PYTHON AND MATHEMATICAL COMPUTATIONS USING PYTHON PROGRAMMING

PREFACE

The topic of our Project "BASICS OF PYTHON AND MATHEMATICAL COMPUTATIONS USING PYTHON PROGRAMMING", focuses on underlying concepts of the discipline and behavioural aspects in python module and mathematical expressions. Python is probably one of the few programming languages which is both simple and powerful. This is good for beginners as well as for experts, and more importantly, is fun to program with. This project aims to help you learn this wonderful language and show how to get things done quickly and painlessly - in effect 'The Anti-venom to your programming problems'.

Basics in python, mathematical operations on arrays, relevant examples and references have also been discussed to scaffold the readers on the necessary concepts. After brief learning, it was a natural progression to apply the learned concepts and practices in real life. Python has multitude of applications in almost all areas of Data science which have been discussed in our project. the Project is structured into five chapters :

Chapter 1 presents briefly the basics of python in arithmetical operations.

Chapter 2 deals with Python NumPy module which is used to create vectors. We use `numpy.array()` method to create a one dimensional array i.e. a vector.

Chapter 3 introduces the symbols and operations, factorizing and expanding expressions, pretty printing and solving equations respectively.

Chapter 4 focuses on creating matrices, transpose, addition, subtraction, determinants, scalar multiplication and solving linear equations

Chapter 5 deals with set construction, creating sets from lists or tuples, subsets, supersets and powersets and Set operations

CONTENT

Chapter 1

1.1	Arithmetic Operators	9
1.2	Comparison operators	11
1.3	Python Package Numpy	13
1.3.1	Creating a Numpy array	14
1.3.2	Basic Array Operations	15
1.4	Array creation	16
1.4.1	Python program to demonstrate	17

Chapter 2

2.1	Vectors	19
2.2	Indexing	19
2.3	Assignment versus copying	20
2.4	Zero vectors	20
2.5	Vector addition	21
2.6	Scalar-vector multiplication	22
2.7	Checking properties	22
2.8	Inner product	23
2.9	Complexity of vector computations	24

Chapter 3

3.1	Defining symbols and operations	26
3.2	Factorizing and Expanding Expressions	27
3.3	Pretty Printing	28
3.4	Substituting in Values	29
3.5	Solving Equations	30
3.5.1	Solving Quadratic Equations	31
3.6	Solving a system of linear equations	31

Chapter 4

4.1	Matrices	33
4.1.1	Creating matrices from the entries	33
4.1.2	Indexing entries	33
4.1.3	Equality of matrices	33
4.1.4	Row and column vectors	33
4.1.5	Slicing and submatrices	34
4.1.6	Block matrices	34
4.2	Zero and identity matrices	34
4.3	Transpose and addition	35
4.4	Matrix – matrix multiplication	36
4.5	Determinants	36
4.6	Left and right inverses	38
4.7	Solving linear equations	39

Chapter 5

5.1	What's a set?	40
5.2	Set construction	40
5.2.1	Checking Whether a Number Is in a set	41
5.2.2	Creating an empty set	41
5.2.3	Creating Sets from Lists or Tuples	41
5.2.4	Set Repetition and Order	41
5.3	Subsets, Supersets, and Power Sets	42
5.4	Set Operations	44
5.4.1	Union and Intersection	44
5.4.2	Cartesian Product	46
6	Applications of python	47
7	Conclusion	51
8	References	52

1. Python arithmetic operations

Python operators

Python Operators in general are used to perform operations on values and variables. These are standard symbols used for the purpose of logical and arithmetic operations.

1.1 Arithmetic Operators

Arithmetic operators are used to performing mathematical operations like addition, subtraction, multiplication, and division.

Operator	Description	Syntax
+	Addition: adds two operands	$x + y$
-	Subtraction: subtracts two operands	$x - y$
*	Multiplication: multiplies two operands	$x * y$
/	Division (float): divides the first operand by the second	x / y
//	Division (floor): divides the first operand by the second	$x // y$
%	Modulus: returns the remainder when the first operand is divided by the second	$x \% y$

**	Power: Returns first raised to power second	x ** y
----	---	--------

Examples of Arithmetic operators in Python

Examples of Arithmetic Operator

a = 9

b = 4

Addition of numbers

add = a + b

Subtraction of numbers

sub = a - b

Multiplication of number

mul = a * b

Division(float) of number

div1 = a / b

Division(floor) of number

div2 = a // b

Modulo of both number

mod = a % b

Power

p = a ** b

print results

```
print(add)
print(sub)
print(mul)
print(div1)
print(div2)
print(mod)
print(p)
```

Output

13

5

36

2.25

2

1

6561

1.2 Comparison operators

Comparison of relational operators compares the values. It either returns **True** or **False** according to the condition.

Operator	Description	Syntax
>	Greater than: True if the left operand is greater than the right	$x > y$
<	Less than: True if the left operand is less than the right	$x < y$

<code>==</code>	Equal to: True if both operands are equal	<code>x == y</code>
<code>!=</code>	Not equal to – True if operands are not equal	<code>x != y</code>
<code>>=</code>	Greater than or equal to True if the left operand is greater than or equal to the right	<code>x >= y</code>
<code><=</code>	Less than or equal to True if the left operand is less than or equal to the right	<code>x <= y</code>

Examples of Comparison Operators in Python

#Examples of Relational Operators

`a = 13`

`b = 33`

`# a > b is False`

`print(a > b)`

`# a < b is True`

`print(a < b)`

`# a == b is False`

```
print(a == b)
```

```
# a != b is True
```

```
print(a != b)
```

```
# a >= b is False
```

```
print(a >= b)
```

```
# a <= b is True
```

```
print(a <= b)
```

Output

False

True

False

True

False

True

1.3 Python Package Numpy

Numpy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data.

Array in Numpy is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. In Numpy, number of dimensions of the array is called rank of the

array. A tuple of integers giving the size of the array along each dimension is known as shape of the array. An array class in Numpy is called as ndarray.

1.3.1 Creating a Numpy array

Arrays in Numpy can be created by multiple ways, with various number of Ranks, defining the size of the Array. Arrays can also be created with the use of various data types such as lists, tuples, etc. The type of the resultant array is deduced from the type of the elements in the sequences.

```
# Python program for
```

```
# Creation of Arrays
```

```
import numpy as np
```

```
# Creating a row 1 Array
```

```
arr = np.array([1, 2, 3])
```

```
print("Array with Rank 1: \n", arr)
```

```
# Creating a row 2 Array
```

```
arr = np.array([[1, 2, 3],  
                [4, 5, 6]])
```

```
print("Array with Row 2: \n", arr)
```

```
# Creating an array from tuple
```

```
arr = np.array((1, 3, 2))
```

```
print("\nArray created using "  
      "passed tuple:\n", arr)
```

Output

Array with Rank 1:

```
[1 2 3]
```

Array with Rank 2:

```
[[1 2 3]
```

```
[4 5 6]]
```

Array created using passed tuple:

```
[1 3 2]
```

1.3.2 Basic Array Operations

In numpy, arrays allow a wide range of operations which can be performed on a particular array or a combination of Arrays. These operation includes some basic Mathematical operation as well as Unary and Binary operations.

```
# Python program to demonstrate
```

```
# basic operations on single array
```

```
import numpy as np
```

```
# Defining Array 1
```

```
a = np.array([[1, 2],
```

```
               [3, 4]])
```

```
# Defining Array 2
```

```
b = np.array([[4, 3],
```

```
               [2, 1]])
```

```
# Adding 1 to every element
```

```
print ("Adding 1 to every element:", a + 1)
```

```
# Subtracting 2 from each element
```

```
print ("\nSubtracting 2 from each element:", b - 2)
```

```
# sum of array elements
```

```
# Performing Unary operations
```

```
print ("\nSum of all array "
```

```
      "elements: ", a.sum())
```

```
# Adding two arrays
```

Angel Meray

```
# Performing Binary operations  
print ("\nArray sum:\n", a + b)
```

Output

Adding 1 to every element:

```
[[2 3]
```

```
[4 5]]
```

Subtracting 2 from each element:

```
[[ 2  1]
```

```
[ 0 -1]]
```

Sum of all array elements: 10

Array sum:

```
[[5 5]
```

```
[5 5]]
```

1.4 Array creation

There are various ways to create arrays in NumPy.

- For example, you can create an array from a regular Python **list** or **tuple** using the **array** function. The type of the resulting array is deduced from the type of the elements in the sequences.
- Often, the elements of an array are originally unknown, but its size is known. Hence, NumPy offers several functions to create arrays with **initial placeholder content**. These minimize the necessity of growing arrays, an expensive operation. **For example:** `np.zeros`, `np.ones`, `np.full`, `np.empty`, etc.
- To create sequences of numbers, NumPy provides a function analogous to `range` that returns arrays instead of lists.
- **arange:** returns evenly spaced values within a given interval. **step** size is specified.
- **linspace:** returns evenly spaced values within a given interval. **num** no. of elements are returned.

- **Reshaping array:** We can use **reshape** method to reshape an array. Consider an array with shape $(a_1, a_2, a_3, \dots, a_N)$. We can reshape and convert it into another array with shape $(b_1, b_2, b_3, \dots, b_M)$. The only required condition is: $a_1 \times a_2 \times a_3 \dots \times a_N = b_1 \times b_2 \times b_3 \dots \times b_M$. (i.e original size of array remains unchanged.)
- **Flatten array:** We can use **flatten** method to get a copy of array collapsed into **one dimension**. It accepts order argument. Default value is 'C' (for row-major order). Use 'F' for column major order.

1.4.1 Python program to demonstrate

array creation techniques

```
import numpy as np
```

```
# Creating array from list with type float
```

```
a = np.array([[1, 2, 4], [5, 8, 7]], dtype = 'float')
```

```
print ("Array created using passed list:\n", a)
```

```
# Creating array from tuple
```

```
b = np.array((1, 3, 2))
```

```
print ("\nArray created using passed tuple:\n", b)
```

```
# Creating a 3X4 array with all zeros
```

```
c = np.zeros((3, 4))
```

```
print ("\nAn array initialized with all zeros:\n", c)
```

```
# Create a constant value array of complex type
```

```
d = np.full((3, 3), 6, dtype = 'complex')
```

```
print ("\nAn array initialized with all 6s."
```

```
      "Array type is complex:\n", d)
```

Output

Array created using passed list:

```
[[ 1.  2.  4.]
```

```
 [ 5.  8.  7.]]
```


Array created using passed tuple:

```
[1 3 2]
```

An array initialized with all zeros:

```
[[ 0.  0.  0.  0.]
```

```
[ 0.  0.  0.  0.]
```

```
[ 0.  0.  0.  0.]]
```

An array initialized with all 6s. Array type is complex:

```
[[ 6.+0.j  6.+0.j  6.+0.j]
```

```
[ 6.+0.j  6.+0.j  6.+0.j]
```

```
[ 6.+0.j  6.+0.j  6.+0.j]]
```

2. VECTORS

2.1 Vectors

The simplest way to represent vectors in Python is using a list structure. A vector is constructed by giving the list of elements surrounded by square brackets, with the elements separated by commas. The assignment operator = is used to give a name to the list. The len() function returns the size (dimension).

```
In [ ] : x = [-1.1, 0.0, 3.6, -7.2]
        len(x)
```

```
Out[ ] : 4
```

Some common mistakes. Don't forget the commas between entries, and be sure to use square brackets. Otherwise, you'll get things that may or may not make sense in Python, but are not vectors.

```
In [ ] : x = [-1.1 0.0 3.6 -7.2]
```

```
Out[ ] : File "<ipython-input-1-18f108d7fed41>", line 1
        x = [-1.1 0.0 3.6 -7.2] ^
        SyntaxError: invalid syntax
```

Here Python returns a SyntaxError suggesting the expression that you entered does not make sense in Python.

Another common way to represent vectors in Python is to use a numpy array. To do so, we must first import the numpy package.

```
In [ ]: import numpy as np
        x = np.array([-1.1, 0.0, 3.6, -7.2])
        len(x)
```

```
Out[ ]: 4
```

We can initialize numpy arrays from Python list. Here, to call the package we put np. in front of the array structure.

Note: we have parentheses outside the square brackets. A major advantage of numpy arrays is their ability to perform linear algebra operations which make intuitive sense when we are working with vectors.

2.2 Indexing

A specific element x_j is extracted with the expression $x[j]$ where j is the index (which runs from 0 to $n - 1$, for an n -vector).

```
In [ ] : import numpy as np
```

```
x = np.array([-1.1, 0.0, 3.6, -7.2])
x[2]
```

```
Out[ ] : 3.6
```

If we used array indexing on the left-hand side of an assignment statement, then the value of the corresponding element changes.

```
In [ ]: x[2] = 20.0
```

```
print(x)
```

```
[-1.1 0. 20. -7.2]
```

`-1` is a special index in Python. It is the index of the last element in an array.

```
In [ ] : x[-1]
```

```
Out[ ] : -7.2
```

2.3 Assignment versus copying

The behavior of an assignment statement `y = x` where `x` is an array may be surprising for those who use other languages like Matlab or Octave. The assignment expression gives a new name `y` to the same array that is already referenced by `x` instead of creating a copy of `x`.

```
In [ ] : import numpy as np
```

```
x = np.array([-1.1, 0.0, 3.6, -7.2])
```

```
y = x
```

```
x[2] = 20.0
```

```
print(y)
```

```
[-1.1 0. 20. -7.2]
```

To create a new copy of array `x`, the method `copy` should be used.

```
In [ ] : import numpy as np
```

```
x = np.array([-1.1, 0.0, 3.6, -7.2])
```

```
y = x.copy()
```

```
x[2] = 20.0
```

```
print(y)
```

```
[-1.1 0. 3.6 -7.2]
```

2.4 Zero vectors

We can create a zero vector of size `n` using `np.zeros(n)`. The expression `np.zeros(len(x))` creates a zero vector with the same dimension as vector `x`.

```
In [ ] : import numpy as np
```

```
np.zeros(3)
```

```
Out[ ] : array([0., 0., 0.])
```

Unit vectors

We can create the i th unit vector of length n using index.

```
In [ ] : import numpy as np
```

```
        i = 2
```

```
        n = 4
```

```
        x = np.zeros(n)
```

```
        x[i] = 1
```

```
        print(x)
```

```
        [0. 0. 1. 0.]
```

Ones vectors

We can create a ones vector of size n using `np.ones(n)`. The expression `np.ones(len(x))` creates a ones vector with the same dimension as vector x .

```
In [ ] : import numpy as np
```

```
        np.ones(3)
```

```
Out[ ] : array([1., 1., 1.])
```

Random vectors

Sometimes it is useful to generate random vectors to check our algorithm or test an identity. In python, we generate a random vector of size n using `np.random.random(n)`.

```
In [ ] : np.random.random(2)
```

```
Out[ ] : array([0.79339885, 0.60803751])
```

2.5 Vector addition

Vector addition and subtraction

If x and y are numpy arrays of the same size, $x+y$ and $x-y$ give their sum and difference, respectively.

```
In [ ] : import numpy as np
```

```
        x = np.array([1,2,3])
```

```
        y = np.array([100,200,300])
```

```
        print('Sum of arrays:', x+y)
```

```
        print('Difference of arrays:', x-y)
```

```
        Sum of arrays: [101 202 303]
```

```
        Difference of arrays: [ -99 -198 -297]
```

Sometimes when we would like to print more than one value, we may add a piece of string in front of the value, followed by a comma. This allows us to distinguish between the values are we printing.

2.6 Scalar-vector multiplication

Scalar-vector multiplication and division If a is a number and x is a numpy array (vector), you can express the scalar-vector product either as $a*x$ or $x*a$.

```
In [ ] : import numpy as np
        x = np.array([1,2,3])
        print(2.2*x)
```

```
Out[ ] : [2.2 4.4 6.6]
```

You can carry out scalar-vector division as x/a .

```
In [ ] : import numpy as np
        x = np.array([1,2,3])
        print(x/2.2)
        [0.45454545 0.90909091 1.36363636]
```

Remark: For Python 2.x, integer division is used when you use the operator $/$ on scalars. For example, $5/2$ gives you 2. You can avoid this problem by adding decimals to the integer, i.e., $5.0/2$. This gives you 2.5.

Scalar-vector addition. In Python, you can add a scalar a and a numpy array (vector) x using $x+a$. This means that the scalar is added to each element of the vector. This is, however, NOT a standard mathematical notation. In mathematical notations, we should denote this as, e.g. $x + a\mathbf{1}$, where x is an n -vector and a is a scalar.

```
In [ ] : import numpy as np
        x = np.array([1,2,3,4])
        print(x + 2)
        [3 4 5 6]
```

2.7 Checking properties

Let's check the distributive property

$$\beta(a + b) = \beta a + \beta b$$

which holds for any two n -vector a and b , and any scalar β . We'll do this for $n = 3$, and randomly generated a , b , and β . (This computation does not show that the property always holds; it only show that it holds for the specific vectors chosen. But it's good to be skeptical and check identities with random arguments.) We use the `lincomb` function we just defined.

```

In [ ] : import numpy as np
        a = np.random.random(3)
        b = np.random.random(3)
        beta = np.random.random()
        lhs = beta*(a+b)
        rhs = beta*a + beta*b
        print('a :', a)
        print('b :', b)
        print('beta :', beta)
        print('LHS :', lhs)
        print('RHS :', rhs)

Out[ ] : a : [0.81037789 0.423708 0.76037206]
        b : [0.45712264 0.73141297 0.46341656]
        beta : 0.5799757967698047
        LHS : [0.73511963 0.6699422 0.70976778]
        RHS : [0.73511963 0.6699422 0.70976778]

```

Although the two vectors lhs and rhs are displayed as the same, they might not be exactly the same, due to very small round-off errors in floating point computations. When we check an identity using random numbers, we can expect that the left-hand and right-hand sides of the identity are not exactly the same, but very close to each other.

2.8 Inner product

The inner product of n -vector x and y is denoted as $x^T y$. In Python the inner product of x and y can be found using `np.inner(x,y)`

```

In [ ] : import numpy as np
        x = np.array([-1,2,2])
        y = np.array([1,0,-3])
        print(np.inner(x,y))

```

```

Out[ ] : -7

```

Alternatively, you can use the `@` operator to perform inner product on numpy arrays.

```

In [ ] : import numpy as np
        x = np.array([-1,2,2])
        y = np.array([1,0,-3])
        x @ y

```


Out[] : -7

Net present value. As an example, the following code snippet finds the net present value (NPV) of a cash flow vector c , with per-period interest rate r .

```
In [ ] : import numpy as np
        c = np.array([0.1,0.1,0.1,1.1]) #cash flow vector
        n = len(c)
        r = 0.05 #5% per-period interest rate
        d = np.array([(1+r)**-i for i in range(n)])
        NPV = c @ d
        print(NPV)
        1.236162401468524
```

In the fifth line, to get the vector d we raise the scalar $1+r$ element-wise to the powers given in the range $\text{range}(n)$ which expands to $0, 1, 2, \dots, n-1$, using list comprehension.

2.9 Complexity of vector computations

Floating point operations. For any two numbers a and b , we have $(a+b)(a-b) = a^2 - b^2$. When a computer calculates the left-hand and right-hand side, for specific numbers a and b , they need not be exactly the same, due to very small floating point round-off errors. But they should be very nearly the same. Let's see an example of this.

```
In [ ] : import numpy as np
        a = np.random.random()
        b = np.random.random()
        lhs = (a+b) * (a-b)
        rhs = a**2 - b**2
        print(lhs - rhs)
        4.336808689942018e-19
```

Here we see that the left-hand and right-hand sides are not exactly equal, but very very close.

Complexity. You can time a Python command using the `time` package. The timer is not very accurate for very small times, say, measured in microseconds (10^{-6} seconds). You should run the command more than once; it can be a lot faster on the second or subsequent runs.

```
In [ ] : import numpy as np
        import time
        a = np.random.random(10**5)
        b = np.random.random(10**5)
```

Out[] : -7

Net present value. As an example, the following code snippet finds the net present value (NPV) of a cash flow vector *c*, with per-period interest rate *r*.

```
In [ ] : import numpy as np
        c = np.array([0.1,0.1,0.1,1.1]) #cash flow vector
        n = len(c)
        r = 0.05 #5% per-period interest rate
        d = np.array([(1+r)**-i for i in range(n)])
        NPV = c @ d
        print(NPV)
1.236162401468524
```

In the fifth line, to get the vector *d* we raise the scalar $1+r$ element-wise to the powers given in the range `range(n)` which expands to 0, 1, 2, ..., $n-1$, using list comprehension.

2.9 Complexity of vector computations

Floating point operations. For any two numbers *a* and *b*, we have $(a+b)(a-b) = a^2 - b^2$. When a computer calculates the left-hand and right-hand side, for specific numbers *a* and *b*, they need not be exactly the same, due to very small floating point round-off errors. But they should be very nearly the same. Let's see an example of this.

```
In [ ] : import numpy as np
        a = np.random.random()
        b = np.random.random()
        lhs = (a+b) * (a-b)
        rhs = a**2 - b**2
        print(lhs - rhs)
4.336808689942018e-19
```

Here we see that the left-hand and right-hand sides are not exactly equal, but very very close.

Complexity. You can time a Python command using the `time` package. The timer is not very accurate for very small times, say, measured in microseconds (10^{-6} seconds). You should run the command more than once; it can be a lot faster on the second or subsequent runs.

```
In [ ] : import numpy as np
        import time
        a = np.random.random(10**5)
        b = np.random.random(10**5)
```


3.ALGEBRA AND SYMBOLIC MATH WITH SYMPY

3.1 Defining symbols and operations

Symbols form the building blocks of symbolic math. The term symbol is just a general name for the x s, y s, z s, and b s you use in equations and algebraic expressions. Creating and using symbols will let us do things differently than before. Consider the following statements:

```
>>>x=1
>>>x + x + 1
3
```

Here we create a label, x , to refer to the number 1. Then, when we write the statement $x + x + 1$, it's evaluated for us, and the result is 3. What if you wanted the result in terms of the symbol x ? That is, if instead of 3, you wanted Python to tell you that the result is $2x + 1$? You couldn't just write $x + x + 1$ without the statement $x = 1$ because Python wouldn't know what x refers to.

let us write programs where we can express and evaluate mathematical expressions in terms of such symbols. To use a symbol in your program, you have to create an object of the Symbol class, like this:

```
>>>from sympy import Symbol
>>>x = Symbol('x')
```

First, we import the Symbol class from the sympy library. Then, we create an object of this class passing 'x' as a parameter. Note that this 'x' is written as a string within quotes. We can now define expressions and equations in terms of this symbol. For example, here's the earlier expression:

```
>>> from sympy import Symbol
>>> x = Symbol('x')
>>> x + x + 1
2*x + 1
```

Now the result is given in terms of the symbol x . In the statement $x = \text{Symbol}('x')$, the x on the left side is the Python label. This is the same kind of label we've used before, except this time it refers to the symbol x instead of a number more specifically, a Symbol object representing the symbol 'x'. This label doesn't necessarily have to match the

symbol either we could have used a label like `a` or `var1` instead. So, it's perfectly fine to write the preceding statements as follows:

```
>>> a = Symbol('x')
```

```
>>> a + a + 1
```

```
2*x + 1
```

Now that we know how to define our own symbolic expressions, let's learn more about using them in our programs.

3.2 Factorizing and Expanding Expressions

The `factor()` function decomposes an expression into its factors, and the `expand()` function expands an expression, expressing it as a sum of individual terms. Let's test out these functions with the basic algebraic identity $x^2 - y^2 = (x + y)(x - y)$. The left side of the identity is the expanded version, and the right side depicts the corresponding factorization. Because we have two symbols in the identity, we'll create two `Symbol` objects:

```
>>> from sympy import Symbol
```

```
>>> x = Symbol('x')
```

```
>>> y = Symbol('y')
```

Next, we import the `factor()` function and use it to convert the expanded version (on the left side of the identity) to the factored version (on the right side):

```
>>> from sympy import factor
```

```
>>> expr = x**2 - y**2
```

```
>>> factor(expr)
```

```
(x - y)*(x + y)
```

As expected, we get the factored version of the expression. Now let's expand the factors to get back the original expanded version:

```
>>> factors = factor(expr)
```

```
>>> expand(factors)
```

```
x**2 - y**2
```

We store the factorized expression in a new label, `factors`, and then call the `expand()` function with it. When we do this, we receive the original expression we started with. Let's try it with the more complicated identity $x^3 + 3x^2y + 3xy^2 + y^3 = (x + y)^3$:

```
>>> expr = x**3 + 3*x**2*y + 3*x*y**2 + y**3
```

```
>>> factors = factor(expr)
```

```
>>> factors (x + y)**3
```

```
>>> expand(factors)
```

```
x**3 + 3*x**2*y + 3*x*y**2 + y**3
```

The `factor()` function is able to factorize the expression, and then the `expand()` function expands the factorized expression to return to the original expression.

If you try to factorize an expression for which there's no possible factorization, the original expression is returned by the `factor()` function. For example, see the following:

```
>>> expr = x + y + x*y
```

```
>>> factor(expr)
```

```
x*y + x + y
```

3.3 Pretty Printing

If you want the expressions we've been working with to look a bit nicer when you print them, you can use the `pprint()` function. This function will print the expression in a way that more closely resembles how we'd normally write it on paper. For example, here's an expression:

```
>>> expr = x*x + 2*x*y + y*y
```

If we print it as we've been doing so far or use the `print()` function, this is how it looks:

```
>>> expr x**2 + 2*x*y + y**2
```

Now, let's use the `pprint()` function to print the preceding expression:

```
>>> from sympy import pprint
```

```
>>> pprint(expr)
```

```
x2 + 2·x·y + y2
```

The expression now looks much cleaner—for example, instead of having a bunch of ugly asterisks, exponents appear above the rest of the numbers.

You can also change the order of the terms when you print an expression.

Consider the expression $1 + 2x + 2x^2$:

```
>>> expr = 1 + 2*x + 2*x**2
```

```
>>> pprint(expr)
```

```
2·x2 + 2·x + 1
```

The terms are arranged in the order of powers of x , from highest to lowest.

If you want the expression in the opposite order, with the highest power of x last, you can make that happen with the `init_printing()` function, as follows:

```
>>> from sympy import init_printing
>>> init_printing(order='rev-lex')
>>> pprint(expr)
1 + 2·x + 2·x2
```

The `init_printing()` function is first imported and called with the keyword argument `order='rev-lex'`. This indicates that we want SymPy to print the expressions so that they're in reverse lexicographical order. In this case, the keyword argument tells Python to print the lower-power terms first.

3.4 Substituting in Values

Let's see how we can use SymPy to plug values into an algebraic expression. This will let us calculate the value of the expression for certain values of the variables. Consider the mathematical expression $x^2 + 2xy + y^2$, which can be defined as follows:

```
>>> x = Symbol('x')
>>> y = Symbol('y')
>>> x*x + x*y + x*y + y*y x**2 + 2*x*y + y**2
```

If you want to evaluate this expression, you can substitute numbers in for the symbols using the `subs()` method:

```
>>> expr = x*x + x*y + x*y + y*y
>>> res = expr.subs({x:1, y:2})
```

First, we create a new label to refer to the expression and then we call the `subs()` method. The argument to the `subs()` method is a Python dictionary, which contains the two symbol labels and the numerical values we want to substitute in for each symbol. Let's check out the result:

```
>>> res
9
```

You can also express one symbol in terms of another and substitute accordingly, using the `subs()` method. For example, if you knew that $x = 1 - y$, here's how you could evaluate the preceding expression:

```
>>> expr.subs({x:1-y})
y**2 + 2*y*(-y + 1) + (-y + 1)**2
```

If you want the result to be simplified further for example, if there are terms that cancel each other out, we can use SymPy's `simplify()` function, as follows:


```
>>> expr_subs = expr.subs({x:1-y})
>>> from sympy import simplify
>>> simplify(expr_subs)
```

```
1
```

we create a new label, `expr_subs`, to refer to the result of substituting $x = 1 - y$ in the expression. We then import the `simplify()` function from SymPy and call it. The result turns out to be 1 because the other terms of the expression cancel each other. Although there was a simplified version of the expression in the preceding example, you had to ask SymPy to simplify it using the `simplify()` function. Once again, this is because SymPy won't do any simplification without being asked to. The `simplify()` function can also simplify complicated expressions, such as those including logarithms and trigonometric functions, but we won't get into that here.

3.5 Solving Equations

SymPy's `solve()` function can be used to find solutions to equations.

When you input an expression with a symbol representing a variable, such as x , `solve()` calculates the value of that symbol. This function always makes its calculation by assuming the expression you enter is equal to zero—that is, it prints the value that, when substituted for the symbol, makes the entire expression equal zero. Let's start with the simple equation $x - 5 = 7$. If we want to use `solve()` to find the value of x , we first have to make one side of the equation equal zero ($x - 5 - 7 = 0$). Then, we're ready to use `solve()`, as follows:

```
>>> from sympy import Symbol, solve
```

```
>>> x = Symbol('x')
```

```
>>> expr = x - 5 - 7
```

```
>>> solve(expr)
```

```
[12]
```

When we use `solve()`, it calculates the value of 'x' as 12 because that's the value that makes the expression ($x - 5 - 7$) equal to zero.

Note that the result 12 is returned in a list. An equation can have multiple solutions for example, a quadratic equation has two solutions. In that case, the list will have all the solutions as its members. You can also ask the `solve()` function to return the result so that each member is dictionary instead. Each dictionary is composed of the symbol (variable name) and its value (the solution). This is especially useful when solving simultaneous equations

where we have more than one variable to solve for because when the solution is returned as a dictionary, we know which solution corresponds to which variable.

3.5.1 Solving Quadratic Equations

Now, we'll learn how we can use SymPy's `solve()` function to find the roots without needing to write out the formulas. Let's see an example:

```
1. >>> from sympy import solve
2. >>> x = Symbol('x')
3. >>> expr = x**2 + 5*x + 4
4. >>> solve(expr, dict=True)
5. [{x: -4}, {x: -1}]
```

The `solve()` function is first imported at 1. We then define a symbol, `x`, and an expression corresponding to the quadratic equation, $x^2 + 5x + 4$, at 3. Then, we call the `solve()` function with the preceding expression at 4. The second argument to the `solve()` function (`dict=True`) specifies that we want the result to be returned as a list of Python dictionaries.

Each solution in the returned list is a dictionary using the symbol as a key matched with its corresponding value. If the solution is empty, an empty list will be returned. The roots of the preceding equation are -4 and -1 , as you can see at 5. The roots of the equation $x^2 + x + 1 = 0$ are complex numbers. Let's attempt to find those using `solve()`:

```
>>> x=Symbol('x')
>>> expr = x**2 + x + 1
>>> solve(expr, dict=True)
[{x: -1/2 - sqrt(3)*I/2}, {x: -1/2 + sqrt(3)*I/2}]
```

Both the roots are imaginary, as expected with the imaginary component indicated by the `I` symbol.

3.6 Solving a system of linear equations

Consider the following two equations

$$2x + 3y = 6$$

$$3x + 2y = 12$$

Say we want to find the pair of values (x, y) that satisfies both the equations.

We can use the `solve()` function to find the solution for a system of equations like this one.

First, we define the two symbols and create the two equations:

```
>>> x = Symbol('x')
```

```
>>> y = Symbol('y')
>>> expr1 = 2*x + 3*y - 6
>>> expr2 = 3*x + 2*y - 12
```

The two equations are defined by the expressions `expr1` and `expr2`, respectively. Note how we've rearranged the expressions so they both equal zero (we moved the right side of the given equations to the left side). To find the solution, we call the `solve()` function with the two expressions forming a tuple:

```
>>> solve((expr1, expr2), dict=True)
[{y: -6/5, x: 24/5}]
```

As mentioned earlier, getting the solution back as a dictionary is useful here. We can see that the value of `x` is `24/5` and the value of `y` is `-6/5`. Let's verify whether the solution we got really satisfies the equations. To do so, we'll first create a label, `soln`, to refer to the solution we got and then use the `subs()` method to substitute the corresponding values of `x` and `y` in the two expressions:

```
>>> soln = solve((expr1, expr2), dict=True)
>>> soln = soln[0]
>>> expr1.subs({x:soln[x], y:soln[y]})
0
>>> expr2.subs({x:soln[x], y:soln[y]})
0
```

The result of substituting the values of `x` and `y` corresponding to the solution in the two expressions is zero.

4. MATRICES

4.1 Matrices

4.1.1 Creating matrices from the entries

Matrices can be represented in Python using 2-dimensional numpy array or list structure (list of lists). For example, the 3x4 matrix

$$A = \begin{pmatrix} 0 & 1 & 4 & 0.1 \\ 1.3 & 4 & 6 & 1 \\ 4.1 & 8 & 0 & 2 \end{pmatrix}$$

is constructed in Python as

```
In [ ]: A = np.array([[0,1,4,0.1], [1.3, 4, 6, 1], [4.1, 8,0, 2]])
        A.shape
```

```
Out [ ]: (3, 4)
```

4.1.2 Indexing entries

We get the i th row j th column entry of a matrix A using $A[i-1, j-1]$ since Python starts indexing from 0.

```
In [ ]: A[0,2] #Get the third element in the first row
```

```
Out[ ]: 4
```

4.1.3 Equality of matrices

$A == B$ determines whether the matrices A and B are equal. For numpy arrays A and B , the expression $A == B$ creates a matrix whose entries are Boolean, depending on whether the corresponding entries of A and B are the same. The expression $\text{sum}(A == B)$ gives the number of entries of A and B that are equal.

```
In [ ]: A = np.array ([0,1,4,0.1], [1.3,4,6,1], [4.1,8,0,2]))
```

```
B = A.copy()
```

```
A == B
```

```
Out[ ]: array([[ True,  True,  True,  True],
               [ True,  True,  True,  True],
               [ True,  True,  True,  True]])
```

4.1.4 Row and column vectors

In Python, n -vectors are the same as $n \times 1$ matrices, there is a subtle difference between a 1D array, a column vector and a row vector.


```
Out[ ]: array([[1., 0., 0., 0.],
               [0., 1., 0., 0.],
               [0., 0., 1., 0.],
               [0., 0., 0., 1.]])
```

Diagonal matrices

In standard mathematical notation, $\text{diag}(1; 2; 3)$ is a diagonal 3×3 matrix with diagonal entries 1; 2; 3. In Python, such a matrix can be created using `np.diag()`.

```
In [ ]: x = np.array([[0, 1, 2],
                      [3, 4, 5],
                      [6, 7, 8]])

print(np.diag(x))
```

```
Out [ ]: [0 4 8]
```

4.3 Transpose and addition

Transpose

In Python, the transpose of A is given by `np.transpose(A)` or simply `A.T`

```
In [ ]: H = np.array([[0,1,-2,1], [2,-1,3,0]])
H.T
```

```
Out[ ]: array([[ 0, 2],
               [ 1, -1],
               [-2, 3],
               [ 1, 0]])
```

Addition, subtraction, and scalar multiplication

In Python, addition and subtraction of matrices, and scalar-matrix multiplication, both follow standard and mathematical notation.

```
In [ ]: U = np.array([[0,4], [7,0], [3,1]])
V = np.array([[1,2], [2,3], [0,4]])
U + V
```

```
Out[ ]: array([[1, 6],
               [9, 3],
               [3, 5]])
```

```
In [ ]: 2.2*U
```

```
Out[ ]: array([[0. , 8.8],
```

```
[15.4, 0. ],  
[6.6 , 2.2]])
```

4.4 Matrix – matrix multiplication

In Python the product of matrices A and B is obtained with A @ B. Alternatively, we can compute the matrix product using the function np.matmul().

```
In [ ]: A = np.array([[ -1.5, 3, 2], [1, -1, 0]]) #2 by 3 matrix  
        B = np.array([[ -1, -1], [0, -2], [1, 0]]) #3 by 2 matrix  
        C = A @ B  
        print(C)
```

```
Out[ ]: [[ 3.5 -4.5]]
```

4.5 Determinants

A special number that can be calculated from a square matrix is known as the Determinant of a square matrix. The Numpy provides us the feature to calculate the determinant of a square matrix using numpy.linalg.det() function.

Syntax

```
numpy.linalg.det(array)
```

Example 1

Calculating Determinant of a 2X2 Numpy matrix using numpy.linalg.det()

function.

```
# importing Numpy package  
import numpy as np  
# creating a 2X2 Numpy matrix  
n_array = np.array([[50, 29], [30, 44]])  
# Displaying the Matrix  
print("Numpy Matrix is:")  
print(n_array)  
# calculating the determinant of matrix  
det = np.linalg.det(n_array)  
print("\nDeterminant of given 2X2 matrix:")  
print(int(det))
```

Output

```
Numpy Matrix is:[[50 29]  
                 [30 44]]
```

Determinant of given 2x2 matrix: 1330

In the above example, we calculate the Determinant of the 2X2 square matrix.

Example 2

Calculating Determinant of a 3X3 Numpy matrix using `numpy.linalg.det()` function

```
#importing Numpy package
```

```
import numpy as np
```

```
#creating a3x3 numpy matrix
```

```
_array = np.array([[55, 25, 15], [30, 44, 2], [11, 45, 77]])
```

```
# Displaying the Matrix
```

```
print("Numpy Matrix is:")
```

```
print(n_array)
```

```
# calculating the determinant of matrix
```

```
det = np.linalg.det(n_array)
```

```
print("\nDeterminant of given 3X3 square matrix:")
```

```
print(int(det))
```

Output

```
Numpy matrix is:[[55 25 15]
                  [30 44  2]
                  [11 45 77]]
```

Determinant of given 3x3 square matrix: 137180

In the above example, we calculate the Determinant of the 3X3 square matrix.

Example 3

Calculating Determinant of a 5X5 Numpy matrix using `numpy.linalg.det()` function

```
# importing Numpy package
```

```
import numpy as np
```

```
# importing Numpy package
```

```
import numpy as np
```

```
# creating a 5X5 Numpy matrix
```

```
n_array = np.array([5, 2, 1, 4, 6],
                    [9, 4, 2, 5, 2],
```

```
[11, 5, 7, 3, 9],
[5, 6, 6, 7, 2],
[7, 5, 9, 3, 3]])
```

```
# Displaying the Matrix
```

```
print("Numpy Matrix is:")
```

```
print(n_array)
```

```
# calculating the determinant of matrix
```

```
det = np.linalg.det(n_array)
```

```
print("\nDeterminant of given 5X5 square matrix:")
```

```
print(int(det))
```

Output

```
Numpy matrix is :[[5 2 1 4 6]
                  [9 4 2 5 2]
                  [11 5 7 3 9]
                  [5 6 6 7 2]
                  [7 5 9 3 3]]
```

```
Determinant of given 5x5 square matrix: -2003
```

4.6 Left and right inverses

```
In [ ]: A = np.array([[ -3,-4], [4,6], [1,1]])
```

```
B = np.array([[ -11,-10,16], [7,8,-11]])/9 #left inverse of A
```

```
C = np.array([[0,-1,6], [0,1,-4]])/2 #Another left inverse of A
```

```
#Lets check
```

```
B @ A
```

```
Out[ ]: array([[ 1.00000000e+00, 0.00000000e+00],
               [-4.4408921e-16, 1.00000000e+00]])
```

```
In [ ]: C @ A
```

```
Out[ ]: array([[1., 0.],
               [0., 1.]])
```

Inverse

If A is invertible, its inverse is given by `np.linalg.inv(A)`. You'll get an error if A is not invertible, or not square

```
In [ ]: A = np.array([[1,-2,3], [0,2,2], [-4,-4, -4]])
```

5.PLAYING WITH SETS

5.1 What's a set?

A set is a collection of distinct objects , often called elements or members. Two characteristics of a set make it different from just any collection of objects. A set is “well defined,” meaning the question “Is a particular object in this collection?” always has a clear yes or no answer, usually based on a rule or some given criteria. The second characteristic is that no two members of a set are the same. A set can contain anything—numbers, people, things, words , and so on. Let's walk through some basic properties of sets as we learn how to work with sets in Python using SymPy.

5.2 Set construction

In mathematical notation, you represent a set by writing the set members enclosed in curly brackets. For example, $\{2, 4, 6\}$ represents a set with 2, 4, and 6 as its members. To create a set in Python, we can use the `FiniteSet` class from the `sympy` package, as follows:

```
>>> from sympy import FiniteSet
```

```
>>> s = FiniteSet(2, 4, 6)
```

```
>>> s
```

```
{2, 4, 6}
```

Here, we first import the `FiniteSet` class from `SymPy` and then create an object of this class by passing in the set members as arguments. We assign the label `s` to the set we just created. We can store different types of numbers—including integers, floating point numbers, and fractions—in the same set:

```
>>> from sympy import FiniteSet
```

```
>>> from fractions import Fraction
```

```
>>> s = FiniteSet(1, 1.5, Fraction(1, 5))
```

```
>>> s
```

```
{1/5, 1, 1.5}
```

The cardinality of a set is the number of members in the set, which you can find by using the `len()` function:

```
>>> s = FiniteSet(1, 1.5, 3)
```

```
>>> len(s)
```


has multiple instances of a number, the number is added to the set only once, and the other instances are discarded:

```
>>> from sympy import FiniteSet
>>> members = [1, 2, 3, 2]
>>> FiniteSet(*members)
{1, 2, 3}
```

Here, even though we passed in a list that had two instances of the number 2, the number 2 appears only once in the set created from that list. In Python lists and tuples, each element is stored in a particular order, but the same is not always true for sets. For example, we can print out each member of a set by iterating through it as follows:

```
>>> from sympy import FiniteSet
>>> s = FiniteSet(1, 2, 3)
>>> for member in s:
print(member)
2
1
3
```

When you run this code, the elements could be printed in any possible order. This is because of how sets are stored by Python—it keeps track of what members are in the set, but it doesn't keep track of any particular order for those members. Let's see another example. Two sets are equal when they have the same elements. In Python, you can use the equality operator, `==`, to check whether two sets are equal:

```
>>> from sympy import FiniteSet
>>> s = FiniteSet(3, 4, 5)
>>> t = FiniteSet(5, 4, 3)
>>> s == t
True
```

Although the members of these two sets appear in different orders, the sets are still equal.

5.3 Subsets, Supersets, and Power Sets

A set, s , is a subset of another set, t , if all the members of s are also members of t . For example, the set $\{1\}$ is a subset of the set $\{1, 2\}$. You can check whether a set is a subset of another set using the `is_subset()` method:

```
>>> s = FiniteSet(1)
>>> t = FiniteSet(1,2)
>>> s.is_subset(t)
True
>>> t.is_subset(s)
```

False

Note that an empty set is a subset of every set. Also, any set is a subset of itself, as you can see in the following:

```
>>> s.is_subset(s)
True
>>> t.is_subset(t)
```

True

Similarly, a set, t , is said to be a superset of another set, s , if t contains all of the members contained in s . You can check whether one set is a superset of another using the `is_superset()` method:

```
>>> s.is_superset(t)
False
>>> t.is_superset(s)
```

True

The power set of a set, s , is the set of all possible subsets of s . Any set, s , has precisely $2^{|s|}$ subsets, where $|s|$ is the cardinality of the set. For example, the set $\{1, 2, 3\}$ has a cardinality of 3, so it has 2^3 or 8 subsets: $\{\}$ (the empty set), $\{1\}$, $\{2\}$, $\{3\}$, $\{1, 2\}$, $\{2, 3\}$, $\{1, 3\}$, and $\{1, 2, 3\}$. The set of all these subsets form the power set, and we can find the power set using the `powerset()` method:

```
>>> s = FiniteSet(1, 2, 3)
>>> ps = s.powerset()
>>> ps
```

```
{1}, {1, 2}, {1, 3}, {1, 2, 3}, {2}, {2, 3}, {3}, EmptySet()
```

As the power set is a set itself, you can find its cardinality using the `len()` function:

```
>>> len(ps)
8
```

The cardinality of the power set is $2^{|s|}$, which is $2^3 = 8$.

Based on our definition of a subset, any two sets with the exact same members would be subsets as well as supersets of each other. By contrast, a set, s , is a proper subset of t only if all the members of s are also in t and t has at least one member that is not in s . So if $s = \{1, 2, 3\}$, it's only a proper subset of t if t contains 1, 2, and 3 plus at least one more member. This would also mean that t is a proper superset of s . You can use the `is_proper_subset()` method and the `is_proper_superset()` method to check for these relationships:

```
>>> from sympy import FiniteSet
```

```
>>> s = FiniteSet(1, 2, 3)
```

```
>>> t = FiniteSet(1, 2, 3)
```

```
>>> s.is_proper_subset(t)
```

```
False
```

```
>>> t.is_proper_superset(s)
```

```
False
```

Now, if we re-create the set t to include another member, s will be considered a proper subset of t and t a proper superset of s :

```
>>> t = FiniteSet(1, 2, 3, 4)
```

```
>>> s.is_proper_subset(t)
```

```
True
```

```
>>> t.is_proper_superset(s)
```

```
True
```

5.4 Set Operations

Set operations such as union, intersection, and the Cartesian product allow you to combine sets in certain methodical ways. These set operations are extremely useful in real-world problem-solving situations when we have to consider multiple sets together. Later in this chapter, we'll see how to use these operations to apply a formula to multiple sets of data and calculate the probabilities of random events.

5.4.1 Union and Intersection

The union of two sets is a set that contains all of the distinct members of the two sets. In set theory, we use the symbol U to refer to the union operation. For example, $\{1, 2\} U \{2, 3\}$ will result in a new set, $\{1, 2, 3\}$. In SymPy, the union of these two sets can be created using the `union()` method:

```
>>> from sympy import FiniteSet
```


5.4.2 Cartesian Product

The Cartesian product of two sets creates a set that consists of all possible pairs made by taking an element from each set. For example, the Cartesian product of the sets {1, 2} and {3, 4} is {(1, 3), (1, 4), (2, 3), (2, 4)}. In SymPy, you can find the Cartesian product of two sets by simply using the multiplication operator:

```
>>> from sympy import FiniteSet
```

```
>>> s = FiniteSet(1, 2)
```

```
>>> t = FiniteSet(3, 4)
```

```
>>> p = s*t
```

```
>>> p
```

```
{1, 2} x {3, 4}
```

This takes the Cartesian product of the sets *s* and *t* and stores it as *p*. To actually see each pair in that Cartesian product, we can iterate through and print them out as follows:

```
>>> for elem in p:
```

```
    print(elem)
```

```
(1, 3)
```

```
(1, 4)
```

```
(2, 3)
```

```
(2, 4)
```

Each element of the product is a tuple consisting of a member from the first set and a member from the second set.

The cardinality of the Cartesian product is the product of the cardinality of the individual sets.

We can demonstrate this in Python:

```
>>> len(p) == len(s)*len(t)
```

```
True
```

APPLICATIONS OF PYTHON

We are living in a digital world that is completely driven by chunks of code. Every industry depends on software for its proper functioning be it healthcare, military, banking, research, and the list goes on. We have a huge list of programming languages that facilitate the software development process. One of these is Python which has emerged as the most lucrative and exciting programming language. As per a survey it is observed that python is the main coding language for more than 80% of developers. The main reason behind this is its extensive libraries and frameworks that fuel up the process.

Python has been at the forefront of Machine learning, data science, and artificial intelligence innovation. Further, it provides ease in building a plethora of applications, web development processes, and a lot more. In this blog, we will discuss the Python applications in a detailed manner.

1. DATA SCIENCE

Python is open source, interpreted, high level language and provides great approach for object-oriented programming. It is one of the best language used by data scientist for various data science projects/application. Python provide great functionality to deal with mathematics, statistics and scientific function. It provides great libraries to deals with data science application.

One of the main reasons why Python is widely used in the scientific and research communities is because of its ease of use and simple syntax which makes it easy to adapt for people who do not have an engineering background. It is also more suited for quick prototyping.

According to engineers coming from academia and industry, deep learning frameworks available with Python APIs, in addition to the scientific packages have made Python incredibly productive and versatile. There has been a lot of evolution in deep learning Python frameworks and it's rapidly upgrading.

In terms of application areas, ML scientists prefer Python as well. When it comes to areas like building fraud detection algorithms and network security, developers leaned towards Java, while for applications like natural language processing (NLP) and sentiment analysis, developers opted for Python, because it provides large collection of libraries that help to solve complex business problem easily, build strong system and data application.

2. Machine Learning and Artificial Intelligence

Machine Learning and Artificial Intelligence are the hottest subjects right now. Python along with its inbuilt libraries and tools facilitate the development of AI and ML algorithms. Further, it offers simple, concise, and readable code which makes it easier for developers to write complex algorithms and provide a versatile flow. Some of the inbuilt libraries and tools that enhance AI and ML processes are:

- Numpy for complex data analysis
- Keras for Machine learning
- SciPy for technical computing
- Seaborn for data visualization

3. Web Development

It is one of the most astonishing applications of Python. This is because Python comes up with a wide range of frameworks like Django, Flask, Bottle, and a lot more that provide ease to developers. Furthermore, Python has inbuilt libraries and tools which make the web development process completely effortless. Use of Python for web development also offers:

- Amazing visualization
- Convenience in development
- Enhanced security
- Fast development process

4. Game Development

With the rapidly growing gaming industry Python has proved to be an exceptional option for game development. Popular games like Pirates of the Caribbean, Bridge commander, and Battlefield 2 use Python programming for a wide range of functionalities and addons. The presence of popular 2D and 3D gaming libraries like pygame, panda3D, and Cocos2D make the game development process completely effortless.

5. Software Development

Python is just the perfect option for software development. Popular applications like Google, Netflix, and Reddit all use Python. This language offers amazing features like:

- Platform independence
- Inbuilt libraries and frameworks to provide ease of development.
- Enhanced code reusability and readability
- High compatibility

Apart from these Python offers enhanced features to work with rapidly growing technologies like Machine learning and Artificial intelligence. All these embedded features make it a popular choice for software development.

Following are some useful features of Python language

- It uses the elegant syntax, hence the programs are easier to read.
- It is a simple to access language, which makes it easy to achieve the program working.
- The large standard library and community support.
- The interactive mode of Python makes its simple to test codes.
- In Python, it is also simple to extend the code by appending new modules that are implemented in other compiled language like C++ or C.
- Python is an expressive language which is possible to embed into applications to offer a programmable interface.
- Allows developer to run the code anywhere, including Windows, Mac OS X, UNIX, and Linux.
- It is free software in a couple of categories. It does not cost anything to use or download Pythons or to add it to the application.

WHAT IS THE FUTURE OF PYTHON FOR DATA SCIENCE?

As Python continues to grow in popularity and as the number of data scientists continues to increase, the use of Python for data science will inevitably continue to grow. As we advance machine learning, deep learning, and other data science tasks, we'll likely see these advancements available for our use as libraries in Python. Python has been well-maintained and continuously growing in popularity for years, and many of the top companies use Python today. With its continued popularity and growing support, Python will be used in the industry for years to come.

CONCLUSION

Python is a concise and extremely powerful language that is rapidly gaining popularity. It has been the epicenter of most amazing technologies like AI, automation, and machine learning. Further, it is used to facilitate hot subjects like data analysis and data visualization.

Programming language can be a key in your hand for your perfect dream job in the software industry. In today's market, Python has been declared as the most used and **most demanding programming language** of all. As the language is getting dominant in all major fields: Ranging from software development to machine learning and data analytics, Python has been declared as the **language of the year 2018**. Hence It currently occupies **37 percent** of Programming language market.

References

- [1] The Complete Reference Complete Reference Series Martin C. Brown Edition illustrated Osborne/McGraw-Hill, 2001 ISBN 007212718X, 9780072127188
- [2] Automate the Boring Stuff with Python Practical Programming for Total Beginners Al Sweigart Edition illustrated, reprint No Starch Press, 2015 ISBN 1593275994, 9781593275990
- [3] <https://www.studytonight.com/python/>
- [4] <https://www.geeksforgeeks.org/python-operators/>
- [5] <https://www.w3schools.com/python/>

QUEUEING THEORY

Project Report submitted to

ST.MARY'S COLLEGE (AUTONOMOUS), THOOTHUKUDI

Affiliated to

MANONMANIAM SUNDARANAR UNIVERSITY, TIRUNELVELI

In partial fulfilment of the requirement for the award of degree of

Bachelor of Science in Mathematics

Submitted by

NAME

MEGI. B

PRAVEENA. S

RASIKA. T

VARSIGA AROCKIYA FRANCIS. A

VINNARASI. M

REG.NO.

19AUMT26

19AUMT34

19AUMT36

19AUMT48

19AUMT49

Under the Guidance of

Dr. A. PUNITHA THARANI M.Sc., M.Phil., Ph.D.

Associate Professor of Mathematics and COE

St. Mary's College (Autonomous), Thoothukudi.



Department of Mathematics

St. Mary's College (Autonomous), Thoothukudi

(2021 - 2022)

CERTIFICATE

We hereby declare that the project report entitled "QUEUEING THEORY" being submitted to **St. Mary's College (Autonomous), Thoothukudi** affiliated to **Manonmaniam Sundaranar University, Tirunelveli** in partial fulfilment for the award of degree of **Bachelor of Science in Mathematics** and it is a record of work done during the year 2021 - 2022 by the following students:

NAME	REG.NO.
MEGI. B	19AUMT26
PRAVEENA. S	19AUMT34
RASIKA. T	19AUMT36
VARSIGA AROCKIYA FRANCIS. A	19AUMT48
VINNARASI. M	19AUMT49

Signature of the Guide
Dr. A. Punitha Tharani
M.Sc., M.Phil., Ph.D.,
Associate Professor,
Dept. of Mathematics,
St. Mary's College (Autonomous),
Thoothukudi - 628 001.

Signature of the HOD
Dr. V.L. Stella Arumugam
M.Sc., M.Phil.
Head & Asst Professor,
St. Mary's College
Thoothukudi - 628 001.

Signature of the Examiner

Signature of the Principal
Principal
St. Mary's College (Autonomous)
Thoothukudi - 628 001.

DECLARATION

We hereby declare that the project report entitled "**QUEUEING THEORY**", is our original work. It has not been submitted to any university for any degree or diploma.

B. Megi
(MEGI. B)

S. Praveena
(PRAVEENA. S)

T. Rasika
(RASIKA. T)

A. Varsiga
(VARSIGA AROCKIYA FRANCIS. A)

M. Vinnarasi
(VINNARASI. M)

ACKNOWLEDGEMENT

First of all, we thank Lord Almighty for showering his blessings to undergo this project.

With immense pleasure, we register our deep sense of gratitude to our guide **Dr. A. Punitha Tharani M.Sc., M.Phil., Ph.D.** for having imparted necessary guidelines throughout the period of our studies.

We thank our beloved Principal, **Rev. Dr. Sr. A.S.J. Lucia Rose M.Sc., M.Phil., Ph.D., PGDCA** and the Head of the Department, **Dr. V. L. Stella Arputha Mary M.Sc., M.Phil., B.Ed., Ph.D.** for providing us the help to carry out our project work successfully.

Finally, we thank all those who extended their helping hands regarding this project.

CONTENT

1.Introduction	1
2.Definitions	3
3.Notation	3
4.Queueing system	
a)Element of Queueing	4
b)Deterministic Queueing system	9
c)Probability Distribution in queueing system	10
5.Classification of Queueing System	16
6.Definition of transient and study states	16
7.poisson's Queueing system	
a)Model I	17
b)Model II	24
c)Model III	24
d)Model IV	29
e)Model V	32
f)Model VI	35
g)Model VII	37
8. Limitation of Queueing theory	38
9. Application of Queueing theory	39
10. Conclusion	40
11.Reference	41

QUEUING THEORY

INTRODUCTION

Queuing theory is a branch of mathematics that studies and models the act of waiting in lines. This paper will take a brief look in to the formulation of the queuing theory along with examples of the models and applications of their use. The goal of the paper is to provide the reader with enough background in order to properly model a basic queuing system in to one of the categories we will look at ,when possible. Also, the reader should begin to understand the basic ideas of how to determine use full information such as average waiting times from a particular queuing system

A common situation that occurs in everyday life is that of queuing or waiting in a line. Queues (waiting lines) are usually seen at bus stops, ticket booths, doctors' clinics, bank counters, traffic lights and so on queues are also found in workshops where the machines wait to be repaired; at a tool crib where the mechanics wait to receive tools; in a warehouse where items wait to be used, incoming calls wait to mature in the telephone exchange, trucks wait to be unloaded, airplanes wait either to take off or land and so on.

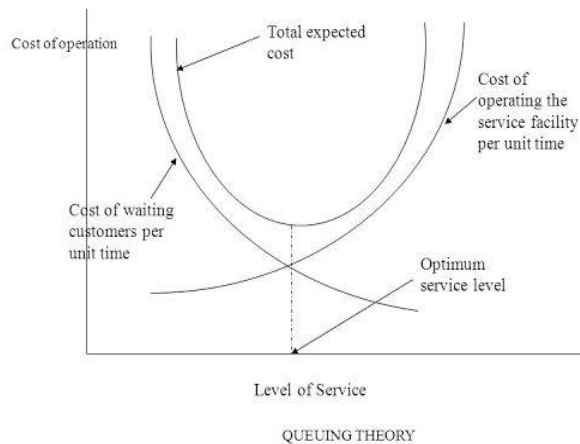
In general, a queue is formed at a queuing system when either customers (human beings or p entities) requiring service wait due to the fact that the number of customers exceeds the number of service facilities, or service facilities do not work efficiently and take more time than prescribed to serve a customer. Queuing theory can be applied to a variety of operational situations where it is not possible to accurately predict the rate (or time) of customers and service rate (or time) of service facility or facilities. In particular, it can be used to determine the level of service (either the service rate or the number of service facilities) that balances the following two conflicting costs:

- (i) cost of offering the service
- (ii) cost incurred due to delay in offering service

The first cost is associated with the service facilities and their operation, and the second represents the cost of customer's waiting time. Obviously, an increase in the existing service facilities would reduce the customer's waiting time.

Conversely, decreasing the level of service would result in long queue(s). This means an increase (decrease) in the level of service increases (decreases) the cost of operating service facilities but decreases (increases) the cost of waiting. Figure illustrates both types of costs as a function of level of service. The optimum service level is one that minimizes the sum of the two costs.

Queuing Costs vs. Level of Service



Since cost of waiting is difficult to estimate, it is usually measured in terms of loss of sales or goodwill when the customer is a human being who has no sympathy with the service. But, if the customer is a machine waiting for repair, then cost of waiting is measured in terms of cost of lost production. Many practical situations in which study of queuing theory can provide solution to waiting line problems are listed in Table

Situation	Customers	Service Facilities
Petrol pumps (station)	Automobiles	Pumps/Passionel
Hospitals	Patients	Doctors/Nurses/Rooms
Airport	Aircraft	Runways
Post office	Letters	Sorting System

Job interviews	Applicants	Interviewers
----------------	------------	--------------

DEFINITIONS

Customer: A person (or object) arriving at a service station for availing service.

Service station (or Service facility): The place where service is provided to the customers.

Queue: A line or a sequence of people/objects awaiting their turn to be attended to or serviced.

Waiting time in queue: The time that a customer spends in the queue before being taken up for service. It is the difference between the time of arrival of a customer and the time at which the service station takes-up the service for the customer.

Line length (or queue size) This refers to the total number of customers in the system who are actually waiting in the line and not being serviced. Queue length may be defined as the number of units waiting in a queue or present in a system.

NOTATIONS

The notations used for analyzing of a queuing system are as follows:

n = number of customers in the system (waiting and in service)

P_0 = Probability of n customers in the system

λ = average (expected) customer arrival rate or in the queuing system

μ = average (expected) service rate or average number of customers served per unit time at the place of service

$$\frac{\lambda}{\mu} = \rho = \frac{\text{Average service completion}(1/\mu)}{\text{Average inter arrival time}(\frac{1}{\lambda})}$$

ρ = traffic intensity or server utilization factor

P_0 = probability of no customer in the system, $1 - (\lambda/\mu)$

s = number of service channels (service facilities or servers)

N = maximum number of customers allowed in the system

L_s = average (expected) number of customers in the system (waiting and in service)

L_q = average (expected) number of customers in the queue (queue length)

L = average (expected) length of non-empty queue

W_s = average (expected) waiting time in the system (waiting and in service)

W_q = average (expected) waiting time in the queue

P_w = probability that an arriving customer has to wait (system being busy),

$1 - P_0 = (\lambda/\mu)$.

QUEUEING SYSTEM

The mechanism of a queueing process is very simple. Customers arrive at a service counter and are attended to by one or more of the servers. As soon as a customer is served, it departs from the system. Thus a queueing system can be described as consisting of customers arriving for service, waiting for service if it is not immediate, and leaving the system after being served. The general framework of a queueing system is shown below.

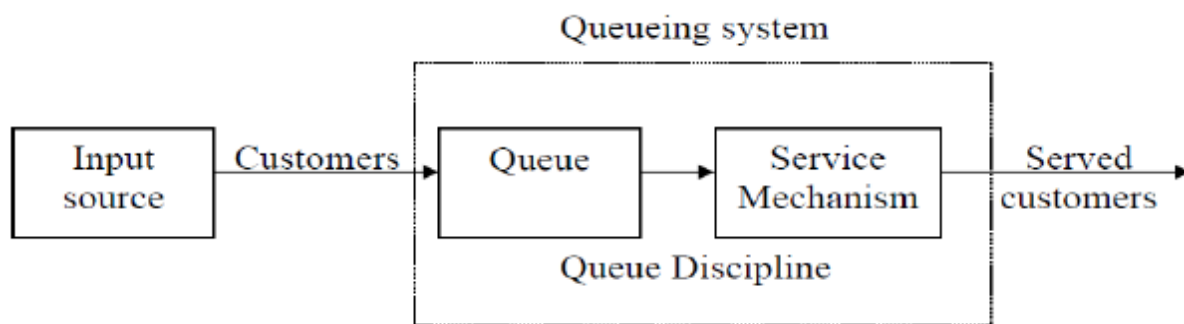


Figure 1. Basic queueing model structure (Balsam and Marin, 2007; Adan and Resing, 2015).

ELEMENTS OF A QUEUEING SYSTEM

The basic elements of a queueing system are as follows:

1. Input (or Arrival) Process. This element of queueing system is concerned with the pattern in which the customers arrive for service. Input source can be described by following three factors:

(a) ***Size of the queue.*** If the total number of potential customers requiring service are only few, then size of the input source is said to be finite. On the other hand, if potential customers requiring service are sufficiently large in number, then the input source is considered to be infinite.

Also, the customers may arrive at the service facility in batches of fixed size or of variable size or one by one. In the case when more than one arrival is allowed to enter the system simultaneously (entering the system does not necessarily mean entering into service), the input is said to occur in bulk or in batches. Ships discharging cargo at a dock, families visiting restaurants, etc. are the examples of bulk arrivals.

(b) ***Pattern of arrivals.*** Customers may arrive in the system at known (regular or otherwise) times, or they may arrive in a random way. In case the arrival times are known with certainty, the queueing problems are categorized as deterministic models. On the other hand, if the time between successive arrivals (inter-arrival times) is uncertain, the arrival pattern is measured by either an arrival rate or inter arrival time. These are characterised by the probability distribution associated with this random process. The most common stochastic queueing models assume that arrival rate follow a Poisson distribution and/or the inter-arrival times follow an exponential distribution.

(c) ***Customer's behaviour.*** It is also necessary to know the reaction of a customer upon entering the system. A customer may decide to wait no matter how long the queue becomes (patient customer or if the queue is too long to suit him, may decide not to enter it (impatient customer). Machines arriving at the maintenance shop in a plant are examples of patient customers. For impatient customers,

(i) if a customer decides not to enter the queue because of its length, he is said to have balked.

(ii) if a customer enters the queue, but after some time loses patience and decides to leave, then he is said to have reneged.

(iii) if a customer moves from one queue to another (providing similar/different services) for his personal economic gains, then he is said to have jockeyed for position.

The final factor to be considered regarding the input process is the manner in which the arrival pattern changes with time. The input process which does not change with time is called a stationary input process. If it is time dependent then the process is termed as transient.

2. Queue Discipline. It is a rule according to which customers are selected for service when queue has been formed. The most common queue discipline is the "first come, first served (FCFS or the first in, first out" (FIFO) rule under which the customers are serviced in the strict order at their arrivals. Other queue discipline include: "last in, first out" (LIFO) rule according to which the last arrival in the system is serviced first.

This discipline is practised in most cargo handling situations where the last item loaded is removed first. Another example may be from the production process, where items arrive at a workplace and are stacked one on top of the other. Item on the top of the stack is taken first processing which is the last one to have arrived for service. Besides these, other disciplines "selection for service in random order" (SIRO) rule according to which the arrivals are serviced randomly irrespective of their arrivals in the system; and a variety of priority schemes-according to which a customer's service is done in preference over some other customer.

Under priority discipline, the service is of two types:

(i) Pre-emptive priority. Under th the customers of high priority are given service over the low priority customers. That is lower priority customer's service is interrupted (pre-empted) to start service for a priority customer. The initial service is resumed again as soon as the highest priority customer has been served.

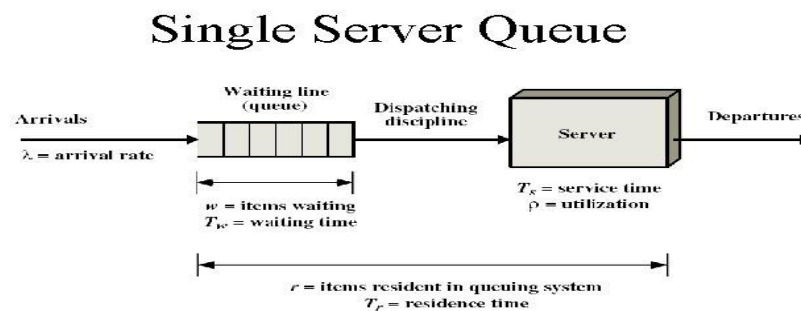
(ii) Non pre-emptive priority. In this case the highest priority customer goes ahead in the queue but his service is started only after the completion of the service of the currently being served customers.

3. Service Mechanism. The service mechanism is concerned with service time and service facilities. Service time is the time interval from the commencement of service to the completion of service. If there are infinite number of servers then all the customers are served instantaneously on arrival and there will be no queue.

If the number of servers is finite, then the customers are served according to a specific order. Further, the customers may be served in batches of fixed size or of variable size rather than individually by the same server, such as a computer with parallel processing or people boarding a bus. The service system in this case is termed as bulk service system.

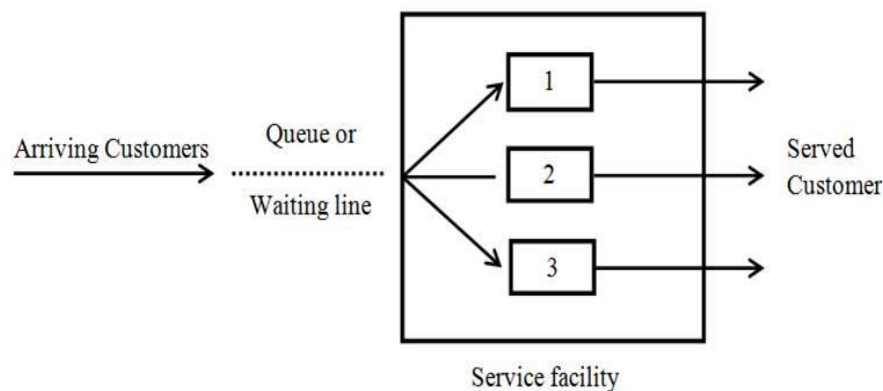
In the case of parallel channels "fastest server rule" (FSR) is adopted. For its discussion we suppose that the customers arrive before parallel service channels. If only one service channel is free, then incoming customer is assigned to free service channel. But it will be more efficient to assume that an incoming customer is to be assigned a server of largest service rate among the free ones. Service facilities can be of the following types:

(a) **Single queue-one server**, i.e., one queue-one service channel, wherein the customer waits till the service point is ready to take him in for servicing.

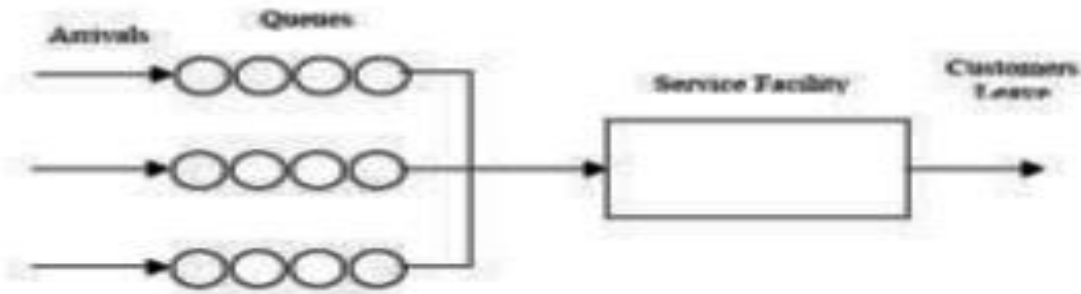


3

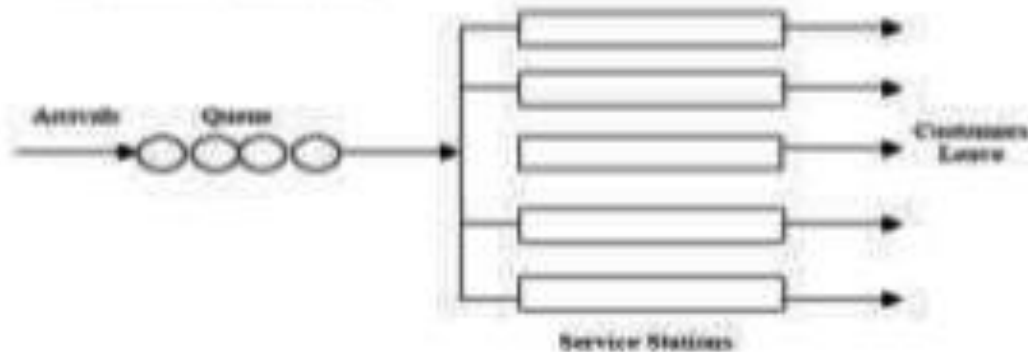
(b) **Single queue-several servers** wherein the customers wait in a single queue until one of the service channels is ready to take them in for servicing.



(c) **Several queues-one server** wherein there are several queues and the customer may join any one of these but there is only one service channel.



(d) **Several servers.** When there are several service channels available to provide service, much depends upon their arrangements. They may be arranged in parallel or in series or a more complex combination of both, depending on the design of the system's service mechanism.



By parallel channels, we mean a number of channels providing identical service facilities. Further, customers may wait in a single queue until one of the service channels is ready to serve, as in a barber shop where many chairs are considered as different service channels; or customers may form separate queues in front of each service channel as in the case of super markets.

For series channels, a customer must pass through all the service channels in sequence before service is completed. The situations may be seen in public offices where parts of the service are done at different service counters.

4. Capacity of the System. The source from which customers are generated may be finite or infinite. A finite source limits the customers arriving for service. i.e., there is a finite limit to the maximum queue size. The queue can also be viewed as one with forced balking where a customer is forced to balk if he arrives at a time when

queue size is at its limit. Alternatively, an infinite source is forever "abundant" as in the case of telephone calls arriving at a telephone exchange.

OPERATING CHARACTERISTICS OF A QUEUEING SYSTEM:

Some of the operational characteristics of a queueing system, that are of general interest for the evaluation of the performance of an existing queueing system and to design a new system are as follows:

1. Expected number of customers in the system denoted by $E(n)$ or L is the average number of customers in the system, both waiting and in service. Here, n stands for the number of customers in the queueing system.

2. Expected number of customers in the queue denoted by $E(m)$ or L_q , is the average number a customers waiting in the queue. Here $m = n - 1$, i.e, excluding the customer being served.

3. Expected waiting time in the system denoted by $E(v)$ or W is the average total time spent by a customer in the system. It is generally taken to be the waiting time plus servicing time.

4. Expected waiting time in queue denoted by $E(w)$ or W_q is the average time spent by a customer in the queue before the commencement of his service.

5. The server utilization factor (or busy period) denoted by $P (= \lambda/\mu)$ is the proportion of time that a server actually spends with the customers. Here, λ stands for the average number of customer arriving per unit of time and μ stands for the average number of customers completing service per of time.

The server utilization factor is also known as traffic intensity or the clearing ratio.

DETERMINISTIC QUEUEING SYSTEM

A queueing system wherein the customers arrive at regular intervals and the service time for each customer is known and constant, is known as a deterministic queueing system.

Let the customers come at the teller counter of a bank for withdrawal every 3 minutes. Thus the interval between the arrival of any two successive customers, that is the inter-arrival time, is exactly 3 minutes. Further, suppose that the incharge of that

particular teller takes exactly 3 minutes to serve a customer. This implies that the arrival and service rates are both equal to 20 customers per hour. In this situation there shall never be a queue and the incharge of the teller shall always be busy with servicing of customers.

Now suppose instead, that the incharge of the teller can serve 30 customers per hour, i.e., he takes 2 minutes to serve a customer and then has to wait for one minute for the next customer to come for service. Here also, there would be no queue, but the teller is not always busy.

Further, suppose that the incharge of the teller can serve only 15 customers per hour, i.e., he takes 4 minutes to serve a customer. Clearly, in this situation he would be always busy and the queue length will increase continuously without limit with the passage of time. This implies that when the service rate is less than the arrival rate, the service facility cannot cope with all the arrivals and eventually the system leads to an explosive situation. In such situations, the problem can be resolved by providing additional service facilities, like opening parallel counters. We can summarize the above as follows:

Let the arrival rate be λ customers per unit time and the service rate be μ customers per unit time. Then,

- (i) if $\lambda > \mu$, the waiting line (queue) shall be formed and will increase indefinitely: the service facility would always be busy and the service system will eventually fail.
- (ii) if $\lambda \leq \mu$, there shall be no queue and hence no waiting time: the proportion of time the service facility would be idle is $1 - \lambda/\mu$.

However, it is easy to visualize that the condition of uniform arrival and uniform service rates has a very limited practicability. Generally, the arrivals and servicing time are both variable and uncertain. Thus, variable arrival rates and servicing times are the more realistic assumptions. The probabilistic queueing models are based on these assumptions.

PROBABILITY DISTRIBUTIONS IN QUEUEING SYSTEMS

It is assumed that customers joining the queueing system arrive in a random manner and follow a *Poisson distribution* or equivalently the inter-arrival times obey

exponential distribution. In most of the cases, service times are also assumed to be exponentially distributed. It implies that the probability of service completion in any short time period is constant and independent of the length of time that the service has been in progress.

In this section, the arrival and service distributions for Poisson queues are derived. The basic assumptions (axioms) governing this type of queues are stated below:

Axiom 1. The number of arrivals in non-overlapping intervals are statistically independent, that is, the process has independent increments.

Axiom 2. The probability of more than one arrival between time t and time $t + \Delta t$ is $o(\Delta t)$; that is, the probability of two or more arrivals during the small time interval Δt is negligible

Thus
$$P_o(\Delta t) + P_l(\Delta t) + o(\Delta t) = 1.$$

Axiom 3. The probability that an arrival occurs between time t and time $t + \Delta t$ is equal to $\lambda \Delta t + o(\Delta t)$.

Thus
$$P_l(\Delta t) = \lambda \Delta t + o(\Delta t).$$

Where λ is a constant and is independent of the total number of arrivals upto time t . Δt is an incremental element and $o(\Delta t)$ represents the terms such that $\lim_{\Delta t \rightarrow 0} \frac{o(\Delta t)}{\Delta t} = 0$

1. Distribution of Arrivals (Pure Birth Process)

The model in which only arrivals are counted and no departure takes place are called *pure birth models*. Stated in terms of queueing, birth-death processes usually arise when an additional customer increases the arrival (referred as birth) in the system and decreases by departure (referred as death) of serviced customer from the system.

Let $P_n(t)$ denote the probability of n arrivals in a time interval of length t (both waiting and in service), where $n \geq 0$ is an integer. Then $P_n(t + \Delta t)$ being the probability of n arrivals in a time interval of length $t + \Delta t$ (making use of *axiom 1*) is as follows:

$$\begin{aligned} P_n(t + \Delta t) &= P\{n \text{ arrivals in time } t \text{ and one arrival in time } \Delta t\} \\ &+ P\{(n-1) \text{ arrivals in time } t \text{ and one arrival in time } \Delta t\} \\ &+ P\{(n-2) \text{ arrivals in time } t \text{ and two arrivals in time } \Delta t\} \\ &+ \dots + P\{\text{no arrival in time } t \text{ and arrivals in time } \Delta t\}. \text{ for } n \geq 1. \end{aligned}$$

Making use of *axiom 2* and *axiom 3*, this difference equation reduces to

$$\begin{aligned} P_n(t + \Delta t) &= P_n(t) P_0(\Delta t) + P_{n-1}(t) P_1(\Delta t) + o(\Delta t) \\ &= P_n(t) [1 - \lambda \Delta t - o(\Delta t)] + P_{n-1}(t) \{ \lambda \Delta t + o(\Delta t) \} + o(\Delta t) \end{aligned}$$

where the last term, $o(\Delta t)$, represents the terms

$$P[(n-k) \text{ arrivals in time } t \text{ and } k \text{ arrivals in time } \Delta t] \quad 2 \leq k \leq n$$

The above equation can be re-written as

$$P_n(t + \Delta t) - P_n(t) = -\lambda \Delta t P_n(t) + \lambda \Delta t P_{n-1}(t) + o(\Delta t)$$

Dividing it by Δt on both sides and then taking the limit as $\Delta t \rightarrow 0$, the equation reduces to

$$\frac{d}{dt} P_n(t) = -\lambda P_n(t) + \lambda P_{n-1}(t). \quad n \geq 1. \quad \dots\dots\dots(A)$$

For the case when $n=0$.

$$P_0(t + \Delta t) = P_0(t) P_0(\Delta t) = P_0(t) [1 - \lambda \Delta t - o(\Delta t)]$$

Rearranging the terms and then dividing on both sides by Δt . taking the limit as $\Delta t \rightarrow 0$, we have

$$\frac{d}{dt} P_0(t) = -\lambda P_0(t) \quad \dots\dots\dots(B)$$

To solve the $n+1$ differential-difference equations given in (A) and (B), we make use of the generating function

$$\phi(z, t) = \sum_{n=0}^{\infty} P_n(t) \cdot z^n.$$

in the unit circle $|z| \leq 1$.

Now multiplying the differential-difference equations given in (B) and (A) by $z^0, z^1, z^2, \dots, z^n$. respectively and then taking summation over n from 0 to ∞ , we get

$$\sum_{n=0}^{\infty} \frac{d}{dt} P_n(t) z^n = -\lambda \phi(z, t) + \lambda z \phi(z, t).$$

This can also be written as

$$\frac{d}{dt} \phi(z, t) = \lambda(z-1)\phi(z, t)$$

An obvious solution of this differential equation is

$$\phi(z, t) = C e^{\lambda(z-1)t}.$$

where C is an arbitrary constant.

To determine the value of C, we use the initial condition that there is no arrival by time $t \rightarrow 0$ and this gives

$$\phi(z, 0) = P_0(0) + \sum_{n=1}^{\infty} P_n(0)Z^n = 1$$

Now, $P_n(0) = 0$ for $n \geq 1$. Therefore, $C = 1$.

$$\text{Hence, } \phi(z, t) = e^{\lambda(z-1)t}. \quad \dots\dots\dots(C)$$

$$\text{Now, } \frac{d}{dz} \phi(z, t)|_{z=0} = P_1(t), \frac{d^2}{dz^2} \phi(z, t)|_{z=0} = 2! P_2(t)$$

$$\frac{d^n}{dz^n} \phi(z, t)|_{z=0} = n! P_n(t)$$

Using the value of $\phi(z, t)$ as given in equation (C), we get

$$\begin{aligned} P_0(t) &= e^{-\lambda t}, & P_1(t) &= \lambda(t)e^{-\lambda t}, \\ P_2(t) &= \frac{1}{2!}(\lambda t)e^{-\lambda t}, & P_n(t) &= \frac{1}{n!}(\lambda t)^n e^{-\lambda t}. \end{aligned}$$

The general formula, therefore is

$$P_n(t) = \frac{(\lambda t)^n}{n!} e^{-\lambda t}, \text{ for } n \geq 0$$

which is the well-known *Poisson probability law* with mean λt . Thus, the random variable defined as the number of arrivals to a system in time t , has the *Poisson distribution* with a mean of λt arrival or a mean arrival rate of λ .

2. Distribution of Inter-arrival Times (*Exponential Process*)

Inter-arrival times are defined as the time intervals between two successive arrivals. Here, we shall show that if the arrival process follows the Poisson distribution, an associated random variable defined as the time between successive arrivals (inter-arrival time) follows the exponential distribution $f(t) = \lambda e^{-\lambda t}$ and vice-versa.

Let the random variable T be the time between successive arrivals; then

$$P(T > t) = P(\text{no arrival in time } t) = P_0(t) = e^{-\lambda t}$$

The cumulative distribution function of T denoted by $F(t)$ is given by

$$\begin{aligned} F(t) &= P(T \leq t) = 1 - P(T > t) \\ &= 1 - P_0(t) = 1 - e^{-\lambda t}, t > 0 \end{aligned}$$

The density function $f(t)$ for inter-arrival times, therefore, is

$$f(t) = \frac{d}{dt} F(t) = \lambda e^{-\lambda t}, t > 0$$

The expected (or mean) inter-arrival time is given by $E(t) = \int_0^{\infty} t \cdot f(t) dt$

$$\begin{aligned} &= \int_0^{\infty} \lambda t e^{-\lambda t} dt \\ &= 1/\lambda \end{aligned}$$

where λ is the *mean arrival rate*.

Thus, T has the *exponential distribution* with mean $1/\lambda$. We would intuitively expect that, if the mean arrival rate is λ , then the mean time between arrivals is $1/\lambda$. Conversely, we can also show that if the inter-arrival times are independent and have the same exponential distribution then the arrival rate follows the Poisson distribution.

3. Distribution of Departures (*Pure Death Process*)

The model in which only departures are counted and no other arrivals allowed are called pure death models. The queueing system starts with N customers at time $t=0$, where $N \geq 1$. Departures occur at the rate of μ customers per unit time. To develop the differential-difference equations for the probability of n customers remaining after ' t ' time units, $P_n(t)$, we make use of similar assumptions as was done for arrivals. Let the three axioms, given at the beginning of this section, be changed by using the word service instead of arrival and condition the probability statements by requiring the system to be non-empty. Let us define

$\mu \Delta t$ = probability that a customer in service at time t will service during time Δt . For small time interval $\Delta t > 0$, $\mu \Delta t$ gives probability of one departure during Δt . Using the same arguments as in pure birth process case, the differential-difference equations for this can easily be obtained.

$$P_n(t + \Delta t) = P_n(t) \{1 - \mu \Delta t + o(\Delta t)\} + P_{n+1}(t) \{\mu \Delta t + o(\Delta t)\}, \quad 1 \leq n \leq N-1$$

$$P_0(t + \Delta t) = P_0(t) + P_1(t) \{\mu \Delta t + o(\Delta t)\}, \quad n = 0$$

$$P_N(t + \Delta t) = P_N(t) \{1 - \mu \Delta t + o(\Delta t)\}, \quad n = N$$

Re-arranging the above equations, dividing them by Δt on both sides and then taking the limits as $\Delta t \rightarrow 0$, we get

$$\frac{d}{dt} P_n(t) = -\mu P_n(t) + \mu P_{n+1}(t) \quad 0 \leq n \leq N-1, \quad t > 0$$

$$\frac{d}{dt} P_0(t) = \mu P_1(t); \quad n = 0, t \geq 0$$

$$\frac{d}{dt} P_N(t) = -\mu P_N(t); \quad n = N, t \geq 0$$

The solution of these equations with initial conditions :

$$P_n(0) = \begin{cases} 1 & ; \quad n = N \neq 0 \\ 0 & ; \quad n \neq N \end{cases}$$

can easily be obtained as earlier. The general solution to the above equation so obtained is

$$P_n(t) = \frac{(\mu t)^{N-n} e^{-\mu t}}{(N-n)!}; \quad 1 \leq n \leq N \quad \text{and} \quad P_0(t) = 1 - \sum_{n=1}^N P_n(t)$$

which is known as a *truncated Poisson law*.

4. Distribution of Service Times

Making similar assumption as done above for arrivals, one could utilize the same type of process to describe the service pattern. Let the three axioms be changed by using the word *service* instead of *arrival* and condition the probability statements by requiring the system to be *non-empty*. Then we can easily show that, the time t to complete the service on a customer follows the exponential distribution:

$$s(t) = \begin{cases} \mu e^{-\mu t} & ; \quad t > 0 \\ 0 & ; \quad t < 0 \end{cases}$$

Where μ is the mean service rate for a particular service channel. This shows that follows exponential distribution which mean $1/\mu$. The number, n , of potential services in time t will follow the *Poisson distribution* given by

$$\Phi(n) = P\{n \text{ service in time } T, \text{ if servicing is going on throughout } T\}$$

$$= \frac{(\mu T)^N}{n!} e^{-\mu T}$$

Consequently, we can also show that

$$P[\text{no service in } \Delta t] = 1 - \mu \Delta t + o(\Delta t) \quad \text{and} \quad P[\text{one service in } \Delta t] = \mu \Delta t + o(\Delta t)$$

CLASSIFICATION OF QUEUEING MODELS

Generally queueing model may be completely specified in the following symbolic form :

$$(a/b/c) : (d/e).$$

The first and second symbols denote the type of distributions of inter-arrival times and inter-service times, respectively. Third symbol specifies the number of servers, whereas fourth symbol stands for the capacity of the system and the last symbol denotes the queue discipline.

If we specify the following letters as:

M = Poisson arrival or departure distribution.

E = Erlangian or Gamma inter-arrival for service time distribution.

GI = General input distribution,

G = General service time distribution.

then $(M/E_k/1) : (\infty/\text{FIFO})$ defines a queueing system in which arrivals follow Poisson distribution service times are Erlangian, single server, infinite capacity and “ first in, first out” queue discipline.

DEFINITION OF TRANSIENT AND STEADY STATES

A queueing system is said to be in *transient state* when its operating characteristic (like input, mean queue length, etc.) are dependent upon time.

If the characteristic of the queueing system becomes independent of time, then the steady state condition is said to prevail

If $P_n(t)$ denotes the probability that there are n customers in the system at time t , then in the steady state case, we have

$$\lim_{t \rightarrow \infty} P_n(t) = P_n(\text{independent of } t)$$

Due to practical viewpoint of the steady state behaviour of the systems, the present chapter is simply focused on studying queueing systems under the existence of steady-state conditions. However the differential-difference equations which can be used for deriving transient solutions will be presented.

POISSON QUEUEING SYSTEM

Queues, that follow the poisson arrivals (exponential inter-arrival time) and Poisson service (exponential service time) are called Poisson queues. In this section, we shall study a number of Poisson queues with different characteristics.

Model 1 ((M/M/1): (∞ /FIFO)). This model deals with a queuing system having single service Thennet. Poisson input, Exponential service and there is no limit on the system capacity while the inmers are served on a "first in, first out" basis.

The solution procedure of this queueing model can be summarized in the following three steps:

Step 1. Construction of Differential Difference Equations. Let $p_n(t)$ be the probability that there are n customers in the system at time t . The probability that the system has n customers at time $(t+\Delta t)$ can be expressed as the sum of the joint probabilities of the four mutually exclusive and collectively exhaustive events as follows :

$$\begin{aligned} P_n(t+\Delta t) = & P_n(t) \cdot P[\text{no arrival in } \Delta t] \cdot P[\text{no service completion in } \Delta t] \\ & + p_n(t) \cdot P[\text{one arrival in } \Delta t] \cdot P[\text{one service completed in } \Delta t] \\ & + P_{n+1}(t) \cdot P[\text{no arrival in } \Delta t] \cdot P[\text{one service completed in } \Delta t] \\ & + P_{n-1}(t) \cdot P[\text{one arrival in } \Delta t] \cdot p[\text{one service completed in } \Delta t] \end{aligned}$$

This is re-written as:

$$\begin{aligned} P_n(t+\Delta t) = & p_n(t)[1 - \lambda\Delta t + o(\Delta t)][1 - \mu\Delta t + o(\Delta t)] + P_n(t)[\lambda\Delta t][\mu\Delta t] \\ & + P_{n+1}(t)[1 - \lambda\Delta t + o(\Delta t)][\mu\Delta t + o(\Delta t)] + p_{n-1}(t)[\lambda\Delta t + o(\Delta t)][1 - \mu\Delta t + o(\Delta t)] \end{aligned}$$

$$\text{Or } P_n(1+\Delta t) - p_n(t) = -(\lambda + \mu)\Delta t P_n(t) + \mu\Delta t P_{n+1}(t) + o(\Delta t)$$

Since Δt is very small, terms involving $(\Delta t)^2$ can be neglected. Dividing the above equation by Δt on both sides and then taking limit as $\Delta t \rightarrow 0$, we get

$$d/dt(p_n) = -(\lambda + \mu)p_n(t) + \mu P_{n+1}(t) + \lambda P_{n-1}(t) ; n \geq 1$$

Similarly, if there is no customer in the system at time $(t + \Delta t)$, there will be no service completion during Δt . Thus for $n=0$ and $t \geq 0$, we have only two probabilities instead of four. The resulting equation is

$$P_0(t + \Delta t) = p_0(t) \{1 - \lambda \Delta t + o(\Delta t)\} + P_1(t) \{\mu \Delta t + o(\Delta t)\} \{1 - \lambda \Delta t + o(\Delta t)\}$$

or $P_0(t + \Delta t) - p_0(t) = -\lambda \Delta t P_0(t) + \mu \Delta t P_1(t) + o(\Delta t).$

Dividing both sides of this equation by Δt and then taking limit as $\Delta t \rightarrow 0$, we get

$$d/dt(p_0(t)) = -\lambda p_0(t) + \mu P_1(t) ; \quad n = 0$$

Step 2. Deriving the Steady-State Difference Equations. In the steady-state. $P_n(t)$ is independent of time t and $\lambda < \mu$ when $t \rightarrow \infty$. Thus $P_n(t) \rightarrow P_n$ and

Consequently the differential-difference equations obtained in Step 1 reduce to

$$0 = -(\lambda + \mu)p_n + \mu P_{n+1} + \lambda P_{n-1} ; n \geq 1$$

and $0 = -\lambda P_n + \mu P_1 ; n = 0$

These constitute the steady-state difference equations.

Step 3. Solution of the Steady-State Difference Equations. For the solution of the above difference equations there exist three methods, namely, the iterative method, use of generating functions and the use of linear operators. Out of these three the first one is the most straightforward and therefore the solution of the above equation will be obtained here by using the iterative method.

Using iteratively, the difference-equation yield

$$P_1 = \frac{\lambda}{\mu} P_0, \quad P_2 = \frac{\lambda + \mu}{\mu} P_1 - \frac{\lambda}{\mu} P_0 = \left(\frac{\lambda}{\mu}\right)^2 P_0$$

$$P_3 = \left(\frac{\lambda + \mu}{\mu}\right) P_2 - \frac{\lambda}{\mu} P_1 = \left(\frac{\lambda}{\mu}\right)^3 P_0, \quad \text{and in general } P_n = \left(\frac{\lambda}{\mu}\right)^n P_0.$$

Now, $P_{n+1} = \frac{\lambda + \mu}{\mu} P_n - \frac{\lambda}{\mu} P_{n-1}, n \geq 1.$

Substituting the values of P_n and P_{n-1} , the equation yields

$$P_{n+1} = \frac{\lambda + \mu}{\mu} \left(\frac{\lambda}{\mu}\right)^n P_0 - \frac{\lambda}{\mu} \left(\frac{\lambda}{\mu}\right)^{n-1} P_0 = \left(\frac{\lambda}{\mu}\right)^{n+1} P_0$$

Thus, by the principle of mathematical induction, the general formulae for P_n , is valid for $n \geq 0$

To obtain the value of P_0 , we make use of the boundary condition $\sum_{n=0}^{\infty} P_n = 1$

$$\begin{aligned} \therefore 1 &= \sum_{n=0}^{\infty} \left(\frac{\lambda}{\mu}\right)^n P_0 = P_0 \sum_{n=0}^{\infty} \left(\frac{\lambda}{\mu}\right)^n; \quad \text{since } P_n = \left(\frac{\lambda}{\mu}\right)^n P_0 \\ &= P_0 \frac{1}{1-\lambda/\mu}, \quad \text{since } \left(\frac{\lambda}{\mu}\right) < 1 \end{aligned}$$

This gives $P_0 = 1 - \left(\frac{\lambda}{\mu}\right)$.

Hence, the steady- state solution is

$$P_n = \left(\frac{\lambda}{\mu}\right)^n \left(1 - \frac{\lambda}{\mu}\right) = \rho^n (1-\rho); \quad \rho = \left(\frac{\lambda}{\mu}\right) < 1, \text{ and } n \geq 0.$$

This expression gives us the probability distribution of queue length.

Characteristic of Model I

(i) Probability of queue size being greater than or equal than or equal to k, the number of customer is given by

$$\begin{aligned} P(n \geq k) &= \sum_{k=n}^{\infty} P_k = \sum_{k=n}^{\infty} (1-\rho)\rho^k = (1-\rho)\rho^n \sum_{k=n}^{\infty} \rho^{k-n} = (1-\rho)\rho^n \sum_{k-n=0}^{\infty} \rho^{k-n} \\ &= (1-\rho)\rho^n \sum_{x=0}^{\infty} \rho^x = \frac{(1-\rho)\rho^n}{1-\rho} = \rho^n \end{aligned}$$

(ii) Average number of customer in the system is given by

$$\begin{aligned} E(n) &= \sum_{n=0}^{\infty} n P_n = \sum_{n=0}^{\infty} n (1-\rho)\rho^n = (1-\rho) \sum_{n=0}^{\infty} n \rho^n = \rho(1-\rho) \sum_{n=0}^{\infty} n \rho^{n-1} \\ &= \rho(1-\rho) \sum_{n=0}^{\infty} \frac{d}{d\rho} \rho^n = \rho(1-\rho) \frac{d}{d\rho} \sum_{n=0}^{\infty} \rho^n, \quad \text{since } \rho < 1 \\ &= \rho(1-\rho) \frac{1}{(1-\rho)^2} = \frac{\rho}{1-\rho} = \frac{\lambda}{\mu-\lambda}. \end{aligned}$$

(iii) Average queue length is given by

$$E(m) = \sum_{m=0}^{\infty} m P_n, \quad \text{where } m = n - 1$$

being the number of customer in the queue excluding the customer which is in service.

$$\begin{aligned} \therefore E(m) &= \sum_{n=1}^{\infty} (n-1) P_n = \sum_{n=1}^{\infty} n P_n - \sum_{n=1}^{\infty} P_n \\ &= \sum_{n=0}^{\infty} n P_n - [\sum_{n=0}^{\infty} P_n - P_0] \end{aligned}$$

$$= \frac{\rho}{1-\rho} - [1-(1-\rho)] = \frac{\rho}{1-\rho} - \rho$$

$$= \rho^2 / (1-\rho) = \lambda^2 / (\mu - \lambda) .$$

(iv) Average length of non-empty queue is given by

$$E(m|m > 0) = \frac{E(m)}{P(m > 0)} = \frac{\lambda^2}{\mu(\mu-\lambda)} \times \frac{1}{(\frac{\lambda}{\mu})^2} = \frac{\mu}{\mu-\lambda} .$$

Since $P(m > 0) = P(n > 1) = \sum_{n=0}^{\infty} P_n - P_0 - P_1 = \left(\frac{\lambda}{\mu}\right)^2$

(v) The fluctuation (variance) of queue length is given by

$$V(n) = \sum_{n=0}^{\infty} [n - E(n)]^2 P_n = \sum_{n=0}^{\infty} n^2 P_n - [E(n)]^2$$

Using some algebraic transformation and the value of P_n the result reduces to

$$V(n) = (1 - \rho) \frac{\rho + \rho^2}{(1 - \rho)^3} - \left[\frac{\rho}{1 - \rho} \right]^2 = \frac{\rho}{(1 - \rho)^2} = \frac{\lambda \mu}{(\mu - \lambda)^2} .$$

Waiting Time Distribution for Model I.

The Waiting time of a customer in the system is, for the most part, a continuous random variable except that there is a non zero probability that the delay will be zero, that is a customer entering service immediately upon arrival. Therefore, if we denote the time spent in the queue by w and $\Psi_w(t)$ denotes its cumulative probability distribution then from the complete randomness of the Poisson distribution, we have

$$\Psi_w(0) = P(w = 0) \quad (\text{No customers in the system upon arrival})$$

$$= P_0 = (1 - \rho).$$

It is now required to find $\Psi_w(t)$ for $t > 0$

Let there be n customers in the system upon arrival then in order for a customer to go into service at a time between 0 and t , all the n customers must have been served by time t . Let $s_1, s_2, s_3, \dots, s_n$ denote service times of n customers respectively. Then

$$W = \sum_{i=1}^n s_i, \quad (n \geq 1) \quad \text{and} \quad w = 0 \quad (n = 0).$$

The distribution function of waiting time, w , for a customer who has to wait is given by

$$P(w \leq t) = P[\sum_{i=1}^n s_i \leq t] ; \quad n \geq 1 \quad \text{and } t > 0.$$

Since, the service time for each customer is dependent and identically distributed, therefore its probability density function is given by $\mu e^{-\mu t} (t > 0)$, where μ is the mean service rate. Thus

$$\Psi_n(t) = \sum_{n=1}^{\infty} p_n \times P(n-1 \text{ customer are served at time } t) \times P(1 \text{ customer is served in time } \Delta t)$$

$$= \sum_{n=1}^{\infty} \left(1 - \frac{\lambda}{\mu}\right) \left(\frac{\lambda}{\mu}\right)^n \frac{(\mu t)^{n-1} e^{-\mu t}}{(n-1)!} \cdot \mu \Delta t.$$

The expression for $\Psi_w(t)$, therefore, can be written as

$$\begin{aligned} \Psi_w(t) &= P(w \leq t) = \sum_{n=1}^{\infty} P_n \int_0^t \Psi_n(t) dt \\ &= \sum_{n=1}^{\infty} (1 - \rho) \rho^n \int_0^t \frac{(\mu t)^{n-1}}{(n-1)!} e^{-\mu t} \cdot \mu dt = (1 - \rho) \rho \int_0^t \mu e^{-\mu t} \sum_{n=1}^{\infty} \frac{(\mu t \rho)^n}{(n-1)!} \cdot dt \\ &= (1 - \rho) \rho \int_0^t \mu e^{-\mu t(1 - \rho)} dt. \end{aligned}$$

Example 1: A road transport company has one reservation clerk on duty at a time. He handles information of bus schedules and makes reservations. Customers arrive at a rate of 8 per hour and the clerk can service 12 customers on an average per hour. After stating your assumptions, answer the following :

(i) What is the average number of customer waiting for the service of the clerk?

(ii) What is the average time a customer has to wait before getting service?

(iii) The management is contemplating to install a computer system to handle the information and reservation. This is expected to reduce the service time from 5 to 3 minutes. The additional cost of having the new system works out to Rs. 50 per day, If the cost of goodwill of having to wait is estimated to be 12 paise per minute spent waiting before being served. Should the company install the computer system? Assume 8 hours working day.

Solution:

We are given

$$\lambda = 8 \text{ customers per hour and } \mu = 12 \text{ customers per hour.}$$

(i) Average number of customers waiting for the service of the clerk(in the system):

$$E(n) = \frac{\lambda}{\mu - \lambda} = \frac{8}{12 - 8} = 2 \text{ customers.}$$

The Average number of customers waiting for the service of the clerk(in the queue)

$$E(m) = \frac{\lambda^2}{\mu(\mu - \lambda)} = \frac{8 \times 8}{12(12 - 8)} \text{ or } 1.33 \text{ customer.}$$

(ii) The Average waiting time of a customer (in the system) before getting service :

$$E(v) = \frac{1}{\mu - \lambda} = \frac{1}{12 - 8} \text{ hour or 15 minutes.}$$

The Average waiting time of a customer(in the queue) before getting service:

$$E(w) = \frac{\lambda}{\mu(\mu - \lambda)} = \frac{8}{12(12 - 8)} = \frac{1}{6} \text{ hours or 10 minutes.}$$

(iii) We now calculate the difference between the goodwill cost of customers with one system and the goodwill cost of customers with an additional system. This difference will be compared with the additional cost (of Rs . 50 per day) of installing another computer system .

An arrival waits for $E(w)$ hours before being served and there are λ arrivals per hour . Thus , expected waiting time for all customer in an 8-hours day with one system

$$= 8\lambda \times E(w) = 8 \times 8 \times \frac{1}{6} \text{ hrs . or } \frac{64}{6} \times 60 \text{ minutes , i.e., 640 minutes.}$$

The goodwill cost per day with one system = $640 \times \text{Rs. } 0.12 = \text{Rs. } 76.80$

The expected waiting time of a customer before getting service when there is an additional computer system is:

$$E(w^*) = \frac{8}{20(20 - 8)} = \frac{8}{20 \times 12} \text{ or } \frac{1}{30} \text{ hr .}$$

Thus expected waiting time of customer in an 8-hour day with an additional computer system is

$$8\lambda \times E(w^*) = 8 \times 8 \times \frac{1}{30} \text{ hr.} = 128 \text{ minutes.}$$

The total goodwill cost with an additional computer system

$$= 128 \times \text{Re. } 0.12 = \text{Rs. } 15.36$$

Hence, reduction in goodwill cost with the installation of a computer system

$$= \text{Re. } 76.80 - \text{Rs. } 15.36 = \text{Rs. } 61.44$$

Whereas the additional cost of a computer system is Rs.50 per day , Rs.61.44 is the reduction in goodwill cost when additional computer system is installed, hence there will be net saving at Rs.11.44 per day. It is ,therefore, worthwhile to install a computer.

Example 2 :In the production shop of a company the breakdown of the machines is found to be poisson with an average rate 3 machines per hour . Breakdown time at one machine cost Rs. 40 per hour to the company. There are two choice before the company for hiring the repairman. One of the repairman is slow but cheap , the other fast but expensive . The slow-cheap repairman demands Rs. 20 per hour and will repair the broken down machines exponentially at the rate of 4 per hour. The fast expensive repairman demands Rs. 30 per hour and will repair machines exponentially at the average rate of 6 per hour. Which repairman should be hired ?

Solution . In this problem ,we compare the total expected daily cost for both the repairman. This would equal the total wages paid plus the downtime cost.

Case 1: Slow-cheap repairman

$\lambda = 3$ machines per hour and $\mu = 4$ machines per hour.

\therefore Average downtime of a machine $= \frac{1}{\mu - \lambda} = \frac{1}{4-3} = 1$ hour.

\therefore The downtime of 3 machine that arrive in an hour $= 1 \times 3 = 3$ hours.

Downtime cost = Rs. 40 \times 3 = Rs. 120,

charges paid to the repairman =Rs 20 \times 3 = Rs. 60

Total cost = Rs. 120 + Rs 60 = Rs 180.

Case 2 : Fast-expensive repairman

$\lambda = 3$ machines per hour and $\mu = 6$ machines per hour

\therefore Average downtime of machines $= \frac{1}{\mu - \lambda} = \frac{1}{3}$ hour

\therefore The downtime of 3 machines that arrive in an hour $= \frac{1}{3} \times 3 = 1$ hour.

Downtime cost = Rs. 40 \times 1 = Rs. 40,

charges paid to the repairman = Rs. 30 \times 1 = Rs.30

Total cost = Rs. 40 + Rs. 30 =Rs. 70.

From the above two cases , the decision of the company should be to engage the fast-expensive repairman.

Model II $\{(M/M/1) : (\infty/SIRO)\}$. This model is essentially the same as Model I, except that the service discipline follows the SIRO – rule (service in random order) instead of FIFO – rule. As the derivation of P_n for model I does not depend on any specific queue discipline, it may be concluded that for SIRO-rule case, we must have

$$P_n = (1 - \rho)\rho^n, \quad n \geq 0$$

Consequently , whether the queue discipline follows the SIRO-rule or FIFO-rule the average number of customers in the system , $E(n)$, will remain the same. In fact $E(n)$ will remain the same as any queue discipline provided, of course, P_n remains unchanged. Thus, $E(v) = \frac{1}{\lambda} E(n)$ under the SIRO – rule is the same as under the FIFO-rule .

This result can be extended to any queue discipline as long as P_n remain unchanged. Specifically the result applies to the three most common disciplines, namely, FIFO, LIFO and SIRO. The three queue disciplines differ only in the distribution of waiting time where the probabilities of long and short waiting times change depending upon the discipline used. Thus we can use the symbol μt (general discipline) to represent the disciplines FIFO, LIFO and SIRO, When the waiting time distribution is not required.

Model III $\{(M/M/1) : (N/FIFO)\}$. This model differs from that of *Model I* in the sense that the maximum number of customers in the system is limited to N . Therefore, the difference equation of *Model I* are valid for this model as long as $n < N$.

The additional difference equation for $n = N$, is

$$P_N(t + \Delta t) = P_N(t)[1 - \mu\Delta t] + P_{N-1}(t) \cdot [\lambda\Delta t][1 - \mu\Delta t] + 0(\Delta t).$$

This gives, after simplification, the differential-difference equation

$$\frac{d}{dt} \cdot P_N(t) = -\mu P_N(t) + \lambda P_{N-1}(t)$$

from which the resultant steady-state difference equation is

$$0 = -\mu P_N + \lambda P_{N-1}$$

The complete set of steady-state difference equations for this model, therefore, can be written as

$$\mu P_1 = \lambda P_0$$

$$\mu P_{n+1} = (\lambda + \mu)P_n - \lambda P_{n-1} \quad 1 \leq n \leq N-1$$

$$\text{and } \mu P_N = \lambda P_{N-1}$$

Using the iterative procedure (as in *Model I*), the first two difference equations give

$$P_n = (\lambda/\mu)^n P_0, \quad n \leq N-1$$

Also, we see that for this value of P_n , the third (last) difference equation holds for $n = N$

Therefore, we have

$$P_n = (\lambda/\mu)^n P_0 = \rho^n P_0, \quad n \leq N \quad \text{and } (\lambda/\mu)^n = \rho$$

For obtaining the value of P_0 , we make use of the boundary conditions, $\sum_{n=0}^N P_n = 1$. Therefore

$$1 = P_0 \sum_{n=0}^N \rho^n = \begin{cases} P_0 \frac{1-\rho^{N+1}}{1-\rho}, & (\rho \neq 1) \\ P_0 (N+1), & (\rho = 1) \end{cases}$$

Thus

$$P_0 = \begin{cases} \frac{1-\rho}{1-\rho^{N+1}}, & (\rho \neq 1) \\ \frac{1}{N+1}, & (\rho = 1) \end{cases}$$

Hence,

$$P_n = \begin{cases} \frac{(1-\rho)\rho^n}{1-\rho^{N+1}}, & (\rho \neq 1); \\ \frac{1}{N+1}, & (\rho = 1) \end{cases} \quad 0 \leq n \leq N$$

Remark. The steady-state solution exists even for $\rho \geq 1$. Intuitively this makes sense since the maximum limit prevents the process from "blowing up". If $N \rightarrow \infty$, then the steady-state solution is

$$P_n = (1-\rho)\rho^n; \quad n < \infty$$

This result is in complete agreement with that of Model I.

Characteristics of Model III

(i) Average number of customers in the system is given by

$$E(n) = \sum_{n=0}^N n P_n = P_0 \sum_{n=0}^N n \rho^n = P_0 \rho \sum_{n=0}^N \frac{d}{d\rho} \rho^n$$

$$\begin{aligned} E(n) &= P_0 \rho \frac{d}{d\rho} \sum_{n=0}^N \rho^n P_0 \rho \frac{d}{d\rho} \left[\frac{1-\rho^{N+1}}{1-\rho} \right] \\ &= P_0 \frac{\rho[1-(N+1)\rho^n + N\rho^{N+1}]}{(1-\rho)^2} \\ &= \frac{\rho[1-(N+1)\rho^N + N\rho^{N+1}]}{(1-\rho)(1-\rho^{N+1})} \end{aligned}$$

$$\text{Since } P_0 = \frac{1-\rho}{1-\rho^{N+1}}; \rho \neq 1$$

(ii) Average queue length is given by

$$\begin{aligned}
 E(m) &= \sum_{n=1}^N (n-1)P_n = E(n) - \sum_{n=1}^N P_n = E(n) - (1 - P_0) \\
 &= E(n) - \frac{\rho(1-\rho^N)}{1-\rho^{N+1}}, \quad \text{since } P_0 = \frac{1-\rho}{1-\rho^{N+1}}, (\rho \neq 1) \\
 &= \frac{\rho^2[1 - N\rho^{N-1} + (N-1)\rho^N]}{(1-\rho)(1-\rho^{N+1})}
 \end{aligned}$$

(iii) The average waiting time in the system can be obtained by using Little's formula, that is $E(v) = \{E(n)\}/\lambda$ where λ' is the mean rate of customers entering the system and is equal to $(1 - P_N)$. The average waiting time in the queue can be obtained by using the relations

$$E(W) = E(V) - 1/\mu \quad \text{or} \quad E(W) = \{E(m)\}/\lambda'.$$

EXAMPLES

1. At a railway station, only one train is handled at a time. The railway yard is sufficient only for two trains to wait while other is given signal to leave the station. Trains arrive at the station at an average rate of 6 per hour and the railway station can handle them on an average of 12 per hour. Assuming Poisson arrivals and exponential service distribution, find the steady-state probabilities for the various number of trains in the system. Also, find the average waiting time of a new train coming into the yard.

Solution

Here, $\lambda = 6$ and $\mu = 12$ so that $\rho = 6/12 = 1/2 = 0.5$

The maximum queue length is 2, i.e., the maximum number of trains in the system is 3(=N).

The probability that there is no train in the system (both waiting and in service) is given by

$$P_0 = \frac{1 - \rho}{1 - \rho^{N+1}} = \frac{1 - 0.5}{1 - (0.5)^{3+1}} = 0.53$$

Now, since $P_n = P_0 \rho^n$, therefore

$$P_1 = (0.53)(0.5) = 0.27, P_2 = (0.53)(0.5)^2 = 0.13, \text{ and}$$

$$P_3 = (0.53)(0.5)^3 = 0.07$$

Hence, we get

$$E(n) = 1(0.27) + 2(0.13) + 3(0.07) = 0.74$$

Thus the average number of trains in the system is 0.74 and each train takes on an average of $\frac{1}{12}$ (= 0.08) hours for getting service. As the arrival of new train expects to find an average of 0.74 train in the system before it.

$$E(W) = (0.74)(0.08) \text{ hours} = 0.0592 \text{ hours or } 3.5 \text{ minutes}$$

2. Assume that the goods trains are coming in a yard at the rate of 30 trains per day and suppose that the inter-arrival times follow an exponential distribution. The service times for each train is assumed to be exponential with an average of 36 minutes. If the yard can admit 9 trains at a time (there being 10 lines, one of which is reserved for shunting purpose), calculate the probability that the yard is empty and find the average queue length.

Solution: We have

$$\lambda = \frac{30}{60 \times 24} = \frac{1}{48} \quad \text{and} \quad \mu = \frac{1}{36} \text{ trains per minutes}$$

$$\therefore \rho = \lambda / \mu = 36 / 48 = 0.75$$

The probability that the yard is empty is given by

$$P_0 = \frac{1 - \rho}{1 - \rho^{N+1}} = \frac{1 - 0.75}{1 - (0.75)^{10}}, \quad \text{since } N = 9$$

$$= \frac{0.25}{0.90} = 0.28$$

Average queue length is given by

$$\begin{aligned} E(m) &= \frac{\rho^2[1 - N \rho^{N-1} + (N-1) \rho^N]}{(1 - \rho)(1 - \rho^{N+1})} \\ &= \frac{(0.75)^2[1 - 9(0.75)^8 + 8(0.75)^9]}{0.25[(0.75)^{10}]} \\ &= (2.22) \frac{(1 - 0.303)}{(1 - 0.005)} \\ &= (2.22) (0.70) \\ &= 1.55 \end{aligned}$$

Model IV (*Generalized Model : Birth-Death Process*). This model deals with a queueing system having single service channel, Poisson input with no limits on the system capacity . Arrivals can be considered as *births* to the system , whereas a departure can be looked upon as a *death*. Let

N = number of customer in the system

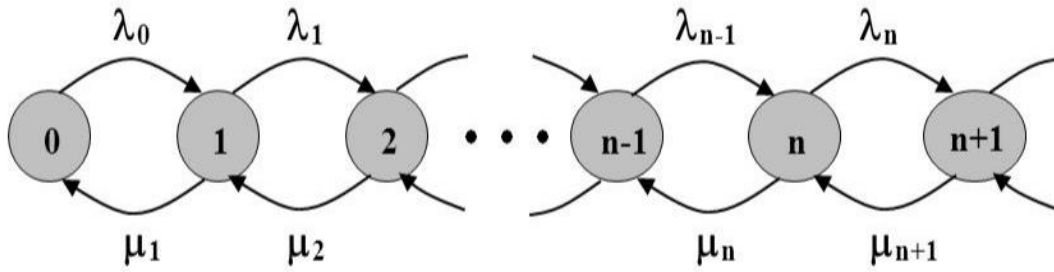
λ_n = arrival rate of the customers given n customers in the system

μ_n = departure rate of customers given n customers in the system, and

P_n = steady-state probability of n customers in the system.

The model determines the values of P_n in terms of λ_n and μ_n . Now, from the axioms of Poisson process , we observe that an arrival during the small time interval Δt is negligible . This implies that for $n > 0$, state n can change only to two possible states : state $n - 1$ when a departure occurs at the rate μ_n and state $n + 1$ when an arrival occurs at rate λ_n . State 0 can only change to state I when an arrival occurs at the rate of λ_o . Since no departure is possible when the system is empty, μ_o is undefined.

Under steady-state conditions, for $n > 0$, the rates of flow into and out of state n must be equal. This is illustrated in the *transition-rate diagram* given below :



The balance equation is :

Expected rate of flow into state n = Expected rate of flow out of state n

$$i.e., \quad \lambda_{n-1}P_{n-1} + \mu_{n+1}P_{n+1} = \lambda_n P_n + \mu_n P_n \quad n \geq 1$$

$$and \quad \mu_1 P_1 = \lambda_0 P_0 \quad n = 0$$

Using the iterative procedure (as in Model I), we have

$$P_1 = \frac{\lambda_0}{\mu_1} P_0, \quad P_2 = \frac{\lambda_1 + \mu_1}{\mu_2} P_1 - \frac{\lambda_0}{\mu_2} P_0 = \frac{\lambda_1 \lambda_0}{\mu_2 \mu_1} P_0$$

$$P_3 = \frac{\lambda_2 + \mu_2}{\mu_3} P_2 - \frac{\lambda_1}{\mu_3} P_1 = \frac{\lambda_2 \lambda_1 \lambda_0}{\mu_3 \mu_2 \mu_1} P_0$$

In general , we can write the following formula

$$P_n = \frac{\lambda_{n-1} \lambda_{n-2} \dots \lambda_0}{\mu_n \mu_{n-1} \dots \mu_1} P_0, \quad n \geq 1 \quad \text{or} \quad P_n = P_0 \prod_{i=0}^{n-1} \frac{\lambda_i}{\mu_{i+1}}, \quad n \geq 1$$

$$Now \quad P_{n+1} = \frac{\lambda_n + \mu_n}{\mu_{n+1}} P_n - \frac{\lambda_{n+1}}{\mu_{n+1}} P_{n-1} = P_0 \prod_{i=0}^n \frac{\lambda_i}{\mu_{i+1}}$$

Thus, by mathematical induction the general value of P_n holds for all n .

To obtain the value of P_0 , we use the boundary condition $\sum_{n=0}^{\infty} P_n = 1$

Or $P_0 + \sum_{n=1}^{\infty} P_n = 1$, to get

$$P_0 = \left(1 + \sum_{n=1}^{\infty} \prod_{i=0}^{n-1} \frac{\lambda_i}{\mu_{i+1}} \right)^{-1}$$

Remark. $P_0 = 0$ if R.H.S is a divergent series . In case R.H.S is convergent , the value of P_0 will depend on λ_i 's and μ_i 's.

Special cases

Case I . When $\lambda_n = \lambda$ for $n \geq 0$; and $\mu_n = \mu$ for $n > 1$

$$P_0 = [1 + \sum_{n=1}^{\infty} (\lambda/\mu)^n]^{-1} = 1 - \rho.$$

In this case ,therefore

$$P_n = \rho^n (1 - \rho), \text{ for } n \geq 0.$$

This result is exactlt the same as that of Model I .

Case II . When $\lambda_n = \frac{\lambda}{n+1}$ for $n \geq 0$ and $\mu_n = \mu$ for $n > 1$.

$$P_0 = \left[1 + \sum_{n=1}^{\infty} \frac{\lambda^n}{n! \mu^n}\right]^{-1} = \left[1 + \rho + \frac{1}{2!} \rho^2 + \frac{1}{3!} \rho^3 + \dots\right]^{-1} = e^{-\rho}$$

$$\therefore P_n = \frac{1}{n!} \rho^n e^{-\rho} \text{ for } n \geq 0 \text{ and } \rho = \frac{\lambda}{\mu}.$$

Which is a poisson distribution with mean $E(n) = \rho$.

Case III . When $\lambda_n = \lambda$ for $n \geq 0$; and $\mu_n = n\mu$ for $n > 1$,

$$P_0 = \left[1 + \sum_{n=1}^{\infty} \frac{\lambda^n}{n! \mu^n}\right]^{-1} = e^{-\rho}$$

$$\therefore P_n = \frac{1}{n!} \rho^n e^{-\rho}, \text{ for } n \geq 0 \text{ and } \rho = \frac{\lambda}{\mu}.$$

Which is again poisson with mean $E(n) = \rho$; and $E(m) = 0$, $E(w) = 0$.

In this case the service rate increases with the increase in queue length and hence is known as a queueing problem with infinite number of channels , i.e., $(M/M/\infty) : (\infty/\text{FIFO})$. This model is known as a *Self-service Model*.

EXAMPLE

Problems arrive at a computing centre in Poisson fashion at an average rate of five per day . The rules of the computing centre are that any man waiting to get his problem solved must aid the man whose problem is being solved. If the time to solve a problem with one man has an exponential distribution with mean time of $\frac{1}{3}$ day , and if the

average solving time is inversely proportional to the number of people working on the problem, approximate the expected time in the centre of a person entering the line.

Solution. Here $\lambda = 5$ problems per day, and $\mu = 3$ problems per day.

It is given that the service rate increases with the increase in the number of person.

$\therefore \mu_n = n\mu$ when there are n problems and $P_n = \frac{1}{n!} \rho^n e^{-\rho}$

$$E(n) = \sum_{n=0}^{\infty} n p_n = \sum_{n=0}^{\infty} n \frac{1}{n!} \rho^n e^{-\rho} = e^{-\rho} \cdot \rho \cdot e^{\rho} = \rho = \frac{5}{3} \text{ or } 1.67$$

Now, the average solving time, which is inversely proportional to the number of people working on the problem, is given by $1/5$ day per problem.

\therefore Expected time for a person entering the line is given by

$$\frac{1}{5} E(n) = \frac{1}{5} \times \frac{5}{3} \text{ days} = \frac{1}{3} \text{ days or } 8 \text{ hours.}$$

Model V {(M/M/C) : (∞ / FIFO)}. This model is a special case of Model IV in the sense that here we consider C parallel service channels. The arrival rate is λ and the service rate per service channel is μ .

The effect of using C parallel service channel is a proportionate increase in the service rate of the facility to $n\mu$ if $n \leq C$ and $C\mu$ if $n > C$. Thus, in terms of the generalized model (Model IV), λ_n and μ_n are defined as

$$\lambda_n = \lambda, \quad n \geq 0$$

and $\mu_n = n\mu$ if $1 \leq n \leq C$ and $C\mu$, if $n \geq C$.

Utilizing the above values of λ_n and μ_n , the steady – state probability of Model IV becomes

$$P_n = \begin{cases} \frac{\lambda^n P_0}{n\mu(n-1)\dots(1)\mu}; & 1 \leq n \leq C, \\ \frac{\lambda^n P_0}{\underbrace{(C\mu)(C\mu)\dots(C\mu)}_{C!} (C-1)\mu(C-2)\mu\dots(1)\mu}; & n > C \end{cases}$$

$$= \frac{\lambda^n P_0}{n! \mu^n} \quad \text{if } 1 \leq n \leq C \quad \text{and} \quad \frac{\lambda^n P_0}{C^{n-C} C! \mu^n} \quad \text{if } n > C$$

$$= \frac{1}{n!} \rho^n P_0 \quad \text{if } 1 \leq n \leq C \quad \text{and} \quad \frac{1}{C^{n-C} C! \mu^n} \rho^n P_0 \quad \text{if } n > C$$

To find the value of P_0 , we use the boundary condition $\sum_{n=0}^{\infty} P_n = 1$

$$\sum_{n=0}^{C-1} P_n + \sum_{n=C}^{\infty} P_n = 1$$

Or
$$\left[\sum_{n=0}^{C-1} \frac{1}{n!} \rho^n + \sum_{n=C}^{\infty} \frac{1}{C^{n-C} C!} \rho^n \right] P_0 = 1$$

Or
$$P_0 = \left[\sum_{n=0}^{C-1} \frac{1}{n!} \rho^n + \rho^C \sum_{n=C}^{\infty} \frac{1}{C^{n-C} C!} \left(\frac{\rho}{C} \right)^{n-C} \right]^{-1}$$

$$= \left[\sum_{n=0}^{C-1} \frac{1}{n!} \left(\frac{\lambda}{\mu} \right)^n + \frac{1}{C!} \left(\frac{\lambda}{\mu} \right)^C \cdot \frac{C\mu}{C\mu - \lambda} \right]^{-1}$$

Remark. The result obtained above is valid only if $\frac{\lambda}{C\mu} < 1$; that is, the mean arrival rate must be less than the mean maximum potential service rate of the system. If $C = 1$, then the value of P_0 is in complete agreement with the value of P_0 for Model I.

Characteristics of Model V

(i) $P(n \geq C)$ = Probability that an arrival has to wait

$$= \sum_{n=C}^{\infty} P_n = \sum_{n=C}^{\infty} \frac{1}{C^{n-C} C!} (\lambda/\mu)^n P_0 = \frac{(\lambda/\mu)^C C\mu}{C!(C\mu - \lambda)} P_0$$

(ii) Probability that an arrival enters the service without wait

$$= 1 - P(n \geq C) \quad \text{or} \quad 1 - \frac{C(\lambda/\mu)^C}{C!(C - \lambda/\mu)} P_0$$

(iii) Average queue length is given by

$$E(m) = \sum_{n=C}^{\infty} (n - C) P_n = \sum_{x=0}^{\infty} x P_{x+C}, \quad \text{for } x = n - C$$

$$= \sum_{x=C}^{\infty} x \cdot \frac{1}{C! C^x} \left(\frac{\lambda}{\mu} \right)^{C+x} P_0$$

$$E(m) = \frac{1}{C!} (\lambda/\mu)^C \sum_{x=0}^{\infty} x \cdot \left(\frac{\lambda}{C\mu} \right)^x P_0$$

$$= \frac{1}{C!} (\lambda/\mu)^C P_0 \sum_{x=0}^{\infty} \left(\frac{d}{dy} y^x \right) \cdot y, \quad \text{where } y = \frac{\lambda}{C\mu}$$

$$= \frac{1}{C!} (\lambda/\mu)^C P_0 y \frac{d}{dy} \left(\frac{1}{1-y} \right)$$

$$= \frac{\lambda \mu \left(\frac{\lambda}{\mu} \right)^C P_0}{(C-1)!(C\mu - \lambda)^2}$$

(iv) Average number of customer in the system is given

$$E(n) = E(m) + \frac{\lambda}{\mu} = \frac{\lambda \mu \left(\frac{\lambda}{\mu} \right)^C P_0}{(C-1)!(C\mu - \lambda)^2} + \frac{\lambda}{\mu}.$$

(v) Average waiting time of an arrival is given by

$$E(w) = \frac{1}{\lambda} E(m) = \frac{\mu \left(\frac{\lambda}{\mu}\right)^C P_0}{(C-1)!(C\mu-\lambda)^2}$$

(vi) Average waiting time an arrival spends in the system is given by

$$E(v) = E(w) + \frac{1}{\mu} = \frac{\mu \left(\frac{\lambda}{\mu}\right)^C P_0}{(C-1)!(C\mu-\lambda)^2} + \frac{1}{\mu} \quad \text{or } E(v) = E(n)/\lambda.$$

(vii) Average number of idle servers is equal to

$$C - \text{Average number of customers served.}$$

EXAMPLE

A Supermarket has two girls serving at the counters. The customers arrive in a poisson fashion at the rate of 12 per hour. The service time for each customer is exponential with mean 10 minutes .Find

(i) the probability that an arriving customer has to wait for service,

(ii) the average number of customers in the system, and

(iii) the average time spent by a customer in the super-market

Solution. We are given

$$\lambda = 12 \text{ customer per hour, } \mu = 10 \text{ per hour, } C = 2 \text{ girls.}$$

$$\therefore P_0 = \left[\sum_{n=0}^{C-1} \frac{1}{n!} \left(\frac{\lambda}{\mu}\right)^n + \frac{1}{C!} \left(\frac{\lambda}{\mu}\right)^C \cdot \frac{C\mu}{C\mu-\lambda} \right]^{-1} = \frac{1}{4} \quad (\text{or } 0.25)$$

(i) Probability of having to wait for service

$$\begin{aligned} P(w > 0) &= \frac{1}{C!} \left(\frac{\lambda}{\mu}\right)^C \frac{C\mu}{(C\mu-\lambda)} P_0 \\ &= \frac{1}{2} \left(\frac{12}{10}\right)^2 \frac{20}{(20-12)} \times \frac{1}{4} = 0.45 \end{aligned}$$

(ii) Average queue length is

$$E(m) = \frac{\lambda \mu \left(\frac{\lambda}{\mu}\right)^C P_0}{(C-1)!(C\mu-\lambda)^2} = \frac{12 \times 10 \times (1.2)^2 \times 0.25}{(2-1)!(20-12)^2} = \frac{27}{40}$$

Average number of customer in the system

$$E(n) = E(m) + \frac{\lambda}{\mu} = \frac{27}{40} + \frac{12}{10} = 1.82 \text{ (or 2 customers) approx..}$$

(iii) Average time spent by a customer in supermarket

$$E(v) = E(n)/\lambda = 1.82/12 = 0.156 \text{ hours or } 9.3 \text{ minutes.}$$

MODEL VI:{(M/M/C):(N/FIFO)}. This model is essentially the same as model V except that the maximum number in the system is limited to N where $N \geq C$. Therefore, utilizing the steady-state probabilities of model IV, with

$$\lambda_n = \lambda \quad \text{if} \quad 0 \leq n < N; \quad \text{and} \quad 0 \text{ otherwise}$$

$$\text{And} \quad \mu_n = n\mu \quad \text{if} \quad 0 \leq n < C; \quad \text{and} \quad C\mu \text{ if } C \leq n < N$$

We get

$$P_n = \begin{cases} \frac{1}{n!} \left(\frac{\lambda}{\mu}\right)^n P_0; & 0 \leq n < C \\ \frac{1}{C^{n-C} C!} \left(\frac{\lambda}{\mu}\right)^n P_0; & C \leq n \leq N \end{cases}$$

$$\begin{aligned} \text{Where} \quad P_0 &= \left\{ \sum_{n=0}^{C-1} \frac{1}{n!} \left(\frac{\lambda}{\mu}\right)^n + \sum_{n=C}^N \frac{1}{C^{n-C} C!} \left(\frac{\lambda}{\mu}\right)^n \right\}^{-1} \\ &= \begin{cases} \left[\sum_{n=0}^{C-1} \left(\frac{\lambda}{\mu}\right)^n + \frac{1}{C!} \left(\frac{\lambda}{\mu}\right)^C \left\{ 1 - \left(\frac{\lambda}{C\mu}\right)^{N-C+1} \right\} \frac{C\mu}{C\mu - \lambda} \right]^{-1} \\ \left[\sum_{n=0}^{C-1} \left(\frac{\lambda}{\mu}\right)^n + \frac{1}{C!} \left(\frac{\lambda}{\mu}\right)^C (N - C + 1) \right]^{-1} ; & \frac{\lambda}{C\mu} = 1 \end{cases} \end{aligned}$$

Remark. If we take $N \rightarrow \infty$ and consider $\lambda/C\mu < 1$, then the reduced result corresponds to that of Model V. Also, if we take $C = 1$ then the reduced result corresponds to that of Model I.

Characteristics of model VI

(i) Average queue length is given by

$$E(m) = \sum_{n=C}^N (n - C) P_n = \sum_{n=C}^N (n - C) \frac{\left(\frac{\lambda}{\mu}\right)^n}{C! C^{n-C}} P_0$$

(ii) Average number of customers in the system is given by

$$E(n) = E(m) + C - P_0 \sum_{n=0}^{C-1} \frac{(C-n)(\rho C)^n}{n!}$$

(iii) Average waiting time in the system can be obtained by using Little's formula, that is,

$E(v) = [E(n)]/\lambda'$ where $\lambda' = \lambda(1 - P_N)$ is the effective arrival rate.

Average waiting time in a queue can be obtained by using

$$E(w) = E(v) - 1/\mu \text{ or } E(w) = [E(m)]/\lambda'.$$

EXAMPLE

A car servicing station has 3 stalls where service can be offered simultaneously. The cars wait in such a way that when a stall becomes vacant, the car at the head of the line pulls up to it. The station can accommodate at most four cars waiting (seven in the station) at one time. The arrival pattern is Poisson with a mean of one car per minute during the peak hours. The service time is exponential with mean 6 minutes. Find the average number of cars in the service station during peak hours, the average number of cars per hour that cannot enter the station because of full capacity.

Solution. Here $\lambda = 1$ car per minute, $\mu = 1/6$ car per minute, $C = 3$,

$N = 7$, $\rho = \lambda/\mu = 6$ and

$$P_0 = \left[\sum_{n=0}^{3-1} \frac{1}{n!} 6^n + \sum_{n=3}^7 \frac{1 \times 6^n}{3^{n-3} 3!} \right]^{-1}$$

Expected number of cars in the queue is

$$\begin{aligned} E &= \frac{(C\rho)^C}{C!(1-\rho)^2} (1 - \rho^{N-C+1} - (1 - \rho)(N - C + 1)\rho^{N-C}) \\ &= \frac{(3 \times 6)^3 \times 6}{3!(-5)^2} \cdot \frac{1}{1141} (1 - 6^5 - (-5)(5)(6)^4) \\ &= 3.09 \text{ Cars} \end{aligned}$$

Expected number of cars in the service station

$$E(n) = 3.09 + 3 - P_0 \sum_{n=0}^2 \frac{(3-n)}{n!} (6)^n = 6.06 \text{ Cars}$$

Expected waiting time a car spends in the system

$$E(v) = \frac{6.06}{1(1-P_7)} = \frac{0.06}{1 - \frac{6^7}{3!3^4} \times \frac{1}{1141}} = 0.121$$

$$\text{Since, } P_n = \frac{1}{C!C^{n-C}} \left(\frac{\lambda}{\mu}\right)^n P_0 \text{ for } C \leq n \leq N$$

Expected number of cars per hour that cannot enter the station is

$$60\lambda P_N = 60 \times 1 \times P_7 = 60 \times \frac{6^7}{3!3^4} \times \frac{1}{1141} = 30.3 \text{ cars per hour}$$

Model VII {M/M/C} : {C/FIFO}. This model is essentially the same as Model VI except that here $N = C$. Therefore, we consider the situation where no waiting queue is allowed to form. This gives rise to stationary distribution known as Erlang's first formula and can be easily obtained by using the result of Model VI with $N = C$. Thus, we have

$$P_n = \frac{1}{n!} \left(\frac{\lambda}{\mu}\right)^n P_0 \text{ if } 0 \leq n \leq C \text{ and } 0 \text{ otherwise}$$

Where
$$P_0 = \left[\sum_{n=0}^C \frac{1}{n!} \left(\frac{\lambda}{\mu}\right)^n \right]^{-1}$$

The resultant formula for P_0 is itself called Erlang's Loss Formula.

EXAMPLE

A tax consulting firm has three counters in its office to receive people who have problem concerning their income, wealth and sales taxes. On the average 48 persons arrive in an 8 hour in a day. Each tax adviser spends 15 on an average on an arrival. If the arrivals are poissonly distribution and the service time are according to exponential distribution,

Find

- 1) The average number of customers in system
- 2) Average number of customers to be served
- 3) Average time a customer spends in the system

Solution:

Here $C=3$, $\lambda=48/8=6$ per hour

$\mu=\frac{1}{15} \times 60=4$ per hour

$$\begin{aligned} \text{Probability } P_0 &= \left[\sum_{n=0}^{C-1} \frac{1}{n!} \left(\frac{\lambda}{\mu}\right)^n + \frac{1}{C!} \left(\frac{\lambda}{\mu}\right)^C \frac{C\mu}{(C\mu-\lambda)} \right]^{-1} \\ &= \left[\sum_{n=0}^{C-1} \frac{1}{n!} \left(\frac{\lambda}{\mu}\right)^n + \frac{1}{3!} \left(\frac{\lambda}{\mu}\right)^3 \frac{3\mu}{(3\mu-\lambda)} \right]^{-1} \\ &= \frac{1}{\left[1 + \frac{\lambda}{\mu} + \frac{1}{2} \left(\frac{\lambda}{\mu}\right)^2 + \frac{(\lambda/\mu)^2}{6} \cdot \frac{3\mu}{3\mu-\lambda} \right]} \\ &= \frac{1}{\left[1 + \frac{3}{2} + \frac{1}{2} \left(\frac{3}{2}\right)^2 + \frac{(3/2)^2}{6} \cdot \frac{3 \times 4}{3 \times 4 - 6} \right]} \end{aligned}$$

$$\begin{aligned}
&= \frac{1}{\left[1 + \frac{3}{2} + \frac{9}{8}\right] + \frac{27}{48} \cdot \frac{12}{12-6}} \\
&= \frac{1}{\frac{29}{8} + \frac{9}{8}} \\
&= \frac{8}{39} \\
&= 0.21
\end{aligned}$$

1) Average number of customers in the system,

$$\begin{aligned}
L_s &= \frac{\lambda_\mu (\lambda/\mu)^c}{(c-1)!(c_\mu - \lambda)^2} \cdot P_0 + \frac{\lambda}{\mu} \\
&= \frac{6 \times 4 \times (3/2)^2}{2! (12 - 6)^2} \times (0.21) + \frac{3}{2} \\
&= 1.74
\end{aligned}$$

2) Average number of customer waiting to be served:

$$\begin{aligned}
L_q &= L_s - \frac{\lambda}{\mu} \\
&= 1.74 - \frac{3}{2} \\
&= 0.24
\end{aligned}$$

3) Average time a customer spends in the system:

$$\begin{aligned}
W_s &= \frac{L_s}{\lambda} = \frac{1.74}{6} \\
&= 0.29 \text{ hour} \\
&= 17.4 \text{ minutes}
\end{aligned}$$

LIMITATION OF QUEUING THEORY

- The problem resolving is based on mathematical distributions and assuming (the client's behaviour is predicted, but no one guarantees the 100 per cent accurateness).
- Situations that take place in real life are usually complex and get beyond the philosophy and mathematics, which means that doubt remains no matter how accurate you are
- Many companies have multi-channel services when one client has to receive services from several operators, and this can mean that customers

would often have to fall in a new queue soon after they get out of the previous one

- It takes much effort, time and energy to analyse a particular situation and solve the problem using the theory (this method is expensive).

APPLICATION OF QUEUING THEORY

(i)Application in communication system

Applicability of queueing theory through Markov process is also found in the field of communication system. This chain is based on the condition that the past, present and future all of them are independent. The natural laws of jump chain done within Markov chain process is also one of the examples of the queueing theory in communication system.

(ii)Applications in Health Care Systems

Queueing theory is “The mathematical approach to the analysis of waiting lines in Health care setting”. Queueing system is very beneficial in the health care systems as well. One of the biggest hurdles in health care organizations is the fact that patients have to wait in long queues for their turn to be assisted. Queueing system minimizes the time that customers have to waste in waiting and utilizing their resources and servers. These servers include the nurses, hospital beds, doctors and other health care services. When a person chose to stop waiting in a queue, he complies with the phenomenon of reneging. This decision is dependent on the length of the queue and the amount of stamina that a patient has to wait in a line. Health care organizations attain dysfunctional equilibrium through exceeding server capacity by reneging. This example can be understood through the example of emergency units in the hospital (Tian & Zhang, n.d). Most of the patients quit emergency departments without even getting treated for their health problem due to capacity, arrival rate and utilization. Statistics and data collected from this amount of number of people leaving, health care organizations determine the rate of revenue loss. The Same queueing method can also be utilized to minimize the reneging factor in health care organizations. One way of doing this is by categorizing patients according to the service they require. Also we use the telecommunication system to avoid queue length by reserving previously appointment to consult a doctor.

CONCLUSION

With the knowledge of probability theory, input and output models, and birth-death processes, it is possible to derive many different queuing models, including but not limited to the ones we observed in this paper. Queuing theory can be applicable in many real-world situations. For example, understanding how to model a multiple-server queue could make it possible to determine how many servers actually needed and at what wage in order to maximize financial efficiency. Or perhaps a queuing model could be used to study the lifespan of the bulbs in street lamps in order to better understand how frequently they need to be replaced.

The applications of queuing theory extend well beyond waiting in line at a bank. It may take some creative thinking, but if there is any sort of scenario where time passes before a particular event occurs, there is probably some way to develop it into a queuing model. Queues are so commonplace in society that it is highly worthwhile to study them, even if only to shave a few seconds off one's wait in the checkout line.

REFERENCE

- 1.**Samuel Fomundam, Jeffrey Herrmann (2007) A survey of Queuing theory applications in Healthcare.
- 2.**Operations Research by Kanti Swarup ,P K Gupta , Man Mohan , Priynshu Gupta.
- 3.** Wayne L Winston, Operations Research: Applications and Algorithms, 2ndedition, PWS-Kent Publishing, Boston, 1991.

RECURRENCE RELATIONS ON COMBINATORICS

Project report submitted to

ST.MARY'S COLLEGE (AUTONOMOUS),THOOTHUKUDI.

Affiliated to

MANONMANIAM SUNDARANAR UNIVERSITY, TIRUNELVELI

In partial fulfillment of the requirement for the award of degree of

Bachelor of Science in Mathematics

Submitted by

NAME
DasnavisRobanci.A
Dilany.A
Sangavi.P
Selliammal.R
Spica.J

REG.NO
19AUMT09
19AUMT10
19AUMT38
19AUMT41
19AUMT45

Under the guidance of

Dr. V.L.STELLA ARPUTHA MARY M.sc.,M.Phil.,B.Ed.,Ph.D.,

Head & Assistant Professor of Mathematics

St. Mary's College (Autonomous), Thoothukudi.



Department of Mathematics

St. Mary's College (Autonomous), Thoothukudi

(2021 - 2022)

CERTIFICATE

We hereby declare that the project report entitled "RECURRENCE RELATIONS ON COMBINATORICS" being submitted to St. Mary's College (Autonomous), Thoothukudi affiliated to Manonmaniam Sundaranar University, Tirunelveli in partial fulfillment for the award of degree of Bachelor of Science in Mathematics and it is a record of work done during the year 2021 - 2022 by the following students:

NAME	REG.NO.
DASNAVIS ROBANCIA	19AUMT09
DILANY.A	19AUMT10
SANGAVI.P	19AUMT38
SELLIAMMAL.R	19AUMT41
SPICA.J	19AUMT45

V. Sr. Stella Arputha Mary
Signature of the Guide

SAD
Signature of the Examiner

V. Sr. Stella Arputha Mary
Signature of the HOD
Dr. V. Sr. Stella Arputha Mary
M. Sc., M. Phil., B. Ed., Ph. D.,
Head & Professor of Mathematics
St. Mary's College (Autonomous)
Thoothukudi-628 001.
Lucia Rose
Signature of the Principal
St. Mary's College (Autonomous)
Thoothukudi - 628 001.

DECLARATION

We hereby declare that the project entitled
“**RECURRENCE RELATION ON COMBINATORICS**” is our original work.
It is not been submitted to any University for any degree or diploma.

R. Selliammal
(**Selliammal.R**)

Dasnavis Robanci.A
(**Dasnavis Robanci.A**)

Spica.J
(**Spica.J**)

A. Dilany
(**Dilany.A**)

Sangavi.P
(**Sangavi.P**)

ACKNOWLEDGEMENT

First of all, we thank Lord Almighty for showering his blessings to undergo this project.

With immense pleasure, we register our deep sense of gratitude to our guide and the Head of the Department, **Dr. V. L. Stella Arputha Mary M.Sc., M.Phil., B.Ed., Ph.D.** for having imparted necessary guidelines throughout the period of our studies.

We thank our beloved Principal, **Rev. Dr. Sr. A.S.J. Lucia Rose M.Sc., M.Phil., Ph.D., PGDCA** for providing us the help to carry out our project work successfully.

Finally, we thank all those who extended their helping hands regarding this project.

**RECURRENCE RELATION
ON
COMBINATORICS**

content

Introduction & Definition

1.1 First order linear Recurrence relation

1.1.1 Fibonacci sequence

1.1.2 Tower of Hanoi

1.2 Method of linear Recurrence relation

1.2.1 Back Tracking method

1.2.2 Forward chaining method

1.2.3 Summation method

1.3 Second order linear Recurrence relation

1.4 The Non-Homogeneous Recurrence relation

1.4.1 Characteristics equation

1.5 Recurrence relation using generating Function.

Reference

INTRODUCTION:

A wide variety of recurrence relations occur in models. Some of these recurrence relations can be solved using iteration on some other adhoc technique. However, one important class of recurrence relation can be explicitly solved in a systematic way. There are recurrence relations that express the terms of a sequence as linear combinations of previous terms.

This study of what are called either recurrence relations on difference equations is the discrete counterpart to ideas applied in ordinary differential equations.

Our development will not employ any ideas from differential equations but will start with the notion of a geometric progression. As further ideas are developed, we shall see some of the many applications that make this topic so important.

A recurrence relation is an equation that uses recursion to relate terms in a sequence an array. It is a way to define a sequence on array in terms of itself .Recurrence relations have applications in many areas of mathematics number theory- the Fibonacci sequences.

Definition:

A **recurrence relation** for the sequence $\{a_n\}$ is an equation that expresses a_n in terms of one or more preceding terms of the sequence, viz, a_0, a_1, \dots, a_{n-1} , for $n \geq n_0$. Here n_0 is used to define initial condition and is a non-negative integer.

A sequence is called as a solution of a recurrence relation if its terms satisfy the recurrence relation.

The initial conditions for a sequence specify the terms that precede the first term where the recurrence relation takes effect.

1.1 THE FIRST-ORDER RECURRENCE RELATION

Suppose n is a natural number, we define 2^n as

$$2^n = \underbrace{2.2.2.....2}_{n \text{ 2's}}$$

or

$$2' = 2, \text{ and for } k \geq 1, 2^{k+1} = 2.2^k$$

We write $0! = 1$ and for $k \geq 0$, $(k+1)! = (k+1)k!$.

A sequence is a function whose domain is some infinite set of integers (often \mathbb{N}) and whose range is a set of real numbers.

The sequence which is the function $f : \mathbb{N} \rightarrow \mathbb{R}$ defined by $f(n) = n^2 = 1, 4, 9, 16, \dots$ (1)

The numbers in the list are called the terms of sequence, the terms are denoted a_0, a_1, a_2, \dots

The sequence 2, 4, 8, 16, can be defined recursively like : $a_1 = 2$ and for $f \geq 1$, $a_{k+1} = 2a_k$ setting $k = 1, 2, 3, \dots$ and $a_1 = 2$ in (1) gives 2, 4, 8,

The equation $a_{k+1} = 2a_k$ in (1), which defines one member of the sequence in terms of a previous one, is called a **recurrence relation**. The equation $a_1 = 2$ is called an initial condition.

For example, we write ,

$$a_0 = 2 \text{ and for } k \geq 0, a_{k+1} = 2a_k \text{ or we say } a_1 = 2 \text{ and for } k \geq 2, a_k = 2a_{k-1} .$$

In (1), for instance, $a_n = 2^n$, we say that , $a_n = 2^n$ is the solution to the recurrence relation.

A sequence of numbers like 50, 64, 78, 92, where each term is determined by adding the same fixed number to the previous one, is called an arithmetic sequence. The fixed number is called the common difference of the sequence.

The arithmetic sequence with first term a and common difference d is the sequence defined by

$$a_1 = a \text{ and } k \geq 1, a_{k+1} = a_k + d$$

The general arithmetic sequence, takes the form

$$a, a + d, a + 2d, \dots$$

and for $n \geq 1$, the n^{th} term of the sequence is $a_n = a + (n - 1) d$.

The sum of n terms of the arithmetic sequence with first term a and common difference d is

$$S = \frac{n}{2} [2a + (n - 1) d]$$

The geometric sequence with first term a and common ratio r is the sequence defined by

$$a_1 = a \text{ and for } k \geq 1, a_{k+1} = ra_k$$

The general geometric sequence, this has the form

$$a, ar, ar^2, ar^3, \dots$$

the n^{th} term being $a_n = ar^{n-1}$, the sum S of n terms ($r \neq 1$), $S = a(1 - r^n)/(1 - r)$

1.1.1 THE FIBONACCI SEQUENCE

The Fibonacci sequence,

$f_1 = 1, f_2 = 1$ and for $k \geq 2, f_{k+1} = f_k + f_{k-1}$ the n th term of the Fibonacci sequence is the closed integer to the number











$$\frac{1}{\sqrt{5}} \left(\frac{1 + \sqrt{5}}{2} \right)^n$$

For example, $a_1 = 1$ and for $k > 1$

$$a_k = \begin{cases} 1 + a_{\frac{k}{2}} & \text{if } k \text{ is even} \\ 1 + a_{3k-1} & \text{if } k \text{ is odd} \end{cases}$$

Problem 1(Rabbits and the Fibonacci numbers)

Consider this problem, which was originally posed by Leonardo di pisa, also known as Fibonacci in the thirteenth century in this book liber abaci. A young pair of rabbits(one of each sex) is placed on an island. A pair of rabbits does not breed until they are 2 months old. After they are 2 months old, each pair of rabbits produces another pair each month. Find a recurrence relation for the number of pair of rabbits on the island after n months, assuming that no rabbits over die.

Reproducing pairs (at least two months old)	Young pairs (less than two months old)	Month	Reproducing pairs	Young pairs	Total pairs
		1	0	1	1
		2	0	1	1
		3	1	1	2
		4	1	2	3
		5	2	3	5
		6	3	5	8

Solution.

Denote by f_n the number of pair of rabbits after n months. We will show that f_n , $n=1,2,3,\dots$ are the terms of the Fibonacci sequence.

The rabbit population can be modeled using a recurrence relation. At the end of the first month the number of pairs of rabbits on the island is $f_1 = 1$. Since this pair does not breed during the second month $f_2 = 1$ also. To find the number of pairs after n months, add the number on the island the previous month, f_{n-1} and the number of new born pairs which equals f_{n-2} , Since each new born pair comes from a pair of least 2 months old.

Consequently the sequence $\{f_n\}$ satisfies the recurrence relation and the initial conditions uniquely determine this sequence the number of pairs of rabbits on the island after n months is given by the n^{th} fibonacci number.

Problem 2

A person invests Rs. 10,000/- @ 12% interest compounded annually. How much will be there at the end of 15 years.

Solution.

Let A_n represents the amount at the end of n years.

So at the end of $n - 1$ years, the amount is A_{n-1} .

Since the amount after n years equals the amount after $n - 1$ years plus interest for the n th year.

Thus the sequence $\{A_n\}$ satisfies the recurrence relation

$$A_n = A_{n-1} + (0.12) A_{n-1} = (1.12) A_{n-1}, n \geq 1.$$

With initial condition $A_0 = 10,000$.

The recurrence relation with the initial condition allow us to compute the value of A_n for any n .

$$\begin{aligned}\text{For example,} \quad A_1 &= (1.12) A_0 \\ A_2 &= (1.12) A_1 = (1.12)^2 A_0 \\ A_3 &= (1.12) A_2 = (1.12)^3 A_0 \\ &\vdots \\ &\vdots \\ A_n &= (1.12)^n A_0\end{aligned}$$

which is an explicit formula and the required amount can be derived from the formula by putting $n = 15$.

$$\text{So, } A_{15} = (1.12)^{15} (10000).$$

Problem 3

Suppose that a person deposits \$10,000 in a savings account at a bank yielding 11% per year with interest compounded annually. How much will be in the account after 30 years ?

Solution.

To solve this problem. Let P_n denote the amount in the account after n years.

Since the amount in the account after n years equals the amount in the account after $n - 1$ years plus interest for the n^{th} year, we see that sequence $\{P_n\}$ satisfies the recurrence relation

$$P_n = P_{n-1} + 0.11 P_{n-1} = (1.11) P_{n-1}$$

This initial condition is $P_0 = 10,000$.

We can use an iterative approach to find a formula for P_n .

Note that $P_1 = (1.11) P_0$

$$P_2 = (1.11) P_1 = (1.11)^2 P_0$$

$$P_3 = (1.11) P_2 = (1.11)^3 P_0$$

.....

.....

$$P_n = (1.11) P_{n-1} = (1.11)^n P_0$$

when we insert the initial condition $P_0 = 10,000$, the formula $P_n = (1.11)^n 10,000$ is obtained.

We can use mathematical induction to establish its validity. That the formula is valid for $n = 0$ is a consequence of the initial condition.

Now assume that $P_n = (1.11)^n 10,000$.

Then, from the recurrence relation and the induction hypothesis.

$$P_{n+1} = (1.11) P_n = (1.11) (1.11)^n 10,000 = (1.11)^{n+1} 10,000.$$

This shows that the explicit formula for P_n is valid.

Inserting $n = 30$ into the formula $P_n = (1.11)^n 10,000$

Shows that after 30 years the account contains $P_{30} = (1.11)^{30} 10,000 = \$228,922.97$

Problem 4

Solve the recurrence relation $a_n = 7 \cdot a_{n-1}$ for $n \geq 1$ given that $a_2 = 98$

Solution.

Given recurrence relation is $a_n = 7 \cdot a_{n-1}$ for $n \geq 1$ $\rightarrow (1)$

The given recurrence relation is a first order linear or homogeneous linear relation.

The general solution of first order linear or homogeneous recurrence relation of

$$a_n = c^n \cdot a_0 \quad \rightarrow (2)$$

In eq (1) substituting $n+1$ in place of n , $a_{n+1} = 7 \cdot a_{n+1-1}$

$$a_{n+1} = 7 \cdot a_n \quad \rightarrow (3)$$

If $a_n = c^n$ then $a_{n+1} = c^{n+1}$

$$7 \cdot a_n = c^n \cdot c \quad [a^{m+n} = a^m \cdot a^n]$$

$$7 \cdot c^n = c^n \cdot c$$

$$C = 7$$

Substituting c value in eq (2), $a_n = 7^n \cdot a_0$ $\rightarrow (4)$

$$a_2 = 98$$

Substituting $n=2$ in eq (4)

$$a_2 = 7^2 \cdot a_0$$

$$98 = 7^2 \cdot a_0$$

$$a_0 = \frac{98}{49} = 2$$

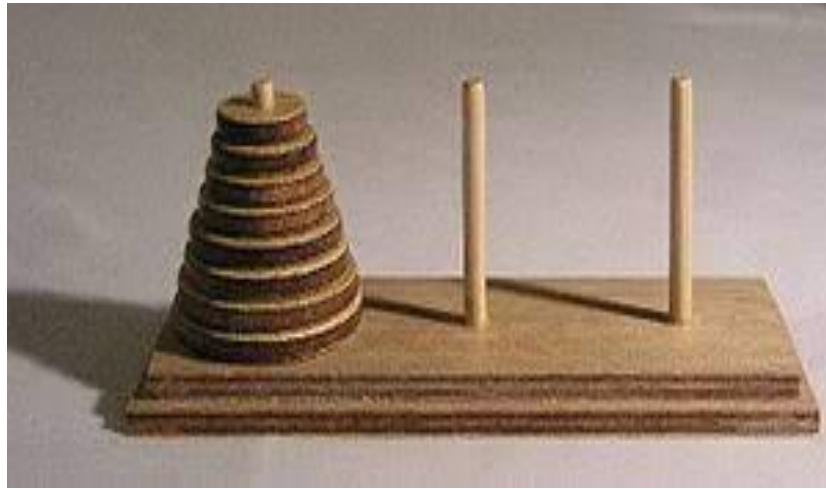
Substitute a_0 value in eq (4)

$$a_n = 7^n \cdot 2$$

1.1.2 TOWER OF HANOI

Problem 5

The game of Hanoi Tower is to play with a set of disks of graduated size with holes in their centers and a playing board having three spokes for holding the disks.



The object of the game is to transfer all the disks from spoke A to spoke C by moving one disk at a time without placing a larger disk on top of a smaller one. What is the minimal number of moves required when there are n disks?

Solution.

Let a_n be the minimum number of moves to transfer n disks from one spoke to another. In order to move n disks from spoke A to spoke C, one must move the first $n - 1$ disks from spoke A to spoke B by a_{n-1} moves, then move the last (also the largest) disk from spoke A to spoke C by one move, and then remove the $n - 1$ disks again from spoke B to spoke C by a_{n-1} moves. Thus the total number of moves should be

$$a_n = a_{n-1} + 1 + a_{n-1} = 2a_{n-1} + 1.$$

This means that the sequence $\{a_n \mid n \geq 1\}$ satisfies the recurrence relation

$$\begin{cases} a_n = 2a_{n-1} + 1, n \geq 1 \\ a_1 = 1 \end{cases} \quad (1)$$

Applying the recurrence relation again and again, we have

$$\begin{aligned} a_1 &= 2a_0 + 1 \\ a_2 &= 2a_1 + 1 = 2(2a_0 + 1) + 1 \\ &= 2^2 a_0 + 2 + 1 \\ a_3 &= 2a_2 + 1 = 2(2^2 a_0 + 2 + 1) + 1 \\ &= 2^3 a_0 + 2^2 + 2 + 1 \\ a_4 &= 2a_3 + 1 = 2(2^3 a_0 + 2^2 + 2 + 1) + 1 \\ &= 2^4 a_0 + 2^3 + 2^2 + 2 + 1 \\ &\vdots \\ a_n &= 2^n a_0 + 2^{n-1} + 2^{n-2} + \dots + 2 + 1 \\ &= 2^n a_0 + 2^n - 1 \end{aligned}$$

Let $a_0 = 0$. The general term is given by

$$a_n = 2^n - 1, n \geq 1.$$

Given a recurrence relation for a sequence with initial conditions. Solving the recurrence relation means to find a formula to express the general term a_n of the sequence.

1.2 METHOD OF LINEAR RECURRENCE RELATION

1.2.1. Back Tracking Method

In this method, we shall start from a_n and move backward towards a_1 to find a pattern, if any, to solve the problem.

To backtrack, we keep on substituting the definition of a_n , a_{n-1} , a_{n-2} and so on. Until a recognizable pattern appears.

1.2.2. Forward Chaining Method

In this method, we begin from initial (terminating) condition and keep on moving towards the n^{th} term until we get a clear pattern.

1.2.3. Summation Method

To solve a first order linear recurrence relation with constant coefficient. In this method, we arrange the given equation in the following form :

$a_n - k a_{n-1} = f(n)$ and then backtrack till terminating condition.

In the process, we get a number of equations. Add these equations in such a way that all intermediate terms get cancelled. Finally, we get the required solution.

Problem 6

Solve the recurrence equation $a_n = a_{n-1} + 3$ with $a_1 = 2$.

Solution.

Backtracking Method :

We have,

$$a_n = a_{n-1} + 3 \text{ with } a_1 = 2$$

$$\begin{aligned} a_n &= a_{n-2} + 3 + 3 \text{ (since } a_{n-1} = a_{n-2} + 3) \\ &= a_{n-2} + 2 \times 3 \\ &= a_{n-3} + 3 + 2 \times 3 \text{ (since } a_{n-2} = a_{n-3} + 3) \\ &= a_{n-3} + 3 \times 3 \\ &= a_{n-4} + 3 + 3 \times 3 \text{ (since } a_{n-3} = a_{n-4} + 3) \\ &= a_{n-4} + 4 \times 3 \end{aligned}$$

$$\begin{aligned} &= a_{n-(n-1)} + (n-1) \times 3 \\ &= a_1 + 3(n-1) \\ &= 2 + 3(n-1) \text{ (since } a_1 = 2 \text{ is the terminating condition)} \end{aligned}$$

$$\therefore a_n = 2 + 3(n-1)$$

Forward Chaining Method :

Given, initial condition : $a_1 = 2$

Now , $a_1 = 2$

$$a_2 = a_1 + 3$$

$$\begin{aligned} a_3 &= a_2 + 3 \\ &= a_1 + 2 \times 3 \end{aligned}$$

$$\begin{aligned} a_4 &= a_3 + 3 \\ &= a_1 + 2 \times 3 + 3 \\ &= a_1 + 3 \times 3 \end{aligned}$$

$$= a_1 + (4 - 1) \times 3$$

$$a_5 = a_1 + (5 - 1) \times 3$$

$$a_n = a_1 + (n - 1) 3$$

$$\therefore a_n = 2 + 3(n - 1)$$

Summation Method :

The given equation can be rearranged as

$$a_n - a_{n-1} = 3$$

$$a_{n-1} - a_{n-2} = 3$$

$$a_{n-2} - a_{n-3} = 3$$

$$a_3 - a_2 = 3$$

$$a_2 - a_1 = 3$$

We stop here, since $a_1 = 2$ is given.

Adding all, we get

$$\begin{aligned} a_n - a_1 &= 3 + 3 + 3 + \dots + (n - 1) \text{ times.} \\ &= 3(n - 1) \end{aligned}$$

$$\Rightarrow a_n = a_1 + 3(n - 1)$$

Problem 7

Solve the recurrence relation $a_n = a_{n-1} + 3$ with $a_1 = 2$ defines the sequence 2, 5, 8,

Solution.

We backtrack the value of a_n by substituting the definition of a_{n-1} , a_{n-2} , and so on until a pattern is clear.

$ \begin{aligned} a_n &= a_{n-1} + 3 \\ &= (a_{n-2} + 3) + 3 \\ &= ((a_{n-3} + 3) + 3) + 3 \end{aligned} $ <p>Eventually this process will produce $a_n =$</p> $ \begin{aligned} &a_n - (n - 1) + (n - 1) \cdot 3 \\ &= a_1 + (n - 1) \cdot 3 \\ &= 2 + (n - 1) \cdot 3 \end{aligned} $	or	$ \begin{aligned} a_n &= a_{n-1} + 3 \\ &= a_{n-2} + 2 \cdot 3 \\ &= a_{n-3} + 3 \cdot 3 \end{aligned} $
--	----	--

An explicit formula for the sequence is $a_n = 2 + (n - 1) \cdot 3$

Problem 8

Write down the first six terms of the sequence defined by $a_1 = 1$, $a_{k+1} = 3a_k + 1$ for $k \geq 1$. Guess a formula for a_n and prove that your formula is correct.

Solution.

The first six terms are

$$a_1 = 1$$

$$a_2 = 3a_1 + 1 = 3(1) + 1 = 4$$

$$a_3 = 3a_2 + 1 = 3(4) + 1 = 13$$

$$a_4 = 40,$$

$$a_5 = 121,$$

$$a_6 = 364.$$

Since there is multiplication by 3 at each step, we might suspect that 3^n is involved in the answer.

After trial and error, we guess that $a_n = \frac{1}{2}(3^n - 1)$ and verify this by mathematical induction.

When $n = 1$, the formula gives,

$$\frac{1}{2}(3^1 - 1) = 1, \text{ which is indeed } a_1, \text{ the first term in the sequence.}$$

Now assume that $k \geq 1$ and that

$$a_k = \frac{1}{2}(3^k - 1).$$

We wish to prove that,

$$a_{k+1} = \frac{1}{2}(3^{k+1} - 1)$$

We have,

$$\begin{aligned} a_{k+1} &= 3a_k + 1 \\ &= 3 \cdot \frac{1}{2}(3^k - 1) + 1 \end{aligned}$$

Using the induction hypothesis,

$$\text{Hence, } a_{k+1} = \frac{1}{2}3^{k+1} - \frac{3}{2} + 1$$

$$= \frac{1}{2}(3^{k+1} - 1) \text{ as required.}$$

By the principle of mathematical induction, our guess is correct.

Problem 9

Backtrack to find an explicit formula for the sequence defined by the recurrence relation $b_n = 2b_{n-1} + 1$ with initial condition $b_1 = 7$

Solution.

We begin by substituting the definition of the previous term in the defining formula.

$$\begin{aligned}
 b_n &= 2b_{n-1} + 1 \\
 &= 2(2b_{n-2} + 1) + 1 \\
 &= 2[2(2b_{n-3} + 1) + 1] + 1 \\
 &= 2^3 b_{n-3} + 4 + 2 + 1 \\
 &= 2^3 b_{n-3} + 2^2 + 2^1 + 1.
 \end{aligned}$$

A pattern is emerging with these rewriting of b_n

(**Note :** There are no set rules for how to rewrite these expressions and a certain amount of experimentation may be necessary.)

The backtracking will end at

$$\begin{aligned}
 b_n &= 2^{n-1} b_{n-(n-1)} + 2^{n-2} + 2^{n-3} + \dots + 2^2 + 2^1 + 1 \\
 &= 2^{n-1} b_1 + 2^{n-1} - 1 \\
 &= 7 \cdot 2^{n-1} + 2^{n-1} - 1 \text{ (using } b_1 = 7) \\
 b_n &= 8 \cdot 2^{n-1} - 1 \text{ (or) } 2^{n+2} - 1
 \end{aligned}$$

Problem 10

Solve the recurrence relation $a_n = a_{n-1} + 2$, $n \geq 2$ subject to initial condition $a_1 = 3$.

Solution.

We backtrack the value of a_n by substituting the expression of a_{n-1} , a_{n-2} and so on, until a pattern is clear.

Given	$a_n = a_{n-1} + 2$...(1)
Replacing n by $n - 1$ in (1), we obtain		

$a_{n-1} = a_{n-2} + 2$ $a_n = a_{n-2} + 2 = (a_{n-2} + 2) + 2$ $= a_{n-2} + 2.2$	From (1),	...(2)
---	-----------	--------

Replacing n by n-2 in (1), we obtain

$$a_{n-2} = a_{n-3} + 2$$

So, from (2),

$$a_n = (a_{n-3} + 2) + 2.2$$

$$= a_{n-3} + 3.2$$

In general

$$a_n = a_{n-k} + k \cdot 2$$

For $k = n - 1$, $a_n = a_{n-(n-1)} + (n - 1) \cdot 2$

$$= a_1 + (n - 1) \cdot 2$$

$$= 3 + (n - 1) \cdot 2$$

which is an explicit formula

1.3 THE SECOND-ORDER LINEAR Homogeneous Recurrence Relation with constant coefficients

Let $k \in \mathbb{Z}^+$ and $C_n (\neq 0)$, C_{n-1} , C_{n-2} , $C_{n-k} (\neq 0)$ be real numbers. If a_n , for $n \geq 0$, is a discrete function, then

$$C_n a_n + C_{n-1} a_{n-1} + C_{n-2} a_{n-2} + \dots + C_{n-k} a_{n-k} = f(x), n \geq k$$

is a linear recurrence relation (with constant coefficients) of order k. When $f(n) = 0$, for all $n \geq 0$, the relation is called **homogeneous**; other wise, it is **non-homogeneous**.

The homogeneous relation of order two :

$$C_n a_n + C_{n-1} a_{n-1} + C_{n-2} a_{n-2} = 0, n \geq 2.$$

A solution of the form $a_n = Cr^n$, where $C \neq 0$ and $r \neq 0$ substituting $a_n = Cr^n$ into

$$C_n a_n + C_{n-1} a_{n-1} + C_{n-2} a_{n-2} = 0$$

We obtain $C_n Cr^n + C_{n-1} Cr^{n-1} + C_{n-2} Cr^{n-2} = 0$,

with $C, r \neq 0$, this becomes

$$C_n r^2 + C_{n-1} r + C_{n-2} = 0,$$

a quadratic equation which is called the **characteristic equation**.

1.4 THE NON HOMOGENEOUS RECURRENCE RELATIONS

The recurrence relations

$$a_n + c_{n-1}a_{n-1} = f(n), n \geq 1 \quad \dots (1)$$

$$a_n + c_{n-1}a_{n-1} + c_{n-2}a_{n-2} = f(n), n \geq 2 \quad \dots (2)$$

Where c_{n-1} and c_{n-2} are constants,

$c_{n-1} \neq 0$ in (1), $c_{n-2} \neq 0$, and $f(n)$ is not identically 0.

Although there is no general method for solving all non homogeneous relations, for certain functions $f(n)$ we shall find a successful technique.

When $c_{n-1} = -1$, (1) gives, for the non homogeneous relation $a_n - a_{n-1} = f(n)$, we have

$$a_1 = a_0 + f(1)$$

$$a_2 = a_1 + f(2) = a_0 + f(1) + f(2)$$

$$a_3 = a_2 + f(3) = a_0 + f(1) + f(2) + f(3)$$

$$\begin{aligned} a_n &= a_0 + f(1) + \dots + f(n) \\ &= a_0 + \sum_{i=1}^n f(i) \end{aligned}$$

We can solve this type of relation in terms of n , if we find a suitable summation formula for $\sum_{i=1}^n f(i)$ (a) **The non homogeneous first-order relation**

$$a_n + c_{n-1}a_{n-1} = kr^n$$

where k is a constant and $n \in \mathbb{Z}^+$

(b) If r^n is not a solution of the associated homogeneous relation $a_n + c_{n-1}a_{n-1} = 0$, then $a_n(P) = Ar^n$, where A is a constant. When r^n is a solution of the associated homogeneous relation, then

$$a_n(P) = Bn r^n, \text{ for } B \text{ a constant.}$$

(c) **The non-homogeneous second order relation**

$$a_n + c_{n-1}a_{n-1} + c_{n-2}a_{n-2} = kr^n.$$

Where k is a constant.

1.4.1. Characteristic Equation Method :

This method can be used to solve any constant order linear recurrence equation with constant coefficient. This recurrence relation may be homogeneous or non-homogeneous. Before attempting to solve any such problem, let us first, understand what is characteristic equation for a given recurrence equation and how to find it.

A recurrence equation of the mentioned type can be arranged in standard form as :

$$A_n + C_1 A_{n-1} + C_2 A_{n-2} + C_3 A_{n-3} = \text{R.H.S} \dots\dots (1)$$

Where C_1, C_2, C_3 are constant coefficients and R.H.S. has one of the following forms :

Form	Examples
Homogeneous	0
A constant to the n^{th} power	$2^n, \pi^{-s}, 2^{-n}, \sqrt{2^n}$
A polynomial in n	$3, n^2, n^2 - n, n^3 + 2n - 1$
A product of a constant to the n^{th} power and a polynomial in n	$2^n (n^2 + 2n - 1), (n - 1) n^6, n 6^n$
A linear combination of any of the above	$(2^n + 3^{n/2}) (n^2 + 2n - 1) + 5$

In the recurrence equation (1), assigning R.H.S. = 0, we get

$$A_n + C_1 A_{n-1} + C_2 A_{n-2} + C_3 A_{n-3} = 0 \dots\dots (2)$$

This equation (2) gives the homogeneous part of the given recurrence equation. Every recurrence equation has a homogeneous part. If the recurrence relation is homogeneous then it has only homogeneous part and solving such equation is one step process. On the other hand, if the given recurrence equation is non-homogeneous then its homogeneous part is obtained by assigning R.H.S. equal to zero.

A characteristic equation corresponds to homogeneous part of the given recurrence relation.

The characteristic equation of (2) is given as :

$$x^3 + c_1 x^2 + c_2 x + c_3 = 0 \quad \dots\dots (3)$$

This has been obtained by the following procedure :

(i) Find the order of the recurrence equation here it is 3.

(ii) Take any variable (say x) and substitute A_n, A_{n-1}, A_{n-2} , by x^3, x^2, x respectively in the homogeneous part of the recurrence equation.

Equation so obtained is called **characteristic equation** of the given recurrence equation .

Example (1) :

The characteristic equation of the recurrence equation $c_n = 3c_{n-1} - 2c_{n-2}$ is given by

$$x^2 - 3x + 2 = 0.$$

Example (2) :

The characteristic equation of the recurrence equation $f_n = f_{n-1} + f_{n-2}$ is given by

$$x^2 - x - 1 = 0.$$

Example (3) :

The characteristic equation of the recurrence equation $A_n - 5A_{n-1} + 6A_{n-2} = 2^n + n$ is given by

$$x^2 - 5x + 6 = 0$$

Theorem 1.1

If the characteristic equation $x^2 - r_1x - r_2 = 0$ of the recurrence equation $a_n = r_1a_{n-1} + r_2a_{n-2}$ has two distinct roots s_1 and s_2 then $a_n = u s_1^n + v s_2^n$ is the closed form formula for the sequence where u and v depend on the initial condition

Proof.

Since s_1 and s_2 are roots of

$$x^2 - r_1x - r_2 = 0 \rightarrow (1)$$

We have

$$s_1^2 - r_1s_1 - r_2 = 0 \rightarrow (2)$$

$$s_2^2 - r_1s_2 - r_2 = 0 \rightarrow (3)$$

Since u and v are dependent on the initial conditions

We have

$$a_1 = us_1 + vs_2$$

And $a_2 = us_1^2 + vs_2^2$

Now,

$$a_n = us_1^n + vs_2^n$$

$$= us_1^{n-2} s_1^2 + vs_2^{n-2} s_2^2$$

$$= us_1^{n-2} [r_1s_1 + r_2] + vs_2^{n-2} [r_1s_2 + r_2]$$

From (2) and (3)

$$= r_1us_1^{n-1} + r_2us_1^{n-2} + r_1vs_2^{n-1} + r_2vs_2^{n-2}$$

$$= r_1[us_1^{n-1} + vs_2^{n-1}] + r_2[us_1^{n-2} + vs_2^{n-2}]$$

$$= r_1a_{n-1} + r_2a_{n-2}$$

(i.e) $a_n = us_1^n + vs_2^n$ is an explicit formula for the given relation .

There are four steps in the process :

Step 1 : Find the homogeneous solution to the homogeneous equation. This results when you set the R.H.S. to zero. If it is already zero, skip the next two steps and go directly to the step 4. Your answer will contains one or more undetermined coefficients whose values cannot be determined until step 4.

Step 2 : Find the particular solution by guessing a form similar to the R.H.S. This step does not produce any additional undetermined coefficients, nor does it eliminate those from step 1.

Step 3 : Combine the homogeneous and particular solution.

Step 4 : Use boundary or initial conditions to eliminate the undetermined constants from the step 1.

Problem 11

What is the solutions of recurrence relation $a_n = a_{n-1} + 2a_{n-2}$ with $a_0 = 2$ and $a_1 = 7$?

Solution :

The characteristic equation of the recurrence relation is $r^2 - r - 2 = 0$.

Its roots are $r = 2$ and $r = -1$.

Hence, the sequence $\{a_n\}$ is a solution to the recurrence relation if and only if $a_n = \alpha_1 2^n + \alpha_2 (-1)^n$ for some constants α_1 and α_2 .

From the initial conditions, it follows that

$$a_0 = 2 = \alpha_1 + \alpha_2$$

$$a_1 = 7 = \alpha_1 \cdot 2 + \alpha_2 \cdot (-1)$$

Solving these two equations shows that $\alpha_1 = 3$ and $\alpha_2 = -1$.

Hence, the solution to the recurrence relation and initial conditions is the sequence $\{a_n\}$ with

$$a_n = 3 \cdot 2^n - (-1)^n.$$

Problem 12`

What is the solution of the recurrence relation $a_n = 6a_{n-1} - 9a_{n-2}$ with initial conditions $a_0 = 1$ and $a_1 = 6$?

Solution :

The only root of $r^2 - 6r + 9 = 0$ is $r = 3$.

Hence, the solution to this recurrence relation is $a_n = \alpha_1 3^n + \alpha_2 n 3^n$ for some constants α_1 and α_2 .

Using the initial conditions, it follows that

$$a_0 = 1 = \alpha_1$$

$$a_1 = 6 = \alpha_1 + 3\alpha_2 + 3$$

Solving these two equations shows that $\alpha_1 = 1$ and $\alpha_2 = 1$.

Consequently, the solution to this recurrence relation and the initial conditions is

$$a_n = 3^n + n3^n$$

Problem 13

Find the solution to the recurrence relation

$$a_n = 6a_{n-1} - 11a_{n-2} + 6a_{n-3} \text{ with initial conditions } a_0 = 2, a_1 = 5 \text{ and } a_2 = 15$$

Solution :

The characteristic polynomial of this recurrence relation is $r^3 - 6r^2 + 11r - 6 = 0$
The characteristic roots are $r = 1, r = 2$ and $r = 3$

Since $r^3 - 6r^2 + 11r - 6 = (r - 1)(r - 2)(r - 3)$

Hence, the solutions to this recurrence relation are of the form

$$a_n = \alpha_1 \cdot 1^n + \alpha_2 \cdot 2^n + \alpha_3 \cdot 3^n.$$

To find the constants α_1, α_2 and α_3 .

use the initial conditions.

$$\text{This gives } a_0 = 2 = \alpha_1 + \alpha_2 + \alpha_3$$

$$a_1 = 5 = \alpha_1 + \alpha_2 \cdot 2 + \alpha_3 \cdot 3$$

$$a_2 = 15 = \alpha_1 + \alpha_2 \cdot 4 + \alpha_3 \cdot 9$$

When these three simultaneous equations are solved for α_1, α_2 and α_3 , we find that $\alpha_1 = 1, \alpha_2 = 1$ and $\alpha_3 = 2$.

Hence, the unique solution to this recurrence relation and the given initial conditions is the sequence $\{a_n\}$ with

$$a_n = 1 + 2^n + 2 \cdot 3^n$$

Problem 14

Find the solution to the recurrence relation

$$a_n = -3a_{n-1} - 3a_{n-2} - a_{n-3} \text{ with initial conditions } a_0 = 1, a_1 = -2 \text{ and } a_2 = 1.$$

Solution :

The characteristic equation of this recurrence relation is $r^3 + 3r^2 + 3r + 1 = 0$. Since $r^3 + 3r^2 + 3r + 1 = (r + 1)^3$, there is a single root $r = -1$ of multiplicity three of the characteristic equation.

The solutions of this recurrence relation are of the form

$$a_n = \alpha_{1,0} (-1)^n + \alpha_{1,1} 1^n (-1)^n + \alpha_{1,2} n^2 (-1)^n$$

To find the constant $\alpha_{1,0}$, $\alpha_{1,1}$, $\alpha_{1,2}$ use the initial conditions.

This gives $a_0 = 1 = \alpha_{1,0}$

$$a_1 = -2 = -\alpha_{1,0} - \alpha_{1,1} - \alpha_{1,2}$$

$$a_2 = -1 = \alpha_{1,0} + 2\alpha_{1,1} + 4\alpha_{1,2}$$

The simultaneous solution of these three equations is

$$\alpha_{1,0} = 1, \quad \alpha_{1,1} = 3, \quad \text{and} \quad \alpha_{1,2} = -2$$

Hence, the unique solution to this recurrence relation and the given initial conditions is the sequence $\{a_n\}$ with

$$a_n = (1 + 3n - 2n^2) (-1)^n.$$

1.6 The Method of Generating Function

One of the uses of generating function method is to find the closed form formula for a recurrence relation. Before using this method, ensure that the given recurrence equation is in linear form.

A non-linear recurrence equation cannot be solved by the Generating Function Method. Use substitution of variable technique to convert a non linear recurrence (equation) relation into linear.

Solving a recurrence (equation) relation using generating function method involves two steps process.

Step 1: Find generating function for the sequences for which the general term is given by recurrence relation.

Step 2: Find coefficient of x^2 or $\frac{x^2}{n!}$ depending upon whether the generating function

The value so obtained will be an algebraic formula for a_n , expressed in terms of n which is the position of a_n in sequence.

A generating function is a polynomial expression of the form

$$f(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \dots + a_nx^n + \dots$$

in which the coefficients a_i are all zero after a certain point, a generating function usually has infinitely many non-zero terms. There is an obvious correspondence between generating functions and sequences.

$$a_0, a_1, a_2, \dots$$

$$a_0 + a_1x + a_2x^2 + a_3x^3 + \dots \leftrightarrow a_0, a_1, a_2, a_3, \dots$$

*If $f(x) = a_0 + a_1x + a_2x^2 + \dots$ and

$g(x) = b_0 + b_1x + b_2x^2 + \dots$ then

$$f(x) + g(x) = (a_0 + b_0) + (a_1 + b_1)x + (a_2 + b_2)x^2 + \dots$$

$$f(x)g(x) = (a_0b_0) + (a_1b_0 + a_0b_1)x + (a_2b_0 + a_1b_1 + a_0b_2)x^2 + \dots$$

The coefficient of x^n in the product $f(x)g(x)$ is the **infinite sum**

$$a_0b_n + a_1b_{n-1} + a_2b_{n-2} + \dots + a_nb_0.$$

Problem 15

If $f(x) = 1 + x + x^2 + \dots + x^n + \dots$ and

$g(x) = 1 - x + x^2 - x^3 + \dots + (-1)^n x^n + \dots$,

find $f(x) + g(x)$ and $f(x)g(x)$.

Solution.

$$\begin{aligned} f(x) + g(x) &= (1 + x + x^2 + \dots + x^n + \dots) + (1 - x + x^2 - x^3 + \dots + (-1)^n x^n + \dots) \\ &= (1 + 1) + (1 - 1)x + (1 + 1)x^2 + \dots + (1 + (-1)^n)x^n + \dots \\ &= 2 + 2x^2 + 2x^4 + \dots \end{aligned}$$

$$\begin{aligned} f(x)g(x) &= (1 + x + x^2 + \dots + x^n + \dots) \cdot (1 - x + x^2 - x^3 + \dots + (-1)^n x^n + \dots) \\ &= 1 + [1(-1) + 1(1)]x + [1(1) + 1(-1) + 1(1)]x^2 + \dots \\ &= 1 + x^2 + x^4 + x^6 + \dots \end{aligned}$$

Problem 16

Solve the recurrence relation $a_n = 3a_{n-1}$, $n \geq 1$, given $a_0 = 1$.

Solution.

Consider the generating function $f(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n + \dots$ of the sequence a_0, a_1, a_2, \dots multiplying by $3x$ and writing the product $3xf(x)$ below $f(x)$ so that terms involving x^n match, we obtain

$$\begin{aligned} f(x) &= a_0 + a_1x + a_2x^2 + \dots + a_nx^n + \dots \\ 3xf(x) &= 3a_0x + 3a_1x^2 + \dots + 3a_{n-1}x^n + \dots \end{aligned}$$

Subtracting gives

$$f(x) - 3xf(x) = a_0 + (a_1 - 3a_0)x + (a_2 - 3a_1)x^2 + \dots + (a_n - 3a_{n-1})x^n + \dots$$

Since $a_0 = 1$, $a_1 = 3a_0$.

In general, $a_n = 3a_{n-1}$, this says that

$$(1 - 3x)f(x) = 1.$$

Thus,
$$f(x) = \frac{1}{1-3x}$$

We have
$$\frac{1}{1-x} = 1 + x + x^2 + \dots \quad (*)$$

Using (*),
$$\begin{aligned} f(x) &= 1 + 3x + (3x)^2 + \dots + (3x)^n + \dots \\ &= 1 + 3x + 9x^2 + \dots + 3^n x^n + \dots \end{aligned}$$

We conclude that a_n , which is the coefficient of x^n in $f(x)$, must equal 3^n .

We have $a_n = 3^n$ as the solution to our recurrence relation.

Problem 17

Solve the recurrence relation $a_n = 2a_{n-1} - a_{n-2}$, $n \geq 2$, given $a_0 = 3$, $a_1 = -2$.

Solution.

Letting $f(x)$ be the generating function of the sequence in question.

We have
$$f(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n + \dots$$

$$2xf(x) = 2a_0x + 2a_1x^2 + \dots + 2a_{n-1}x^n + \dots$$

$$x^2 f(x) = a_0x^2 + \dots + a_{n-2}x^n + \dots$$

Therefore,
$$f(x) - 2xf(x) + x^2f(x)$$

$$\begin{aligned} &= a_0 + (a_1 - 2a_0)x + (a_2 - 2a_1 + a_0)x^2 + \dots + (a_n - 2a_{n-1} + a_{n-2})x^n + \dots \end{aligned}$$

$$= 3 - 8x.$$

Since $a_0 = 3$, $a_1 = -2$ and $a_n - 2a_{n-1} + a_{n-2} = 0$ for $n \geq 2$.

So, $(1 - 2x + x^2) f(x) = 3 - 8x$

$$(1 - x)^2 f(x) = 3 - 8x$$

$$f(x) = \frac{3-8x}{(1-x)^2}$$

$$= (1 + 2x + 3x^2 + \dots + (n+1)x^n + \dots) (3 - 8x)$$

$$= 3 - 2x - 7x^2 - 12x^3 + \dots + [3(n+1) - 8n]x^n + \dots$$

$$= 3 - 2x - 7x^2 - 12x^3 + \dots + (-5n+3)x^n + \dots$$

Therefore $a_n = 3 - 5n$ is the desired solution.

Problem 18

Find the sequence $\{y_x\}$ having the generating function G given by

$$G(x) = \frac{3}{1-x} + \frac{1}{1-2x}$$

Solution

We have

$$G(x) = 3(1-x)^{-1} + (1-2x)^{-1}$$

$$= 3(1 + x + x^2 + \dots + x^n + \dots) + (1 + 2x^1 + 2^2x^2 + \dots + 2^n x^n + \dots)$$

$$= (3+1) + (3+2)x + (3+2^2) + \dots + (3+2^n)x^n + \dots$$

where

$$y_n = 3 + 2^n$$

Problem 19

Suppose a is a real number. Show that $\frac{1}{1-ax}$ is the generating function for a certain geometric sequence.

Solution

We have

$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + \dots \quad (1)$$

Replacing x by ax in (1), we see that

$$\frac{1}{1-ax} = 1 + ax + (ax)^2 + (ax)^3 + \dots$$

$$= 1 + ax + a^2x^2 + a^3x^3 + \dots$$

From this, we see that $\frac{1}{1-ax}$ is the generating function for the sequence $1, a^1, a^2, a^3, \dots$ which is the geometric sequence with first term 1 and common ratio a.

Problem 20

Prove that $\frac{1}{(1-x)^2} = 1 + 2x + 3x^2 + 4x^3 + \dots + (n+1)x^n + \dots$.

Solution

$$\begin{aligned} 1 &= (1-x)^2(1 + 2x + 3x^2 + \dots + (n+1)x^n + \dots) \\ &= (1-2x+x^2)(1 + 2x + 3x^2 + \dots + (n+1)x^n + \dots) \\ &= 1 + [1(2) - 2(1)]x + [1(3) - 2(2) + 1(1)]x^2 + \dots + [1(n+1) - 2(n) + 1(n-1)]x^n + \dots \\ &= 1, \text{ since } n+1 - 2n + n = 0. \end{aligned}$$

Expression for Generating Functions

If $A(x) = \sum_{n=0}^{\infty} a_n x^n$ then

$$\sum_{n=k}^{\infty} a_n x^n = A(x) - a_0 - a_1 x^1 - \dots - a_{k-1} x^{k-1}$$

$$\sum_{n=k}^{\infty} a_{n-1} x^n = x^1 (A(x) - a_0 - a_1 x^1 - \dots - a_{k-2} x^{k-2})$$

$$\sum_{n=k}^{\infty} a_{n-2} x^n = x^2 (A(x) - a_0 - a_1 x^1 - \dots - a_{k-3} x^{k-3})$$

$$\sum_{n=k}^{\infty} a_{n-k} x^n = x^k (A(x))$$

Table of Generating Functions

Sequence a_n	Generating Function $A(x)$
$C(k,n)$	$(1+x)^k$
1	$\frac{1}{1-x}$
a^n	$\frac{1}{1-ax}$

-1^n	$\frac{1}{1+x}$
$-a^n$	$\frac{1}{1+ax}$
$C(k-1+n, n)$	$\frac{1}{(1-x)^k}$
$C(k-1+n, n)a^n$	$\frac{1}{(1-ax)^k}$
$C(k-1+n, n)(-a)^n$	$\frac{1}{(1+ax)^k}$
$n+1$	$\frac{1}{(1-x)^2}$
n	$\frac{1}{(1-x)^2}$
$(n+2)(n+1)$	$\frac{2}{(1-x)^3}$
$(n+1)(n)$	$\frac{2x}{(1-x)^3}$
n^2	$\frac{x(1+x)}{(1-x)^3}$
$(n+3)(n+2)(n+1)$	$\frac{6}{(1-x)^4}$
$(n+2)(n+1)(n)$	$\frac{6x}{(1-x)^4}$
n^3	$\frac{x(1+4x+x^2)}{(1-x)^2}$
$(n+1)a^n$	$\frac{1}{(1-ax)^2}$
na^n	$\frac{ax}{(1-ax)^2}$
n^2a^n	$\frac{(ax)(1+ax)}{(1-ax)^3}$
n^3a^n	$\frac{(ax)(1+4ax+a^2x^2)}{(1-ax)^4}$

Theorem 1.2

If $\{a_n\}_{n=0}^{\infty}$ is a sequence of numbers which satisfy the linear recurrence relation with constant coefficients $a_n + c_1 a_{n-1} + \dots + c_k a_{n-k}$ where $c_k \neq 0$, and $n \geq k$, then the generating function

$$A(x) = \sum_{n=0}^{\infty} a_n x^n \text{ equals } \frac{P(x)}{Q(x)},$$

where

$$P(x) = a_0 + (a_1 + c_1 a_0)x^1 + \dots + (a_{k-1} + c_1 a_{k-2} + \dots + c_{k-1} a_0)x^{k-1}$$

$$Q(x) = 1 + c_1 x^1 + \dots + c_k x^k.$$

Conversely, given such polynomials $P(x)$ and $Q(x)$, where $P(x)$ has degree less than k , and $Q(x)$ has degree k , there is a sequence $\{a_n\}_{n=0}^{\infty}$ whose generating function is $A(x) = \frac{P(x)}{Q(x)}$

Problem 21

Solve $a^n - 8a_{n-1} + 21a_{n-2} - 18a_{n-3} = 0$ for $n \geq 3$.

Solution.

Here, if $A(x) = \sum_{n=0}^{\infty} a_n x^n$, then

$$\sum_{n=3}^{\infty} a_n x^n - 8 \sum_{n=3}^{\infty} a_{n-1} x^n + 21 \sum_{n=3}^{\infty} a_{n-2} x^n - 18 \sum_{n=3}^{\infty} a_{n-3} x^n = 0,$$

$$(A(x) - a_0 - a_1 x^1 - a_2 x^2) - 8x^1(A(x) - a_0 - a_1 x^1) + 21x^2(A(x) - a_0) - 18x^3 A(x) = 0$$

$$A(x) = \frac{a_0 + (a_1 - 8a_0)x^1 + (a_2 - 8a_1 + 21a_0)x^2}{1 - 8x^1 + 21x^2 - 18x^3}$$

Since $1 - 8x^1 + 21x^2 - 18x^3 = (1 - 2x^1)(1 - 3x^1)^2$

We see that there are constants C_1, C_2, C_3 , such that

$$A(x) = \frac{C_1}{(1-2x)} + \frac{C_2}{(1-3x)} + \frac{C_3}{(1-3x)^2}$$

$$A(x) = \sum_{n=0}^{\infty} [C_1 2^n + C_2 3^n + C_3 n^3(n+1, n)] x^n$$

$$a_n = C_1 2^n + C_2 3^n + C_3 (n+1) 3^n.$$

Conclusion:

First Order homogenous linear recurrence is in fact a geometric sequence
We studied characteristics of second and higher order linear homogenous recurrence relations with constant coefficients. The non-homogenous recurrence relations with constant coefficients. Different methods to solve such recurrences. As most of the algorithms are recursive therefore to give analysis of such algorithms, we have discussed his powerful technique.

Reference

- [1] c.vasudev - Theory and problems on combinatorics – recurrence relation
- [2] M.k sen chakaraborthy - Introduction to discrete mathematics - application of recurrence relation.