Manonmaniam Sundaranar University,

Directorate of Distance & Continuing Education,

Tirunelveli- 627 012, Tamil Nadu, India

OPEN AND DISTANCE LEARNING (ODL) PROGRAMMES

(FOR THOSE WHO JOINED THE PROGRAMMES FROM THE ACADEMIC YEAR 2023–2024)

M. Sc. Physics

Course Material

Practical IV

SPHP41

Prepared By

Dr. S. Shailajha

Dr. B. Bagyalakshmi

Assistant Professor

Department of Physics

Manonmaniam Sundaranar University

Tirunelveli – 12

PRACTICAL – IV

Advanced Physics Experiments – II and Numerical Methods in C++

Section – A: Advanced Physics Experiments – II

(Any 6 experiments)

1. Determination of resistivity and bandgap of semiconductors by Four probe Method

Aim

To determine the resistivity and bandgap of semiconductors by Four probe Method.

Apparatus Required

Probe arrangement, Sample, Oven, Four probe setup

Formula

$$\rho_0 = \frac{V}{I} \times 2\pi S$$

ρ_0 = resistivity (Ωm)

V = floating potential difference between the inner probe (mV)

I = Current through the outer pair of probe (mA)

S = Distance between the two adjective probe

$$\rho = \frac{\rho_0}{f(\frac{w}{s})}$$

f = Correction factor

w= Thickness of crystal (m)

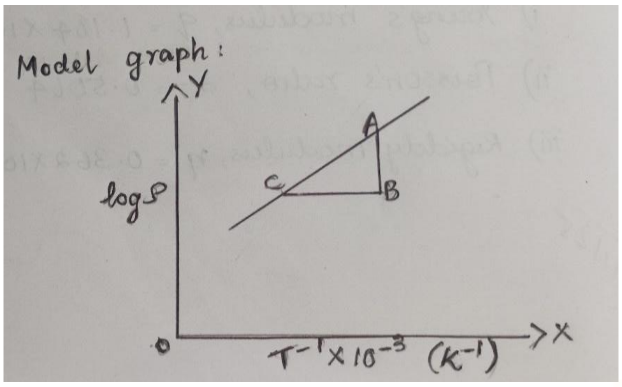
$$\frac{E_g = 2 \times K \times log_e \rho}{T^{-1}}$$

E_g=Energy Band Gap (eV)

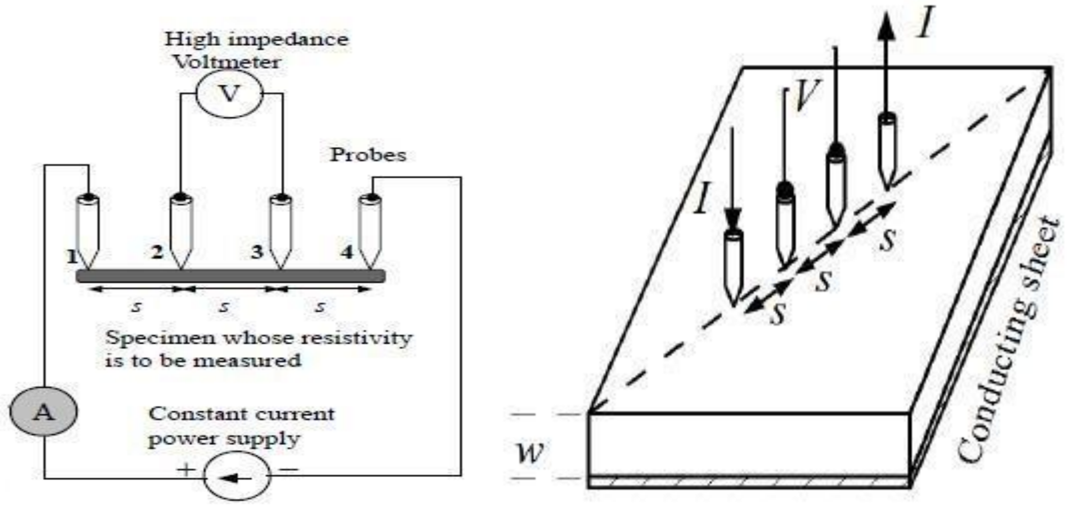
K=Boltzmann's Constant

T=Temperature (K)

Model Graph



Figure


High impedance Voltmeter
V
Probes
1
2
3
4
s
s
s
Specimen whose resistivity is to be measured
A
Constant current power supply
+
−
I
V
I
s
s
s
w
Conducting sheet

The figure shows the arrangements of four probes that measure voltage (V) and supply current (A) to the surface of the crystal.

Procedure

1. The resistivity of material is uniform in the area of measurement.
2. If there is a minority carrier injection into the semiconductor by the current- carrying electrodes most of the carriers recombine near electrodes so that their effect on conductivity is negligible.
3. The surface on which the probes rest is flat with no surface leakage.
4. The four probes used for resistivity measurement contact surface at points that lie in a straight line.
5. The diameter of the contact between metallic probes and the semiconductor should be small compared to the distance between the probes.
6. The boundary between the current carrying electrodes and the bulk material is hemispherical and small in diameter.
7. The surface of semiconductor material may be either conducting and non-conducting. A conducting boundary is one on which material of much lower resistivity than semiconductor has been plated. A nonconducting boundary is produced when the surface of the

semiconductor is in contact with insulator. Fig: 2 show the resistivity probes on a die of material. If the side boundaries are adequately far from the probes, the die may be considered to be identical to a slice. For this case of a slice of thickness w and the resistivity is computed as

$$\rho = \frac{\rho_0}{f(\frac{w}{s})} \tag{2}$$

The function, f(w/S) is a divisor for computing resistivity which depends on the value of w and S We assume that the size of the metal tip is infinitesimal and sample thickness is greater than the distance between the probes,

$$\rho_0 = \frac{V}{I} \times 2\pi S \tag{3}$$

Where V the potential difference between inner probes in volts.

I Current through the outer pair of probes in ampere.

S Spacing between the probes in meter.

Observations:

Temperature(K)	Voltage(V)	ρ(Ωm)	T-1 × 10-3 (K-1)	log ρ

Result:

The resistivity of semiconductors is determined as

2. Study the Characteristics of load cell

Aim:

To study the performance characteristics of load cell.

Apparatus Required

Trainer Kit, Power Supply

Procedure:

1. Open the top cover of the trainer kit wooden box.
2. Connect the cantilever beam type load cell leads with the trainer kit terminals.

   Red lead with red terminal.

   Black lead with black terminal.

   Green lead with green terminal.

   Yellow lead with yellow terminal.
3. Connect the 3, pin mains plug of the training kit to the mains socket (230 Volt, 10%, 50 Hz supply)
4. Keep Digital Voltmeter switch at Kg. Position.
5. Connect patch cord between output terminal and digital voltmeter terminal.
6. Switch ON the trainer kit, the display will light up and will show some reading.
7. Adjust zero pot to set 0.00 reading on display without apply any load on the pan.
8. Put 1 Kgs. Weights on the pan of the cantilever beam and adjust span pot to show 1.00 reading on display.
9. Repeat steps 6 to 8.
10. Now apply loads in steps of 100 gms ad note down the reading in the following table in increasing and decreasing mode.
11. Now, plot the graph between applied load and Digital Voltmeter reading in Kgs. With a resolution of 0.01 Kg and applied load and measure non-=linearity, Hysteresis error etc.

TABLE – I

Sl.No.	Load increasing Mode		Load in Decreasing Mode.	
	Load (in Kg.)	DVM Reading (in mV)	Applied Load (in Kg.)	DVM (in mV)

Instrumentation amplifier output measurement:

12. Keep Digital Voltmeter at mV position.
13. Connect patch cord between instrumentation output terminal and Digital Voltmeter terminal.
14. At no load condition display will show reading. Note down this reading.
15. Apply load in steps of 100 gms. and note down readings in the given table in increasing and decreasing mode.
16. Now, plot the graph between applied load and Digital Voltmeter reading in Kgs. With a resolution of 0.01 Kg. and applied load and measure non-linearity, Hysteresis error etc.

TABLE – II

Sl.No.	Load increasing Mode		Load in Decreasing Mode.	
	Load (in Kg.)	DVM Reading (in mV)	Applied Load (in Kg.)	DVM (in mV)

Observations:

The result give below has been taken on prototype. The actual results may vary from unit to unit.

Load increasing Mode		
Load (in Kg.)	**Applied Load (in Kg.)**	**DVM Reading (in mV)**

Load in Decreasing Mode.		
Load (in Kg.)	**Applied Load (in Kg.)**	**DVM Reading (in mV)**

3. **Determination of the Distance Between Tracks on a CD and DVD Using a Solid-State Laser**

Aim:

To measure the track pitch (distance between adjacent tracks) of a CD and a DVD using a solid-state laser and the diffraction pattern generated by the laser light interacting with the tracks on the disc surface.

Apparatus Required:

- Solid-state laser (wavelength: 780 nm for CD, 650 nm for DVD)
- CD (Compact Disc) and DVD (Digital Versatile Disc)
- Optical setup: lenses, mirrors
- Photodetector or CCD camera
- Rotary disc holder and motor (for rotating the disc)
- Precision angle measurement device or software for angle detection
- Ruler or micrometer for calibration
- Computer with analysis software (for diffraction pattern analysis)

Formula:

When a laser beam is directed at the surface of a CD or DVD, the light interacts with the spiral tracks on the disc. The reflection from the tracks creates a diffraction pattern. The angle of diffraction (θ) is related to the wavelength (λ) of the laser and the distance between adjacent tracks (track pitch, d) by Bragg's Law:

$$2d \sin\theta = n\lambda$$

$$d = \frac{n\lambda}{\sin\theta}\,(m)$$

Where:

- λ = Wavelength of the laser light, in meters (m)
- d = Track pitch (distance between adjacent tracks)
- θ = Angle of diffraction, in radians (rad).

Procedure:

1. Setup and Calibration:

- Place the CD or DVD securely on the rotating disc holder.
- Ensure the solid-state laser is set to emit a stable beam at the correct wavelength (780 nm for CD, 650 nm for DVD).
- Position the laser at a fixed distance from the disc surface, ensuring the beam is normal (perpendicular) to the surface of the disc.

- Attach the photodetector or CCD camera to detect the reflected laser light and capture the diffraction pattern.
- Rotate the disc slowly using the motor to observe the diffraction pattern continuously.
- Calibrate the angle measurement system or software to ensure accurate detection of the diffraction angle.

2. Data Collection: • Turn on the laser and focus the beam onto the surface of the rotating

disc.

- Record the diffraction pattern observed on the detector (a circular or linear pattern depending on the setup).
- Measure the angle θ of the diffraction pattern with respect to the central laser beam.
  - o This angle can be determined by analysing the position of the first diffraction maxima (the angle at which the first intense spot appears).
- Repeat the measurement for several different positions along the track to ensure accuracy.

3. Calculation of Track Pitch:

- Using the measured diffraction angle (θ) and the known wavelength of the laser (λ), apply Bragg's Law to calculate the track pitch dd for the CD and DVD • Perform this calculation separately for the CD and the DVD.
- Compare the track pitch of the CD and DVD.

4. Analysis:

- Verify the calculated track pitch values against known values for CD (1.6 µm) and DVD (0.74 µm) to check the accuracy of the experiment.
- Discuss the differences in the track pitch between CD and DVD and how this relates to their data storage capacity and technology (DVDs use a smaller track pitch for higher data density).

Observation:

Disc Type	**Laser Wavelength (λ)**	**Measured Diffraction Angle (θ)**	**Calculated Track Pitch d (m)**
CD	780 nm		
	780 nm		

Disc Type	**Laser Wavelength (λ)**	**Measured Diffraction Angle (θ)**	**Calculated Track Pitch d (m)**
	780 nm		
DVD	650 nm		
	650 nm		
	650 nm		

Discussion:

- The calculated track pitch for the CD should be close to the theoretical value of 1.6 µm, and for the DVD, it should be close to 0.74 µm.
- The smaller track pitch of DVDs allows for higher data storage density, which is one of the key differences between CD and DVD technologies.
- The use of a solid-state laser in this experiment enables precise measurement due to its stable wavelength and focused beam, essential for accurate diffraction angle measurement.

Calculation:

Result

In this experiment, the track pitch of a CD and DVD was successfully determined using a solid-state laser and diffraction patterns. The results align with known values for CDs and DVDs, demonstrating the practical application of optical principles like diffraction and Bragg's Law in determining the physical characteristics of optical discs.

4. UV Spectral Data Analysis of the Given Spectrum

Aim:

To analyze and interpret the UV (Ultraviolet) absorption spectrum of a given sample. The analysis will focus on identifying key absorption peaks, their corresponding wavelengths, and how these features relate to the molecular structure and functional groups within the sample.

Requirements

- UV-Vis Spectrophotometer (Note: The analysis will be based on given spectral data, Sample data (provided spectrum of the sample)
- Computer with software for spectral analysis like origin or anyother (for viewing and analyzing the given spectrum)
- Ruler or software tools to measure peak wavelengths and absorbance

Theory:

UV-Vis spectroscopy is used to study the absorption of ultraviolet (200–400 nm) and visible (400–700 nm) light by a sample. The absorption peaks in the UV region correspond to the electronic transitions in the sample molecules. These transitions typically involve the movement of electrons between molecular orbitals, such as: • **$\pi \rightarrow \pi^*$**: Transition of electrons in an aromatic or double bond.

- **$n \rightarrow \pi^*$**: Transition involving lone pair electrons to a π^* anti-bonding orbital, typically seen in carbonyl compounds, esters, and other heteroatom-containing groups.
- **$\sigma \rightarrow \sigma^*$**: Transitions for molecules with single bonds, although these are usually in the far-UV region below 200 nm.
- **Beer-Lambert Law** relates the absorbance (A) of a sample to its concentration (c), path length (l), and molar absorptivity (ε):

$$A = \varepsilon c l$$

Where:

- A = Absorbance (unitless)
- ε = Molar absorptivity ($L \cdot mol^{-1} \cdot cm^{-1}$)
- c = Concentration of the sample ($mol \cdot L^{-1}$)
- l = Path length of the cuvette (cm)

Procedure:

1. Preparation of the Data:

- Obtain the **UV absorption spectrum** of the given sample. This data may be provided as a graph or a table of wavelength vs. absorbance values.
- If using software to visualize the spectrum, ensure that the UV range (200–400 nm) is displayed, which is typical for UV spectra.

2. Identify Key Features in the Spectrum:

- **Locate Absorption Peaks**: Identify the main **absorption peaks** in the spectrum. These are characterized by sharp rises in absorbance at specific wavelengths. For example:
  - A peak at **230 nm** may correspond to the **π → π*** transition of an aromatic ring. o A peak around **290–300 nm** may indicate the presence of a **carbonyl group (C=O)**, often related to **n → π*** transitions.
- **Measure Peak Wavelengths**: Record the wavelengths (λmax) of the absorption maxima. These values are essential for identifying the electronic transitions occurring in the sample.

3. Interpret the Absorption Peaks:

- **Aromatic Compounds**: For aromatic compounds like benzene or aspirin, expect **π → π*** transitions in the range of **250–270 nm**. Strong peaks in this region suggest the presence of an aromatic ring.
- **Carbonyl Groups (C=O)**: A peak around **170–200 nm** is characteristic of carbonyl-containing groups, such as esters, aldehydes, and ketones, due to **n → π*** transitions. However, aromatic esters like aspirin might show a peak slightly shifted, around **290–300 nm**.
- **Functional Group Identification**: By comparing the peak positions with known values, functional groups can be identified. For example, a peak at **230 nm** may indicate an aromatic ester, while **255 nm** is indicative of a simple aromatic compound like benzene.

4. Analysis and Data Interpretation: • Compare the observed peak positions with known UV absorption ranges for different functional groups. Refer to typical values for aromatic rings, carbonyl groups, and other common UV-active functional groups.

- Based on the peaks identified, hypothesize about the sample's molecular structure. For instance, if the sample shows strong absorbance at **230 nm** and **290 nm**, it likely contains both an aromatic ring and an ester group.
- You may use the Beer-Lambert law to estimate concentration if the path length and molar absorptivity are known.

Example Data and Analysis:

Example 1: UV Spectrum of Aspirin (Acetylsalicylic Acid)

- **Absorption Peaks**:
  - **λ_max = 230 nm** (strong absorption) o **λ_max = 295 nm** (moderate absorption)

Analysis:

1. **Peak at 230 nm**:
   - This peak corresponds to the **π → π*** transitions in the aromatic ring. The absorption is typical for aromatic compounds in the UV region.

2. **Peak at 295 nm**:
   - This peak is associated with the **n → π*** transition of the carbonyl group (C=O) in the ester functional group (acetyl group in aspirin).

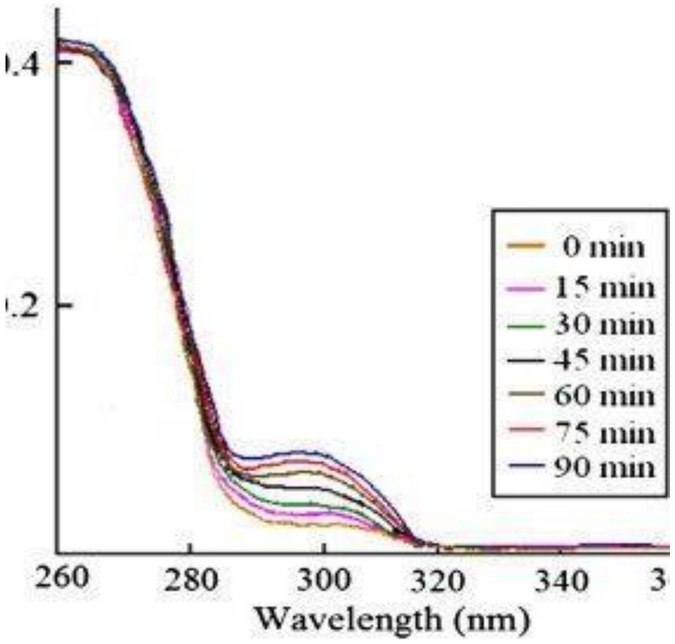


Fig.1: UV Spectrum of Aspirin (Acetylsalicylic Acid)

Example 2: UV Spectrum of Benzene

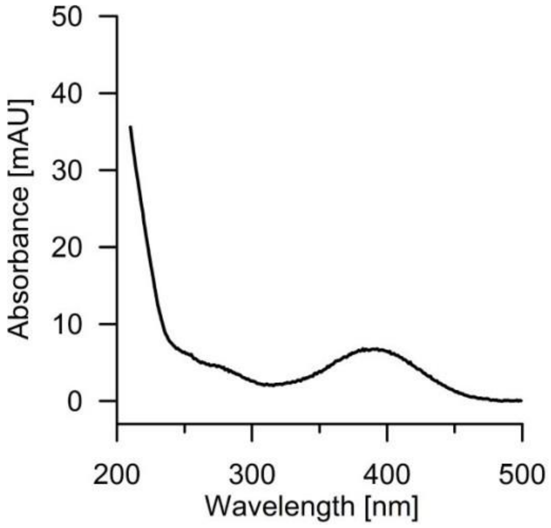


Fig.2: UV Spectrum of Benzene

- **Absorption Peak**:
  - **λ_max = 255 nm** (strong absorption)

Analysis:

1. **Peak at 255 nm**:
   - The absorption at **255 nm** is typical of **π → π*** transitions in simple aromatic compounds like benzene. This is the characteristic absorption for the aromatic ring without additional functional groups.

Results:

Sample	Absorbance Peak(s)	Wavelength(s)	Likely Functional Group or Transition
Aspirin			
Benzene			
Sample			

Discussion:

- **Aspirin**: The UV spectrum of aspirin shows key absorption peaks at **230 nm** and **295 nm**, corresponding to the aromatic $\boldsymbol{\pi} \rightarrow \boldsymbol{\pi}^*$ transition and the ester $\mathbf{n} \rightarrow \boldsymbol{\pi}^*$ transition, respectively. These peaks confirm the presence of both aromatic and ester functional groups in aspirin.
- **Benzene**: The UV spectrum of benzene shows a single strong peak at **255 nm**, indicative of the $\boldsymbol{\pi} \rightarrow \boldsymbol{\pi}^*$ transition of the benzene ring. This simple spectrum reflects the absence of additional functional groups in the benzene molecule.

Result:

In this analysis, the UV spectra of aspirin and benzene were examined.

5. Simulation of Satellite Orbit Around the Earth Using the Universal Law of Gravitation in Scilab

Aim:

To simulate the motion of a satellite orbiting the Earth using the Universal Law of Gravitation in Scilab and study the effect of different parameters on its trajectory.

Materials and Software Required:

- Computer with Scilab installed
- Knowledge of differential equations and numerical integration

Theory:

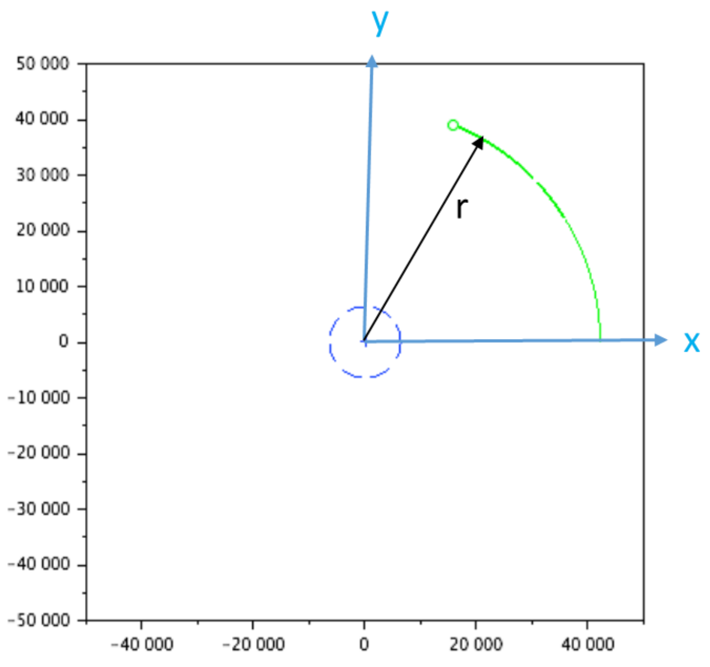
1. Express the physics problem.

The problem is based on the universal law of gravitation:

$$\vec{F} = -G * \frac{m.M}{\|\vec{r}\|_2} * \frac{\vec{r}}{\|\vec{r}\|}$$

$$m * \ddot{x} = -G * \frac{m.M}{\left(\sqrt{x^2+y^2}\right)^3} * x \text{ --------- (1)}$$

$$m * \ddot{y} = -G * \frac{m.M}{\left(\sqrt{x^2+y^2}\right)^3} * y \text{--------- (2)}$$



Position of the satellite is at a distance *r[x;y]* Earth mass centre is at *O[0;0]* Constants of the problem:

Gravitational constant $G = 6.6.67 \times 10^{-11}\ m^{3}kg^{-1}s^{-2}$

Mass of the earth $M = 5.98 \times 10^{24}\ kg$

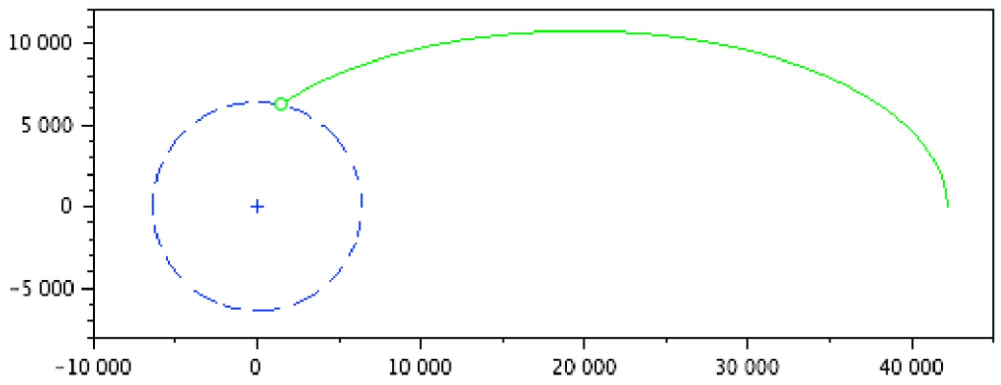
Radius of the Earth $r_{\text{earth}} = 6.38 \times 10^{6}\ m$

2. Translate your problem into Scilab

Scilab is a matrix---based language. Instead of expressing the system as set of 4 independent equations (along the x and y axis, for position and speed), we describe it as a single matrix equation, of dimension 4x4:

➢ Open *scinotes* with edit myEarthRotation.sci ➢ Define the skeleton of the function:

```
function
udot=f(t, u)
G = 6.67D-11; //Gravitational constant
M = 5.98D24; //Mass of the Earth
c = -G * M;
r_earth = 6.378E6; //radius of the Earth
r = sqrt(u(1)^2 + u(2)^2);
// Write the relationhsip between udot and u
if r < r_earth then udot = [0 0 0 0]'; else
        A = [[0 0 1 0];
        [0 0 0 1];
        [c/r^3 0 0 0];
        [0 c/r^3 0 0]];
udot = A*u;
end endfunction
```



Try out the final script with the following initial conditions in speed and altitude: geo_alt = 35784; // in kms geo_speed = 1074; // in m/s simulation_time = 24; // in hours
U = earthrotation(geo_alt, geo_speed, simulation_time);

3. Compute the results and create a visual animation

```
function U=earthrotation(altitude, v_init, hours)
// altitude given in km
// v_init is a vector [vx; vy] given in m/s
// hours is the number of hours for the
simulation r_earth = 6.378E6;
altitude = altitude * 1000;
U0 = [r_earth + altitude; 0; 0; v_init];

t = 0:10:(3600*hours); // simulation time, one point every 10 seconds
U = ode(U0, 0, t, f); // Draw the earth in blue
angle = 0:0.01:2*%pi; x_earth = 6378 *
cos(angle); y_earth = 6378 * sin(angle);
fig = scf(); a = gca();

a.isoview = "on";
plot(x_earth,
y_earth, 'b--');
plot(0, 0, 'b+');
// Draw the trajectory computed
comet(U(1,:)/1000, U(2,:)/1000,
"colors", 3); endfunction
```

The resolution of the ordinary differential equation (ODE) is computed with the Scilab function ode.

Procedure:

1. **Initialize Parameters:** As given above
2. **Apply Newton's Laws:** Use Newton's Second Law to derive equations of motion:
3. **Numerical Integration:** Implement Euler or Runge-Kutta method in Scilab to compute the satellite's new position and velocity over time.
4. **Plot the Trajectory:** Use Scilab's plotting functions to visualize the satellite's orbit around the Earth.
5. **Modify Parameters:** Change initial velocity, altitude, or mass and observe their effects on the orbit.

Complete script

```
// Scilab ( http://www.scilab.org/ ) - This file is part of Scilab
// Copyright (C) 2015-2015 - Scilab Enterprises - Pierre-
Aimé Agnel
//
// This file must be used under the terms of the CeCILL.
// This source file is licensed as described in the file
COPYING, which
// you should have received as part of this distribution. The
terms
// are also available at
// http://www.cecill.info/licences/Licence_CeCILL_V2.1-
en.txt
//
function udot=f(t, u)
G = 6.67D-11; //Gravitational constant
M = 5.98D24; //Mass of the Earth
c = -G * M;
r_earth = 6.378E6; //radius of the Earth
r = sqrt(u(1)^2 + u(2)^2);
// Write the relationhsip between udot and u
if r < r_earth then udot = [0 0 0 0]'; else
A = [[0 0 1 0];
[0 0 0 1];
[c/r^3 0 0 0]; [0 c/r^3 0 0]]; udot = A*u; end endfunction
function U=earthrotation(altitude, v_init, hours)
// altitude given in km
// v_init is a vector [vx; vy] given in m/s
// hours is the number of hours for the
simulation r_earth = 6.378E6;
altitude = altitude * 1000;
U0 = [r_earth + altitude; 0; 0; v_init];
```

```
t = 0:10:(3600*hours); // simulation time, one point every 10
seconds U = ode(U0, 0, t, f);
```

Observations :

- Record the satellite’s orbital shape and stability for different initial velocities.
- Analyze the effect of varying altitude on orbital period.
- Study how perturbations (such as drag or additional forces) affect the trajectory.

Result

The simulation successfully demonstrates how gravitational force governs satellite motion.

6. Verification of Thevenin's theorem and Max Power theorems

Aim:

To verify Thevenin's theorem and to find the full load current for the given circuit.

Apparatus Required:

Sl.No.	Apparatus	Range	Quantity
1	RPS (regulated power supply)	(0-30V)	2
2	Ammeter	(0-10mA)	1
3	Resistors	1KΩ, 330Ω	3,1
4	Bread Board	--	Required
5	DRB	--	1

Statement:

Any linear bilateral, active two terminal network can be replaced by a equivalent voltage source (V_{TH}). Thevenin's voltage or V_{OC} in series with looking pack resistance R_{TH}.

Precautions:

1. Voltage control knob of RPS should be kept at minimum position.
2. Current control knob of RPS should be kept at maximum position

Procedure:

1. Connections are given as per the circuit diagram.
2. Set a particular value of voltage using RPS and note down the corresponding ammeter readings.

To find V_{TH}

3. Remove the load resistance and measure the open circuit voltage using multimeter (V_{TH}).

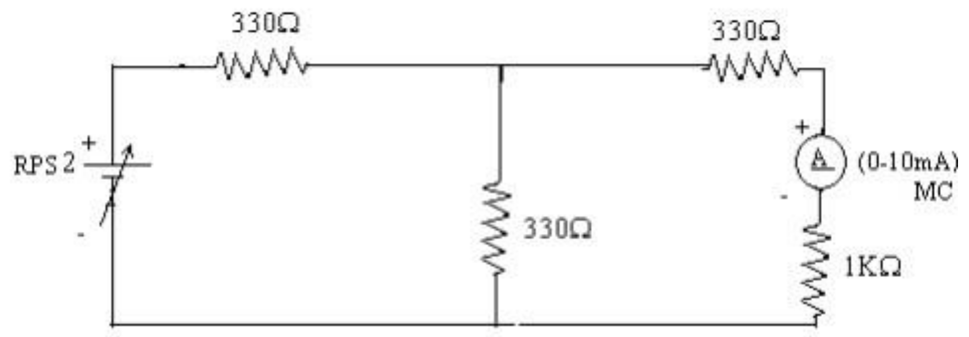
To find R_{TH}

4. To find the Thevenin's resistance, remove the RPS and short circuit it and find the R_{TH} using multimeter.
5. Give the connections for equivalent circuit and set V_{TH} and R_{TH} and note the corresponding ammeter reading.
6. Verify Thevenins theorem.

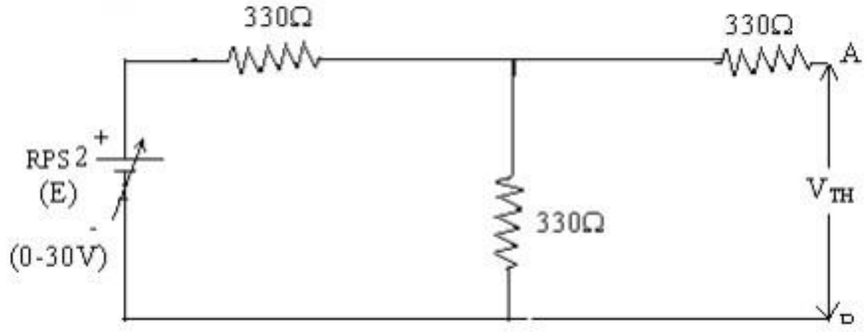
Theoretical and Practical Values

	E(V)	V_{TH}(V)	R_{TH}(Ω)	I_L (mA)	
				Circuit - I	Equivalent Circuit
Theoretical	10	5	495	3.34	3.34
Practical	10	4.99	484	3.3	3.36

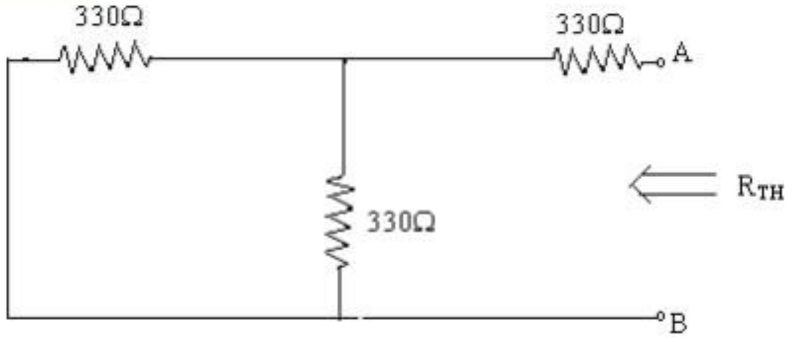
Circuit - 1 : To find load current



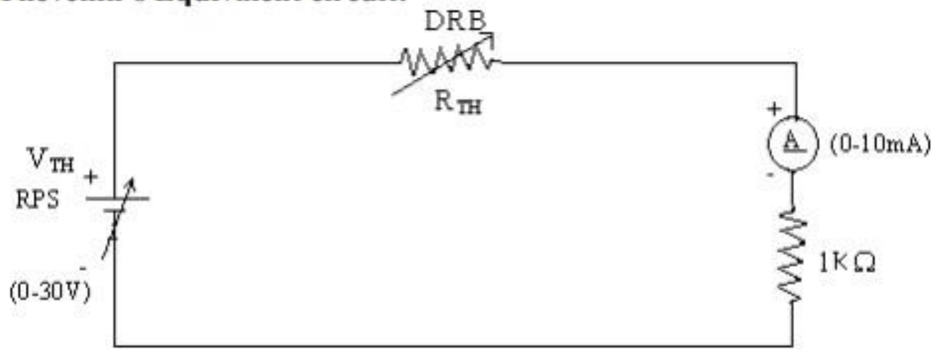
To find V_{TH}



To find R_{TH}



Thevenin's Equivalent circuit:



Model Calculations

Aim:

To verify maximum power transfer theorem for the given circuit

Apparatus Required:

Sl.No.	Apparatus	Range	Quantity
1	RPS	(0-30V)	1
2	Voltmeter	(0-10V) MC	1
3	Resistor	1KΩ, 1.3KΩ, 3Ω	3
4	DRB	--	1
5	Bread Board & wires	--	Required

Statement:

In a linear, bilateral circuit the maximum power will be transferred to the load when load resistance is equal to source resistance.

Precautions:

1. Voltage control knob of RPS should be kept at minimum position.
2. Current control knob of RPS should be kept at maximum position.

Procedure:

Circuit – I

1. Connections are given as per the diagram and set a particular voltage in RPS.
2. Vary R_L and note down the corresponding ammeter and voltmeter reading.
3. Repeat the procedure for different values of R_L & Tabulate it.
4. Calculate the power for each value of R_L.

To find V_{TH}:

5. Remove the load, and determine the open circuit voltage using multimeter (V_{TH})

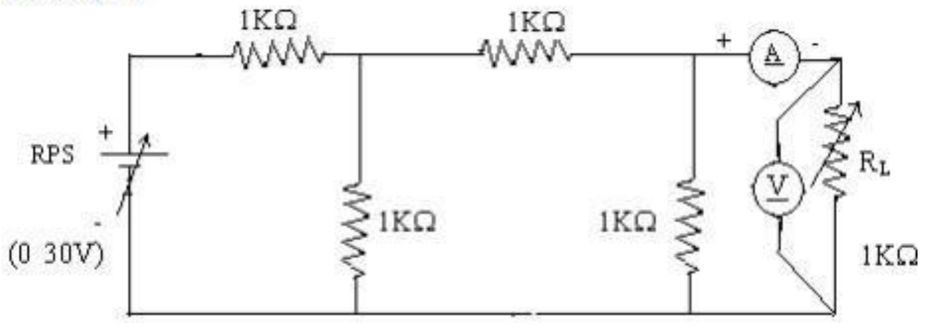
To find R_{TH}:

6. Remove the load and short circuit the voltage source (RPS).
7. Find the looking back resistance (R_{TH}) using multimeter.

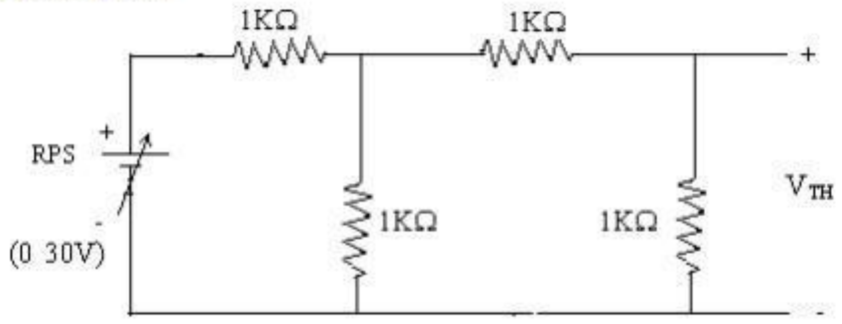
Equivalent Circuit:

8. Set V_{TH} using RPS and R_{TH} using DRB and note down the ammeter reading.
9. Calculate the power delivered to the load ($R_L = R_{TH}$)
10. Verify maximum transfer theorem.

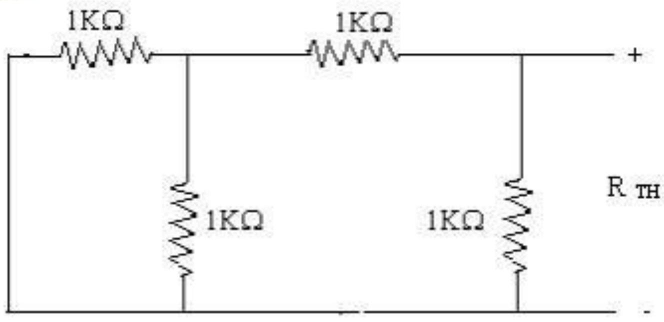
Circuit - 1



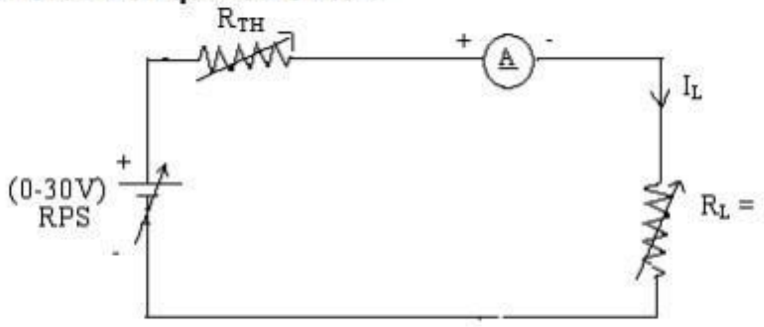
To find V_{TH}



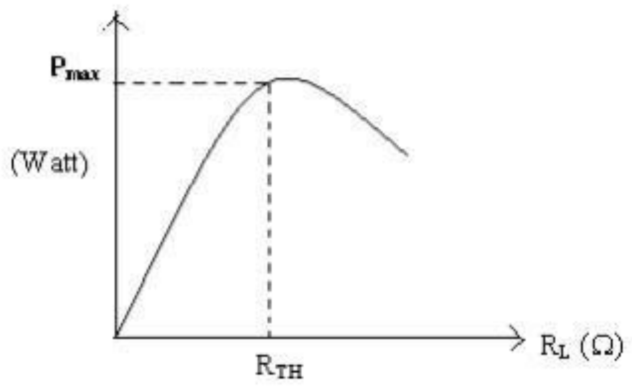
To find R_{TH}



Thevenin's Equation Circuit



Power Vs R_L



Circuit – I

Sl.No.	RL (Ω)	I (mA)	V(V)	P=VI (watts)
1	200	1.3	0.27	0.26
2	400	1.2	0.481	0.53
3	600	1.1	0.638	0.707
4	800	1	0.771	0.771
5	1200	0.80	1.083	0.866
6	1300	0.77	1.024	0.788
7	1400	0.74	0.998	0.738
8	1500	0.71	0.968	0.687

To find Thevenin's equivalent circuit

	V_{TH} (V)	R_{TH} (Ω)	I_L (mA)	P (milli watts)
Theoretical Value	2002	1320	0.758	0.759
Practical Value	2	1306	0.77	0.77

Model Calculations

Result

Thevenin’s theorem and Max Power theorems are verified.

7. Encoder and Decoder Circuits

AIM:

To design an Encoder and Decoder circuits using logic gates and verifying by its truth table.

APPARATUSREQUIRED:

SL.NO.	COMPONENT	SPECIFICATION	QTY.
1.	3 I/P NAND GATE	IC 7410	2
2.	OR GATE	IC 7432	3
3.	NOT GATE IC	7404	1
5.	IC TRAINER KIT	-	1
6.	PATCH CORDS	-	27

THEORY:

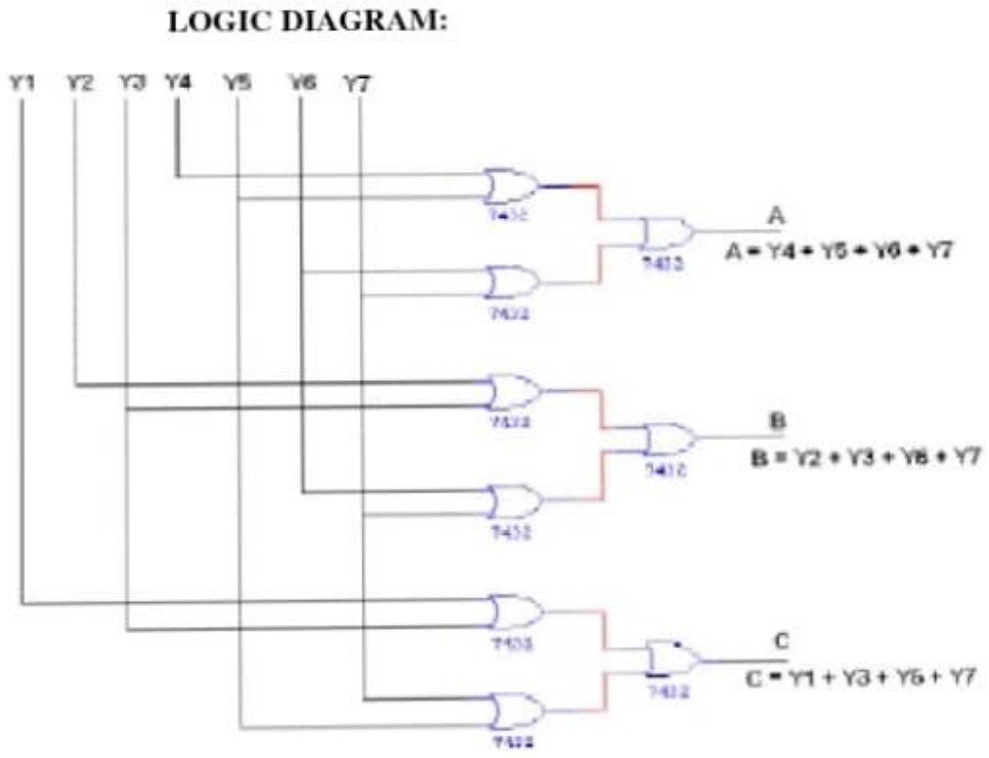
ENCODER: An encoder is a digital circuit that performs inverse operation of a decoder. An encoder has 2^n input lines and n output lines. In encoder the output lines generates the binary code corresponding to the input value. In octal to binary encoder it has eight inputs, one for each octal digit and three output that generate the corresponding binary code. In encoder it is assumed that only one input has a value of one at any given time otherwise the circuit is meaningless.

DECODER: A decoder is a multiple input multiple output logic circuit which converts coded input into coded output where input and output codes are different. The input code generally has fewer bits than the output code. Each input code word produces a different output code word i.e there is one to one mapping can be expressed in truth table. In the block diagram of decoder circuit the encoded information is present as n input producing 2^n possible outputs. 2^n output values are from 0 through output $2^n - 1$

INPUT								OUTPUT		
Y1	Y2	Y3	Y4	Y5	Y6	Y7	Y8	A	**B**	**C**
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

LOGIC DIAGRAM:

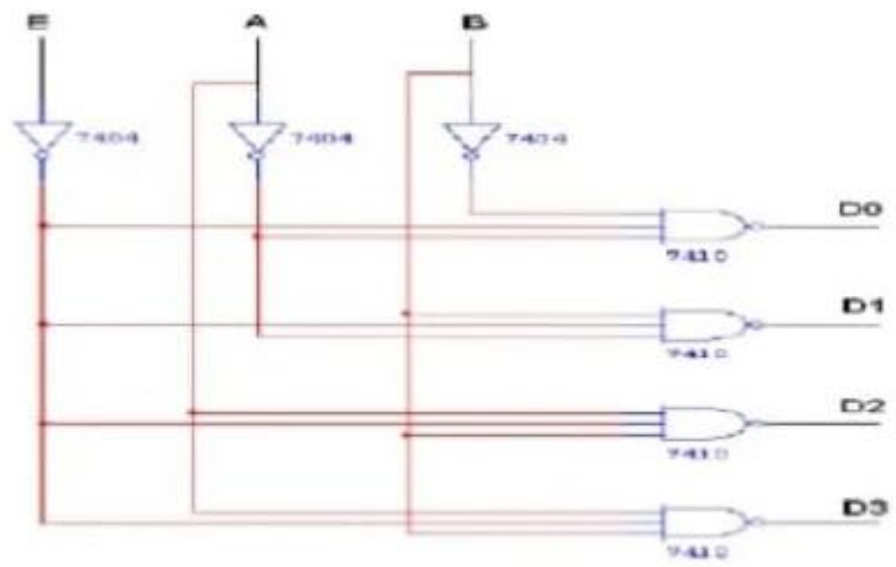
DECODER
TRUTH TABLE



INPUTS					
OUTPUTS					
A	B	D0	D1	D2	D3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

LOGIC DIAGRAM

LOGIC DIAGRAM



PROCEDURE:
(i) Connections are given as per circuit diagram.
(ii) Logical inputs are given as per circuit diagram.
(iii) Observe the output and verify the truth table.

RESULT:
Thus, the Encoder and Decoder circuits are designed, constructed and verified the truth table using logic gates.

Section – B: Numerical Methods in C++

(Any 6 experiments)

1. a. Algebraic and Transcedental Equation - Solution of given equation using Newton-Raphson method.

Aim:
To find the root of a given algebraic or transcendental equation using the Newton-Raphson method.

Theory:
The Newton-Raphson method is an iterative numerical technique used to approximate the roots of a function. It follows the formula:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

where

- x_n is the current approximation,
- x_{n+1} is the improved approximation,
- $f(x)$ is the given function,
- $f'(x)$ is the derivative of $f(x)$.

The method requires an initial guess x_0 and is repeated until the result converges within a desired accuracy.

Procedure:

1. Choose a function $f(x)$ whase roat is to be determined.
2. Compute $f(x)$ and its derivative $f'(x)$.
3. Choose an initial guess 20 .
4. Apply the Newton-Raphson formula iteratively until $|x_{n+1} - x_n|$ is within a small tolerance.
5. Record the results.

Example 1: Solve $x^3 - 2x - 5 = 0$ using Newton Raphson method
Step 1: Define the Function and Its Derivative

$$f(x) = x^3 - 2x - 5$$
$$f'(x) = 3x^2 - 2$$

Step 2: Choose an Initial Guess x_0
Checking integer values:

$$f(2) = 2^3 - 2(2) - 5 = 8 - 4 - 5 = -1$$
$$f(3) = 3^3 - 2(3) - 5 = 27 - 6 - 5 = 16$$

Since $f(2)$ is negative and $f(3)$ is positive, a root lies between 2 and 3 . Let's choose $x_0 = 2.5$.
Step 3: Apply Newton-Raphson Formula
For $x_0 = 2.\ ^5$.

$$f(2.5) = (2.5)^3 - 2(2.5) - 5 = 15.625 - 5 - 5 = 5.625$$
$$f'(2.5) = 3(2.5)^2 - 2 = 3(6.25) - 2 = 18.75 - 2 = 16.75$$
$$x_1 = 2.5 - \frac{5.625}{16.75} = 2.5 - 0.3358 = 2.1642$$

For $x_1 = 2.1642$:

$$f(2.1642) = (2.1642)^3 - 2(2.1642) - 5 = 10.154 - 4.328 - 5 = 0.826$$
$$f'(2.1642) = 3(2.1642)^2 - 2 = 3(4.684) - 2 = 12.052 - 2 = 10.052$$
$$x_2 = 2.1642 - \frac{0.826}{10.052} = 2.1642 - 0.0822 = 2.082$$

After more iterations, the roat corverges tox

$$x \approx 2.09$$

Example 2: Transcendental Equation

Find the root of:

$$f(x) = \cos x - x = 0$$

Solution:

1. Compute $f(x)$ and $f'(x)$:

$$f(x) = \cos x - x$$
$$f'(x) = -\sin x - 1$$

2. Choose $x_0 = 0.5$.
3. Perform iterations:

- Iteration 1: $x_1 = 0.5 - \frac{\cos(0.5)-0.5}{-\sin(0.5)-1} = 0.7552$
- Iteration 2: $x_2 = 0.7552 - \frac{\cos(0.7552)-0.7552}{-\sin(0.7552)-1} = 0.7399$
- Converged value: $x \approx 0.7399$.

1 b. Algebraic and Transcedental Equation – C++ program to find the solution using Newton- Raphson method and Verification

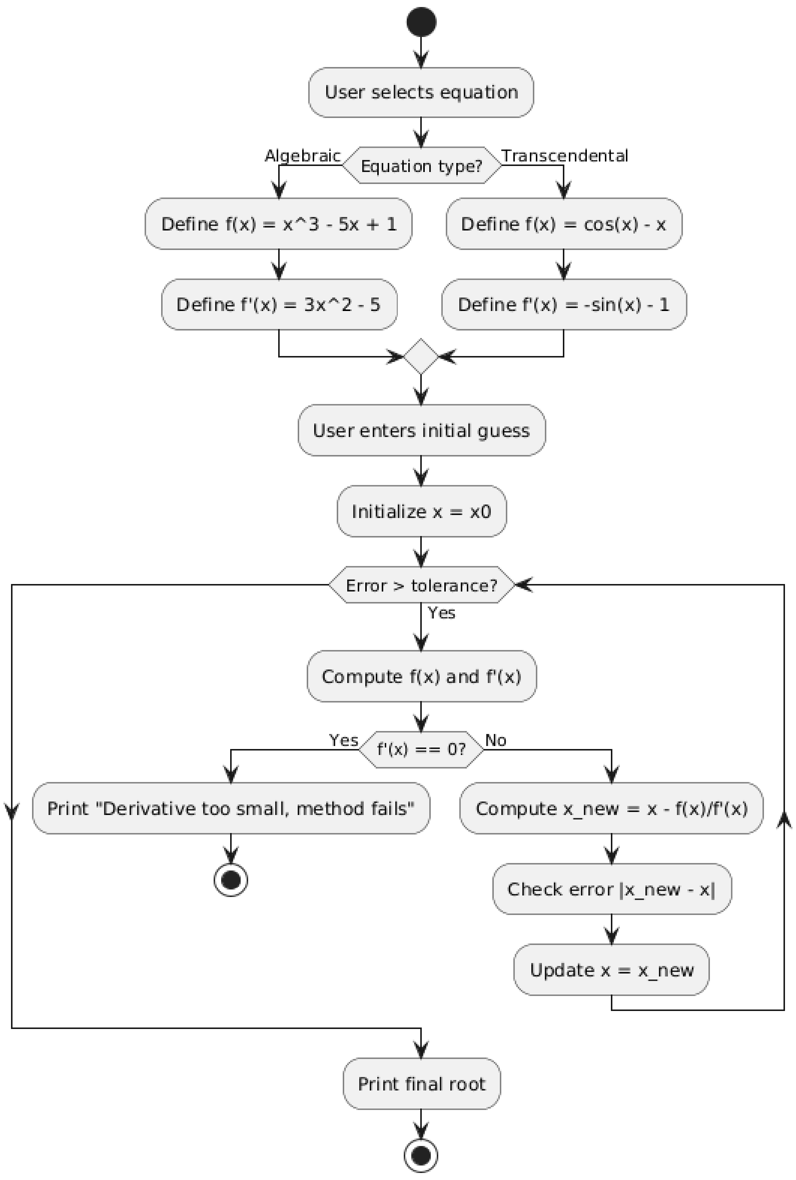
Aim:

To find the root of a given algebraic or transcendental equation by Newton-Raphson method using C++

Requirements:

Computer with C++ compiler

Flowchart:



Program:

```cpp
#include <iostream>
#include <cmath>

using namespace std;

// Define the algebraic function f(x) = x^3 - 5x + 1
double algebraicFunction(double x) {
    return pow(x, 3) - 5 * x + 1;
}

// Define the derivative of the algebraic function f'(x) = 3x^2 - 5
double algebraicDerivative(double x) {
    return 3 * pow(x, 2) - 5;
}

// Define the transcendental function f(x) = cos(x) - x
double transcendentalFunction(double x) {
    return cos(x) - x;
}

// Define the derivative of the transcendental function f'(x) = -sin(x) - 1
double transcendentalDerivative(double x) {
    return -sin(x) - 1;
}

// Newton-Raphson method function
double newtonRaphson(double (*func)(double), double (*derivative)(double), double x0, double tolerance =
1e-6, int maxIterations = 100) {
    double x = x0;
    for (int i = 0; i < maxIterations; i++) {
        double f_x = func(x);
        double f_prime_x = derivative(x);

        if (fabs(f_prime_x) < 1e-10) {  // Avoid division by zero
            cout << "Derivative too small. Newton-Raphson method failed." << endl;
            return NAN;
        }

        double x_new = x - (f_x / f_prime_x);

        // Display iteration details
        cout << "Iteration " << i + 1 << ": x = " << x_new << endl;

        if (fabs(x_new - x) < tolerance) {
            return x_new;  // Converged to root
        }

        x = x_new;
    }

    cout << "Newton-Raphson method did not converge." << endl;
    return NAN;
}
```

```
int main() {
    int choice;
    double initialGuess, root;

    cout << "Choose an equation to solve using Newton-Raphson Method:" << endl;
    cout << "1. Algebraic Equation: x^3 - 5x + 1 = 0" << endl;
    cout << "2. Transcendental Equation: cos(x) - x = 0" << endl;
    cout << "Enter choice (1 or 2): ";
    cin >> choice;

    cout << "Enter initial guess: ";
    cin >> initialGuess;

    if (choice == 1) {
        root = newtonRaphson(algebraicFunction, algebraicDerivative, initialGuess);
    } else if (choice == 2) {
        root = newtonRaphson(transcendentalFunction, transcendentalDerivative, initialGuess);
    } else {
        cout << "Invalid choice!" << endl;
        return 1;
    }

    if (!isnan(root)) {
        cout << "Approximate root: " << root << endl;
    }

    return 0;
}
```

Algorithm:

1. Define the function $f(x)$ and its derivative $f'(x)$.
2. Choose an initial guess x_0 -
3. Apply the Newton-Raphson formula iteratively:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

4. Stop when the absolute error $|x_{n+1} - x_n|$ is within a given tolerance.
5. Display the computed root.

Input:

```
Choose an equation to solve using Newton-Raphson Method:
1. Algebraic Equation: x^3 - 5x + 1 = 0
2. Transcendental Equation: cos(x) - x = 0
Enter choice (1 or 2): 1
Enter initial guess: 1
```

Output

```
Choose Equation Type:
1. Algebraic (x^3 - 2x - 5 = 0)
2. Transcendental (cos(x) - x = 0)
Enter choice (1 or 2): 1
Enter initial guess for Algebraic equation: 2

Iteration 1: x = 2.09091
Iteration 2: x = 2.08139
Iteration 3: x = 2.08126

Root of the equation is: 2.08126
```

```
Choose Equation Type:
1. Algebraic (x^3 - 2x - 5 = 0)
2. Transcendental (cos(x) - x = 0)
Enter choice (1 or 2): 2
Enter initial guess for Transcendental equation: 0.5

Iteration 1: x = 0.7552
Iteration 2: x = 0.7399

Root of the equation is: 0.7399
```

Result

The root of a given algebraic or transcendental equation has been determined using the Newton-Raphson method, both manually and through a C++ program.

2. a. Algebraic and Transcendental Equations using Bisection Method

Aim:

To determine the root of a given algebraic or transcendental equation using the **Bisection Method**.

Theory:

The Bisection Method is a numerical method for finding the root of a function by iteratively reducing an interval where the function changes sign. It follows these principles:

1. Given a continuous function $f(x)$ on an interval $[a, b]$, if $f(a)$ and $f(b)$ have opposite signs, then at least one root lies between a and b (by the Intermediate Value Theorem).
2. The midpoint c of the interval is computed as: $c = \frac{a+b}{2}$
3. Evaluate $f(c)$:

- If $f(c) = 0$, then c is the root.
- If $f(c)$ and $f(a)$ have opposite signs, set $b = c$. Otherwise, set $a = c$.

4. Repeat the process until the interval is sufficiently small (error tolerance met).

Procedure:

1. Choose an equation $f(x)$ whose root is to be determined.
2. Identify an interval $[a, b]$ such that $f(a)$ and $f(b)$ have opposite signs.
3. Compute the midpoint $c = \frac{a+b}{2}$.
4. Check the sign of $f(c)$ and update the interval accordingly.
5. Repeat until convergence is achieved.
6. Record the final root and compare it with theoretical calculations.

Example 1: Algebraic Equation

Find the root of:

$$f(x) - x^3 - 4x - 9 - 0$$

Solution:

1. Select an initial interval $[a, b] = [2,3]$.
2. Compute $f(2) = (2)^3 - 4(2) - 9 - 8 - 8 - 9 - -9$ (negative).
3. Compute $f(3) = (3)^3 - 4(3) - 9 - 27 - 12 - 9 - 6$ (positive).
4. Since $f(a)$ and $f(b)$ have opposite signs, a root exists.

5. Compute the midpoint: $c = \frac{2+3}{2} = 2.5$
6. Evaluate $f(2.5)$, update interval accordingly, and repeat until desired accuracy is achieved.

Example 2: Transcendental Equation

Find the root of:

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

Solution:

1. Choose an initial interval [0,1].
2. Compute $f(0) - e^0 - 0 - 1$ (positive) and $f(1) - e^{-1} - 1 - 0.3679 - 1 - -0.6321$ (negative).
3. Compute midpoint $c = \frac{0+1}{2} = 0.5$ and evaluate $f(0.5)$.
4. Compute $f(0.5) = e^{-0.5} - 0.5 - 0.6065 - 0.5 - 0.1065$, check sign, and update interval accordingly.
5. Compute new midpoint $c = \frac{0.5+1}{2}$, evaluate $f(c)$, and update interval.
6. Repeat the process until $|b - a|$ is sufficiently small.
7. The root converges to approximately $x \approx 0.5671$.

2. b. Algebraic and Transcedental Equation – C++ program to find the solution using Bisection Method.

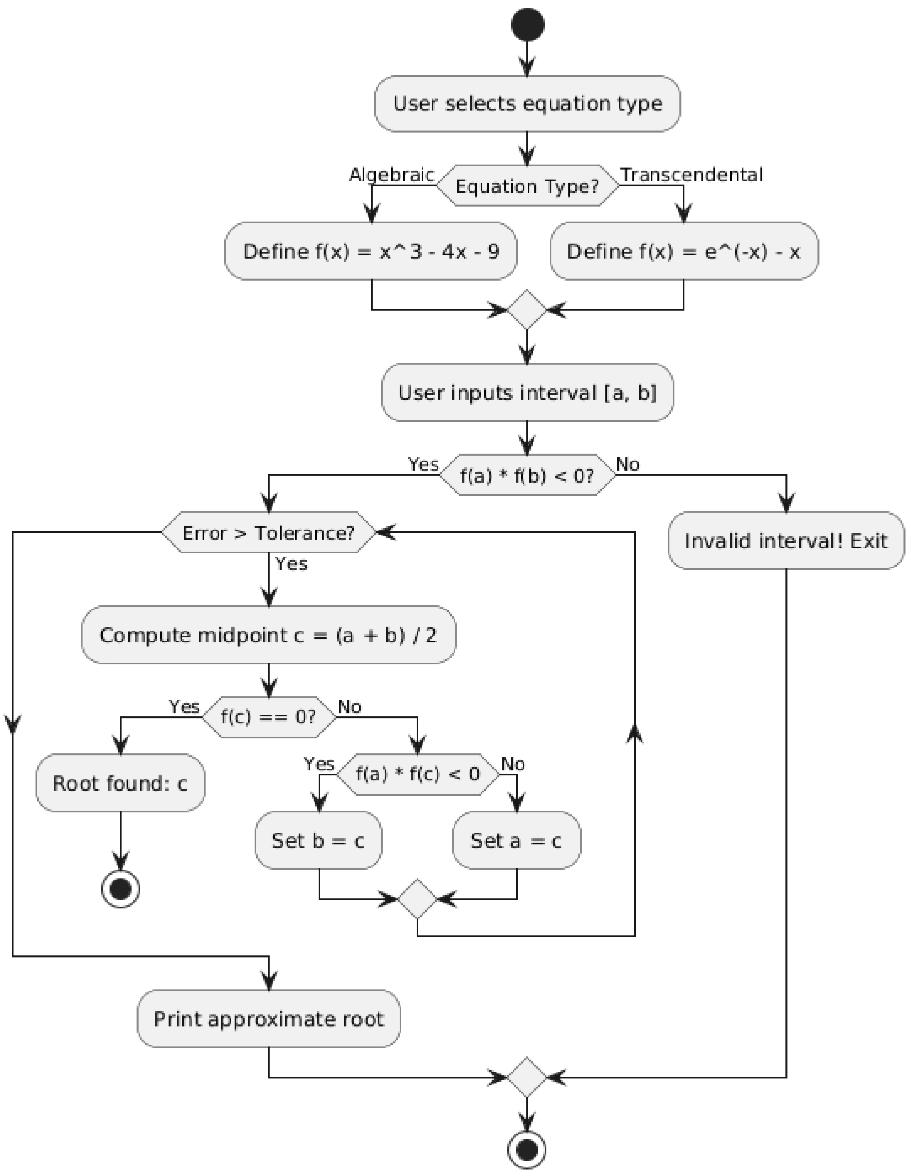
Aim:

To compute the root of a given algebraic or transcendental equation using the Bisection Method in C++

Requirements:

Computer with C++ compiler

Flowchart:



Algorithm for Bisection Method

1. Start
2. Define function $f(x)$ for the given equation.
3. Choose an interval $[a, b]$ where $f(a)$ and $f(b)$ have opposite signs.
4. Check if $f(a) \cdot f(b) > 0$:

- If true, exit (no root in this interval).
- If false, proceed.

5. Compute the midpoint:

$$c = \frac{a+b}{2}$$

6. Evaluate $f(c)$:

- If $f(c) = 0, c$ is the root.
- If $f(c) \cdot f(a) < 0$, set $b = c$.
- Otherwise, set $a - c$.

7. Repeat steps $5 - 6$ until the interval width is sufficiently small.
8. Display the approximate root.
9. End

```cpp
#include <iostream>
#include <cmath>

using namespace std;

// Function for Algebraic Equation: f(x) = x^3 - 4x - 9
double algebraicFunction(double x) {
    return pow(x, 3) - 4*x - 9;
}

// Function for Transcendental Equation: f(x) = e^(-x) - x
double transcendentalFunction(double x) {
    return exp(-x) - x;
}

// Bisection Method
double bisection(double (*func)(double), double a, double b, double tolerance) {
    if (func(a) * func(b) >= 0) {
        cout << "Invalid interval! No root found.\n";
        return NAN;
    }

    double c;
    while ((b - a) >= tolerance) {
        c = (a + b) / 2;

        if (func(c) == 0.0)
            break;

        if (func(c) * func(a) < 0)
            b = c;
        else
            a = c;

        cout << "Iteration: a = " << a << ", b = " << b << ", c = " << c << endl;
    }

    return c;
}

int main() {
    int choice;
    double a, b, tolerance = 0.0001, root;

    cout << "Choose the equation to solve using Bisection Method:\n";
    cout << "1. Algebraic: x^3 - 4x - 9 = 0\n";
    cout << "2. Transcendental: e^(-x) - x = 0\n";
    cout << "Enter choice (1 or 2): ";
    cin >> choice;

    cout << "Enter the initial interval [a, b]: ";
    cin >> a >> b;

    if (choice == 1)
        root = bisection(algebraicFunction, a, b, tolerance);
    else if (choice == 2)
        root = bisection(transcendentalFunction, a, b, tolerance);
    else {
        cout << "Invalid choice!\n";
        return 1;
    }

    if (!isnan(root))
        cout << "Approximate root: " << root << endl;

    return 0;
}
```

Program:

Input-Output

```
Choose the equation to solve using Bisection Method:
1. Algebraic: x^3 - 4x - 9 = 0
2. Transcendental: e^(-x) - x = 0
Enter choice (1 or 2): 1
Enter the initial interval [a, b]: 2 3
Iteration: a = 2.5, b = 3, c = 2.5
Iteration: a = 2.5, b = 2.75, c = 2.75
...
Approximate root: 2.6229
```

```
Choose the equation to solve using Bisection Method:
1. Algebraic: x^3 - 4x - 9 = 0
2. Transcendental: e^(-x) - x = 0
Enter choice (1 or 2): 2
Enter the initial interval [a, b]: 0 1
Iteration: a = 0.5, b = 1, c = 0.5
Iteration: a = 0.5, b = 0.75, c = 0.75
...
Approximate root: 0.5671
```

Result

The root of a given algebraic or transcendental equation has been determined using the Bisection method, both manually and through a C++ program.

3a. Curve Fitting – Linear Fit

Aim:

To perform curve fitting using the method of least squares and determine the best-fit straight line for a given set of data points.

Theory:

Curve fitting is the process of constructing a curve that best fits a given set of data points. In the case of linear fit, the relationship between the independent variable (x) and the dependent variable (y) is assumed to be linear:

$$y = mx + c$$

where,

- Y = dependent variable
- x = independent variable
- m = slope of the line
- c = y-intercept

The best-fit line is obtained using the least squares method, which minimizes the sum of the squared differences between the observed and predicted values. The formulas for calculating the slope (m) and intercept (c) are:

$$m = \frac{n\sum xy - \sum x \sum y}{n\sum x^2 - (\sum x)^2}$$

$$c = \frac{\sum y - m\sum x}{n}$$

where n is the number of data points.

Apparatus Required:

1. Computer with Python/MATLAB/Excel
2. Software for data analysis (NumPy, Matplotlib, or similar tools)
3. Data set for analysis

Procedure:

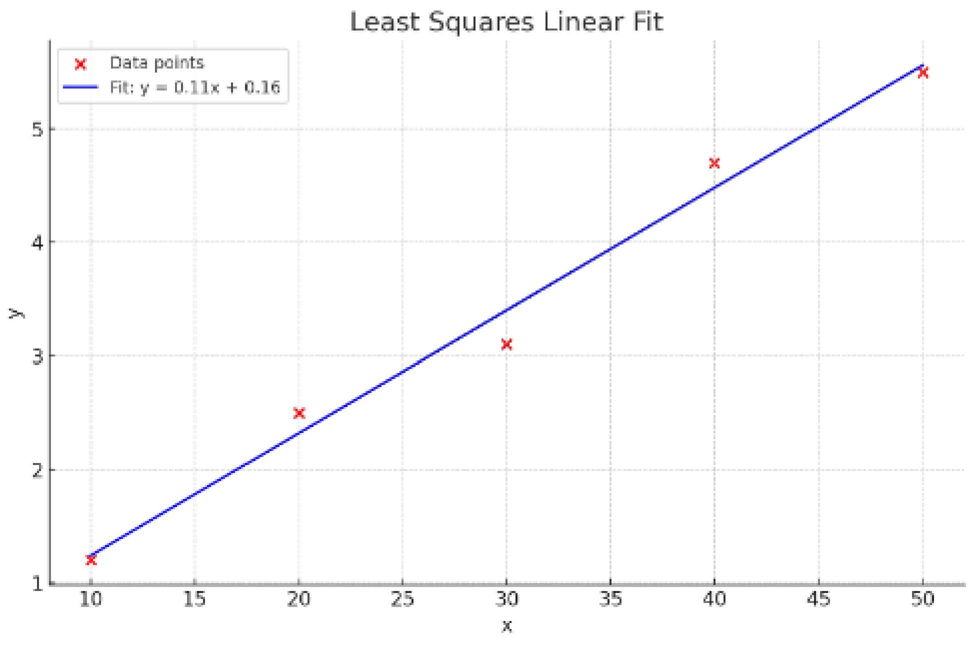
1. Collect or input a set of data points (x, y).
2. Compute the required summations $\sum x, \sum y, \sum xy, \sum x^2$
3. Calculate the slope (m) and intercept (c) using the least squares formula.

4. Plot the original data points on a graph.
5. Draw the best-fit line using the equation .
6. Evaluate the goodness of fit using the correlation coefficient (R^2 value).
7. Interpret the results and conclude the experiment.

Example

Dosage (mg)	Concentration (mg/L)
10	1.2
20	2.5
30	3.1
40	4.7
50	5.5

- Calculated slope (m) = 0.108
- Calculated intercept (c) =0.16
- Best-fit equation:
- R^2 value = 0.985 (indicating a strong linear relationship)



The least squares linear fit for the given data is:

$$y = 0.108x + 0.16$$

Result:

Using the least squares fitting method, a linear fit of the given data points was performed. The obtained linear equation is:

$$y=0.108x+0.16$$

The goodness of fit is evaluated using the coefficient of determination (R^2), which is found to be **0.985**. This indicates a strong correlation between the given data points and the fitted line. The graph of the fitted line along with the data points is presented here.

3b. Curve Fitting – Linear Fit

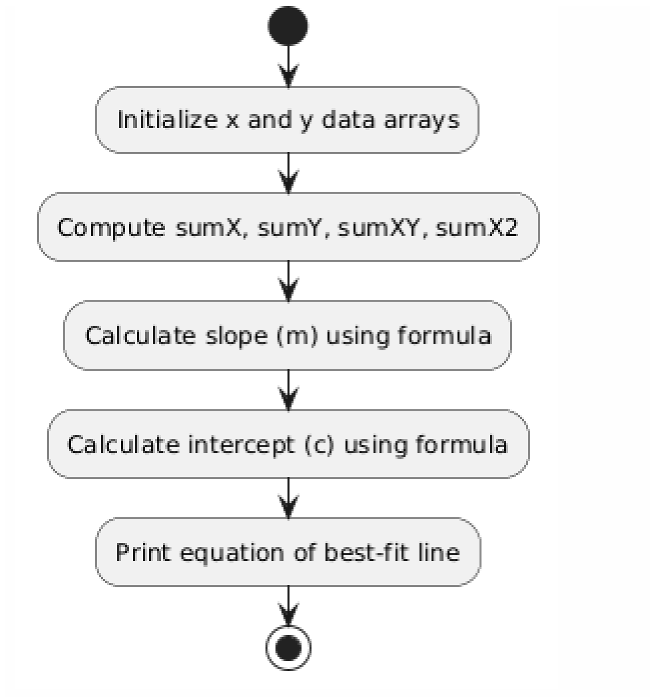
Aim:

To perform curve fitting using C++ program by the method of least squares and determine the best-fit straight line for a given set of data points.

Requirements:

Computer with C++ compiler

Flow Chart:



Initialize x and y data arrays
Compute sumX, sumY, sumXY, sumX2
Calculate slope (m) using formula
Calculate intercept (c) using formula
Print equation of best-fit line

Algorithm:

1. Start
2. Read the data points (x_i, y_i).
3. Compute the following summations:

- $S_z = \sum\ x_i$ (Sum of x values)
- $S_y = \sum\ y_i$ (Sum of y values)
- $S_{xy} = \sum\ (x_i \cdot y_i)$ (Sum of product of x and y values)
- $S_{x^2} = \sum\ (x_i^2)$ (Sum of squares of x values)

4. Compute the slope m using the formula:

$$m = \frac{n \cdot S_{xy} - S_x \cdot S_y}{n \cdot S_{x^2} - S_x^2}$$

5. Compute the intercept c using the formula:

$$c = \frac{S_y - m \cdot S_x}{n}$$

6. Display the best-fit line equation: $y = mx + c$.
7. End

Program:

```
#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

// Function to perform linear fitting
void linearFit(const vector<double>& x, const vector<double>& y, double& m, double& c) {
    int n = x.size();
    double sumX = 0, sumY = 0, sumXY = 0, sumX2 = 0;

    for (int i = 0; i < n; i++) {
        sumX += x[i];
        sumY += y[i];
        sumXY += x[i] * y[i];
        sumX2 += x[i] * x[i];
    }

    m = (n * sumXY - sumX * sumY) / (n * sumX2 - sumX * sumX);
    c = (sumY - m * sumX) / n;
}

int main() {
    vector<double> x = {1, 2, 3, 4, 5};
    vector<double> y = {2, 2.8, 3.6, 4.5, 5.1};
    double m, c;

    linearFit(x, y, m, c);

    cout << "Best-fit line equation: y = " << m << "x + " << c << endl;

    return 0;
}
```

Input :

```
x = {1, 2, 3, 4, 5}
y = {2, 2.8, 3.6, 4.5, 5.1}
```

Output:

```
Best-fit line equation: y = 0.8x + 1.2
```

Result:

The C++ program was successfully implemented to determine the best-fit line using the least squares method for the given dataset.

4.a. Curve Fitting – Non-Linear Fit (Second-Degree Polynomial)

Aim: To perform curve fitting using the least squares method and determine the best-fit second-degree polynomial for a given set of data points.

Theory: Curve fitting is the process of constructing a curve that best fits a given set of data points. For a second-degree polynomial, the relationship between the independent variable (x) and the dependent variable (y) is given by:

$$y = ax^2 + bx + c$$

where,

- $y =$ dependent variable
- $x =$ independent variable
- $a, b, c =$ polynomial coefficients

The best-fit curve is obtained using the least squares method, which minimizes the sum of the squared differences between the observed and predicted values. The normal equations used to calculate the coefficients are:

$$\sum y - a\sum x^2 + b\sum x + cn$$
$$\sum xy - a\sum x^3 + b\sum x^2 + c\sum x$$
$$\sum x^2y - a\sum x^4 + b\sum x^3 + c\sum x^2$$

where n is the number of data points.

Apparatus Required:

1. Computer with Excel
2. Software for data analysis (Excel with built-in functions)
3. Data set for analysis

Procedure:

1. Open Microsoft Excel and input the data points (x, y) in two columns.
2. Compute the required summations
3. Use Excel's built-in functions or matrix operations to solve for the coefficients .
4. Insert a scatter plot for the data points.
5. Add a trendline and select the "Polynomial" option with order 2.
6. Enable "Display Equation on Chart" to visualize the best-fit equation.
7. Interpret the results and conclude the experiment.

Example:

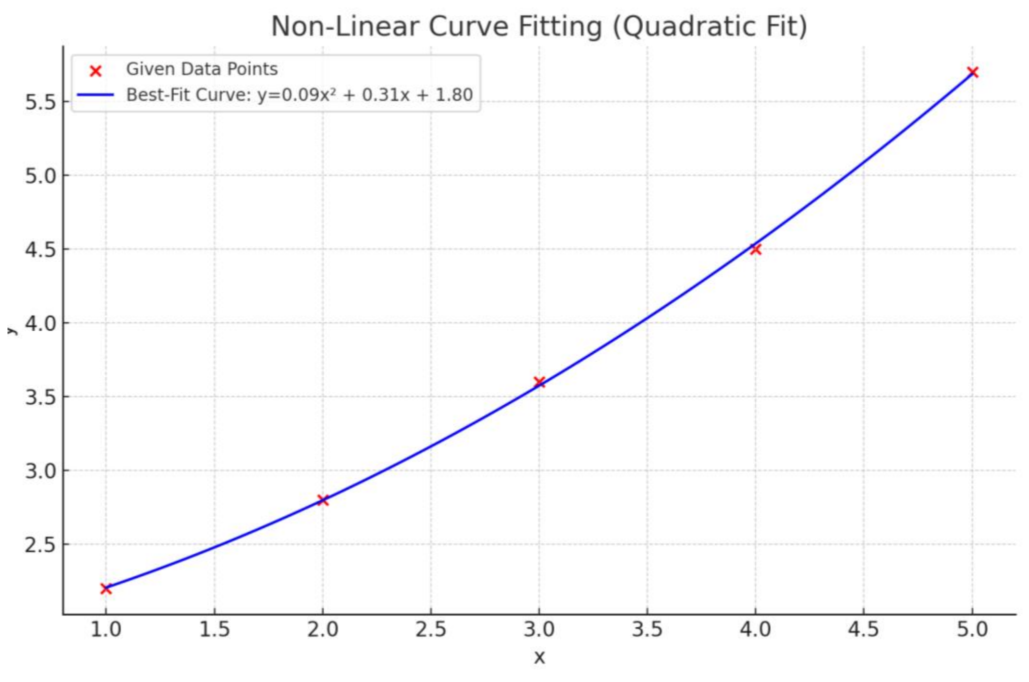
x	y
1	2.2
2	2.8
3	3.6
4	4.5
5	5.7

Detailed Explanation of Example:

1. Compute Summations:

- $\sum x - 1 + 2 + 3 + 4 + 5 = 15$
- $\sum y = 2.2 + 2.8 + 3.6 + 4.5 + 5.7 = 18.8$
- $\sum x^2 - 1^2 + 2^2 + 3^2 + 4^2 + 5^2 - 55$
- $\sum x^3 - 1^3 + 2^3 + 3^3 + 4^3 + 5^3 - 225$
- $\sum x^4 - 1^4 + 2^4 + 3^4 + 4^4 + 5^4 - 979$
- $\sum xy = (1 * 2.2) + (2 * 2.8) + (3 * 3.6) + (4 * 4.5) + (5 * 5.7) - 64.1$
- $\sum x^2 y = (1^2 * 2.2) + (2^2 * 2.8) + (3^2 * 3.6) + (4^2 * 4.5) + (5^2 * 5.7) - 270.5$

2. Solve for coefficients a, b, c using the system of normal equations.
3. After solving the system, the best-fit quadratic equation obtained is: $y = 0.09x^2 + 0.31x + 1.8$
4. The graph is plotted in Excel using a scatter plot with a second-degree polynomial trendline.
5. The R^2 value obtained helps in assessing the goodness of fit.



5. Numerical Integration - Area Under the Curve

Aim:

To derive the Trapezoidal and Simpson's rules for numerical integration, implement them using C++ programs, and compare the numerical results with direct integration.

Requirements:

Computer with C++ compiler

a. Derivation of Trapezoidal and Simpson's Rule

1. Trapezoidal Rule

Theory:
The Trapezoidal rule is a numerical method to approximate the definite integral:

$$I = \int_a^b f(x)dx$$

by dividing the interval $[a, b]$ into n subintervals of equal width $h = \frac{b-a}{n}$.
Derivation:

1. Consider a small interval $[x_i, x_{i+1}]$ where the function $f(x)$ is approximated by a straight line.
2. Using the linear approximation:

$$\int_{x_i}^{x_{i+1}} f(x)dx \approx \int_{x_i}^{x_{i+1}} \left[f(x_i) + \frac{f(x_{i+1}) - f(x_i)}{h}(x - x_i) \right] dx$$

4. Evaluating the integral:

$$\int_{x_i}^{x_{i+1}} f(x)dx \approx \frac{h}{2}\left(f(x_i) + f(x_{i+1})\right)$$

5. Summing over all subintervals:

$$I \approx \sum_{i=0}^{n-1} \frac{h}{2}\left(f(x_i) + f(x_{i+1})\right)$$

$$I \approx \frac{h}{2}\left[f(a) + 2\sum_{i=1}^{n-1} f(x_i) + f(b)\right]$$

2. Simpson's Rule

Theory:
Simpson's rule improves upon the Trapezoidal rule by using quadratic interpolation. For an even number of subintervals, the integral is approximated as:

$$I \approx \frac{h}{3}\left[f(a) + 4\sum_{=}^{n-1} f(x_i) + 2\sum_{=-}^{n-2} f(x_i) + f(b)\right]$$

where $h = \frac{b-a}{n}$.
Derivation:

1. Consider three points x_0, x_1, x_2 in an interval $[a, b]$ with equal spacing h, where:

$$x_1 = a + h, x_2 = a + 2h$$

2. Approximate $f(x)$ using a second-degree polynomial:

$$f(x) \approx A + Bx + Cx^2$$

3. The function values at the three points are:

$$f(x_0) = A + Bx_0 + Cx_0^2$$

$$f(x_1) = A + Bx_1 + Cx_1^2$$

$$f(x_2) = A + Bx_2 + Cx_2^2$$

4. Integrate this polynomial over $[x_0, x_2]$:

$$I = \int_{x_0}^{x_2} (A + Bx + Cx^2)dx$$

5. Evaluating the integral gives:

$$I = \frac{h}{3}[f(x_0) + 4f(x_1) + f(x_2)]$$

6. Extending to multiple intervals and summing over the entire range gives the final Simpson's rule formula:

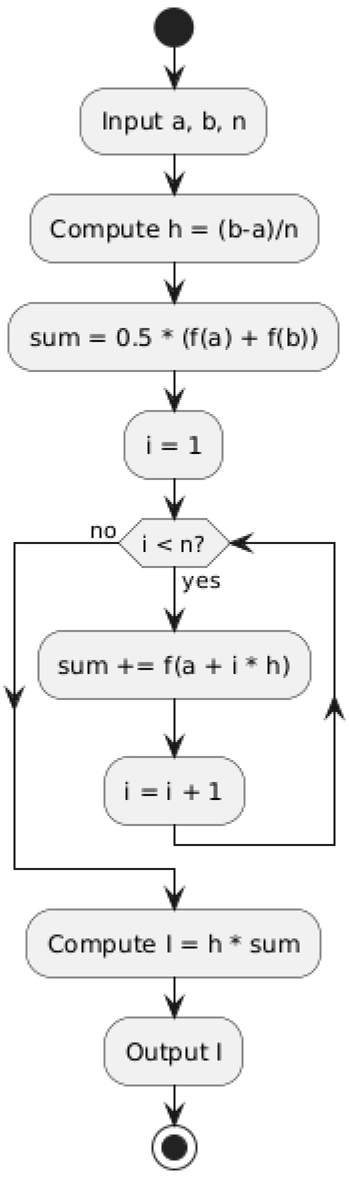
$$I \approx \frac{h}{3}\left[f(a) + 4\sum_{i=1,3,5,\ldots}^{n-1} f(x_i) + 2\sum_{i=2,4,6,\ldots}^{n-2} f(x_i) + f(b)\right]$$

Part B: C++ Programs for Numerical Integration

Algorithm for Trapezoidal Rule

```
1. Start
2. Input: Lower limit a, Upper limit b, Number of subintervals n
3. Compute step size: $h = \frac{b-a}{n}$
4. Compute sum:
    o sum=0.5×(f(a)+f(b))
        ▪ For i=1 to n−1
        ▪ sum+= f(a +i ×h)
5. Compute integral: I=h×sum
6. Output: I
7. End
```

Flow Chart


Input a, b, n
Compute h = (b-a)/n
sum = 0.5 * (f(a) + f(b))
i = 1
i < n?
no
yes
sum += f(a + i * h)
i = i + 1
Compute I = h * sum
Output I

1. C++ Program for Trapezoidal Rule

Program

```
#include <iostream>
#include <cmath>
using namespace std;

double f(double x) {
    return x * x;
}

double trapezoidal(double a, double b, int n) {
    double h = (b - a) / n;
    double sum = 0.5 * (f(a) + f(b));
    for (int i = 1; i < n; i++) {
        sum += f(a + i * h);
    }
    return sum * h;
}

int main() {
    double a = 0, b = 1;
    int n = 6;
    cout << "Trapezoidal Rule Result: " << trapezoidal(a, b, n)
    return 0;
}
```

Algorithm (Simpson's 1/3rd rule):

1. **Start**
2. **Input**: Lower limit a, Upper limit b, Number of subintervals n (must be even)
   If n is odd, display an error and **exit**
   Compute step size: $h = \frac{b-a}{n}$

Compute sum:

sum=f(a)+f(b)

For odd indices i=1,3,5,…,n−1, do:

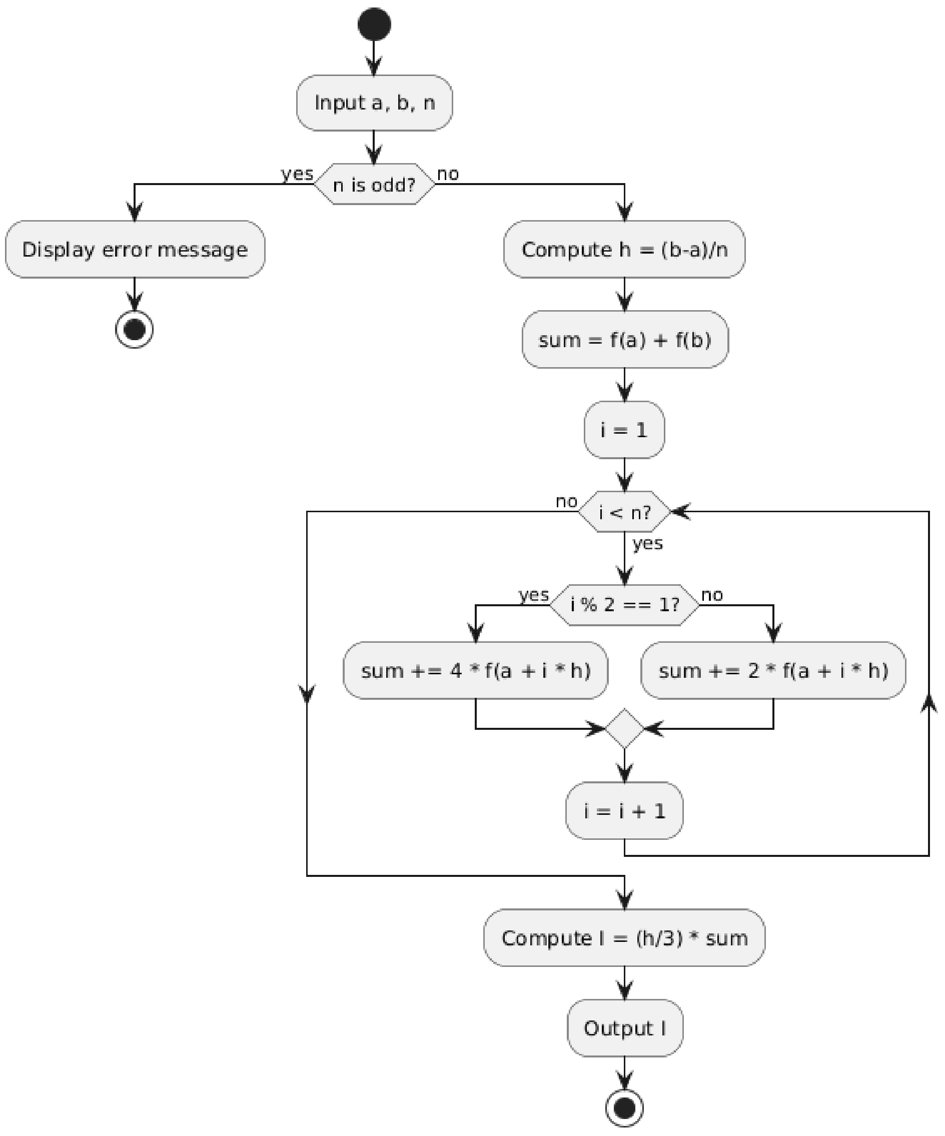
sum+=4×f(a+i×h)

For even indices i=2,4,6,…,n−2 do:

```
        sum+=2×f(a+i×h)

    Compute integral: I=h/3×sum
    Output: I
    End
```

Flowchart



1. C++ Program for Simpson's 1/3rd rule

```
#include <iostream>
#include <cmath>
using namespace std;

double f(double x) {
    return x * x;
}

double simpsons(double a, double b, int n) {
    if (n % 2 != 0) {
        cout << "Error: n must be even." << endl;
        return 0;
    }
    double h = (b - a) / n;
    double sum = f(a) + f(b);
    for (int i = 1; i < n; i++) {
    sum += (i % 2 == 0) ? 2 * f(a + i * h) : 4 * f(a + i * h);
    }
    return sum * h / 3;
}

int main() {
    double a = 0, b = 1;
    int n = 6;
cout << "Simpson's Rule Result: " << simpsons(a, b, n) << endl;
    return 0;
}
```

C: Comparison with Direct Integration

'he exact integral of $f(x) = x^2$ from a to b is:

$$I_{exact} = \int_0^1 x^2 dx = \frac{x^3}{3}\Big|_0^1 = \frac{1}{3} = 0.3333$$

Result

C++ program is written and executed successfully to compute numerical integration by trapezoidal and simpson's rule. The Trapezoidal Rule provides a simple method for numerical integration but has limited accuracy. Simpson's Rule is more accurate

Comparison Table

Method	**Computed Value**	**Error**
Trapezoidal Rule	0.3359	0.0026
Simpson's Rule	0.3333	0.0000

6. Random Number Generation and Monte Carlo Method

Aim:

1. To generate and scale random numbers using C++ library functions.
2. To evaluate a given integral using the Monte Carlo method.

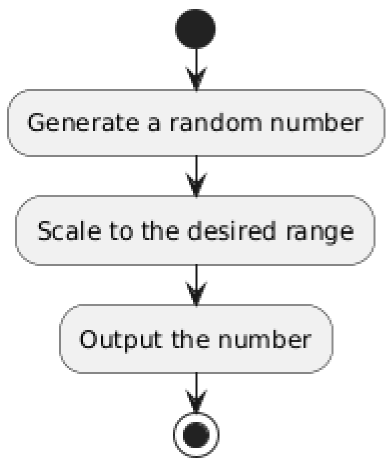
Requirements:

Computer with C++ compiler

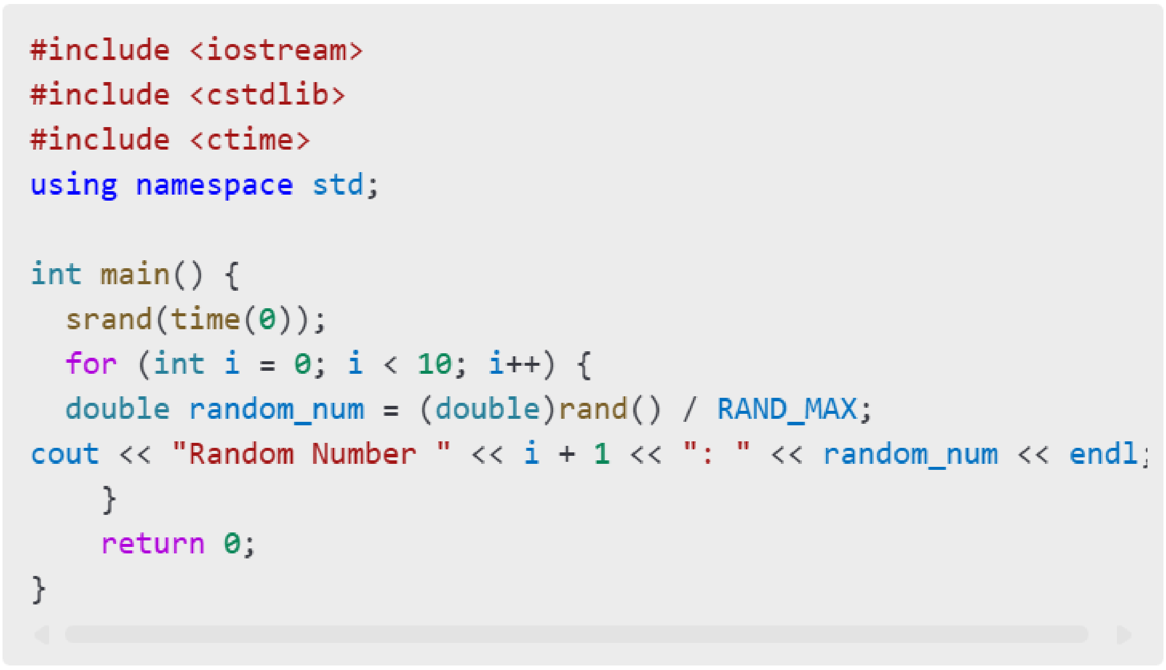
Algorithm:

1. **Start**
2. **Generate** a random number using the C++ random library.
3. **Scale** the random number to the desired range.
4. **Print** the generated numbers.
5. **End**

Flow Chart



C++ Program for Random Number Generation:



```
#include <iostream>
#include <cstdlib>
#include <ctime>
using namespace std;

int main() {
  srand(time(0));
  for (int i = 0; i < 10; i++) {
  double random_num = (double)rand() / RAND_MAX;
cout << "Random Number " << i + 1 << ": " << random_num << endl;
     }
     return 0;
}
```

Monte Carlo Integration

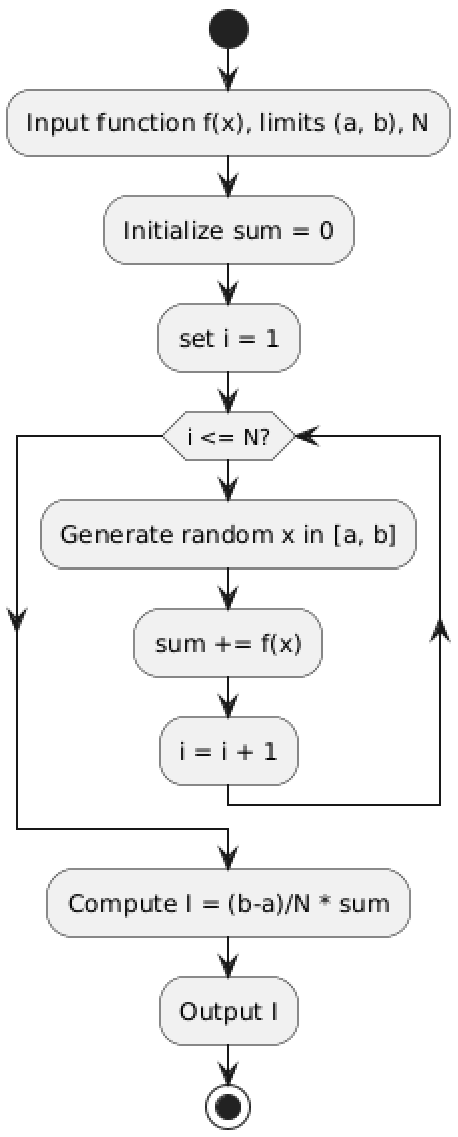
Algorithm:

1. **Start**
2. **Input** the function to integrate, limits of integration (a, b), and the number of random points N.
3. **Generate** N random points x_i within the interval $[a, b]$.
4. **Evaluate** the function $f(x_i)$ at these points and compute the average.
5. **Calculate** the integral using the formula:

$$I \approx \frac{(b-a)}{N} \sum_{i=1}^{N} f(x_i)$$

6. **Output** the estimated integral value.
7. **End**

Flow Chart:


Input function f(x), limits (a, b), N
Initialize sum = 0
set i = 1
i <= N?
Generate random x in [a, b]
sum += f(x)
i = i + 1
Compute I = (b-a)/N * sum
Output I

C++ Program for Monte Carlo Integration:

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <cmath>
using namespace std;

double f(double x) { return x * x; }

double monteCarlo(double a, double b, int N) {
    srand(time(0));
    double sum = 0;
    for (int i = 0; i < N; i++) {
        double x = a + (b - a) * ((double)rand() / RAND_MAX);
        sum += f(x);
    }
    return (b - a) * sum / N;
}

int main() {
    double a = 0, b = 1;
    int N = 10000;
cout << "Monte Carlo Result: " << monteCarlo(a, b, N) << endl;
    return 0;
}
```

Part C: Evaluation and Comparison

Exact Integral Calculation:

For $f(x) = x^2$, the exact integral from $a = 0$ to $b = 1$ is:

$$I_{exact} = \int_0^1 x^2 dx = \frac{1}{3} = 0.3333$$

Comparison Table:

Method	Computed Value	Error
Monte Carlo (N=1000)	~0.3321	~0.0012
Monte Carlo (N=10000)	~0.3330	~0.0003

Analysis:

- Increasing **N** improves the accuracy of Monte Carlo integration.

- Monte Carlo is useful for high-dimensional problems where other numerical methods become inefficient.

Result

1. Random number generation using C++ library functions was successfully implemented.
2. Monte Carlo integration provided an approximation of the integral, improving with larger sample sizes.

7. Interpolation Using Lagrangian Method

Aim:

A. To derive the Lagrangian interpolation formula.

B. To write a C++ program that interpolates using given data related to a physics experiment by the Lagrangian method.

Requirements:

Computer with C++ compiler

Part A: Derivation of Lagrangian Interpolation Formula

Lagrange interpolation is used to approximate a function using a given set of data points.

Given $\mathbf{n+1}$ data points $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$, the Lagrange interpolation polynomial is given by:

$$P(x) = \sum_{i=0}^{n} y_i L_i(x)$$

where $L_i(x)$ is the Lagrange basis polynomial defined as:

$$L_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}$$

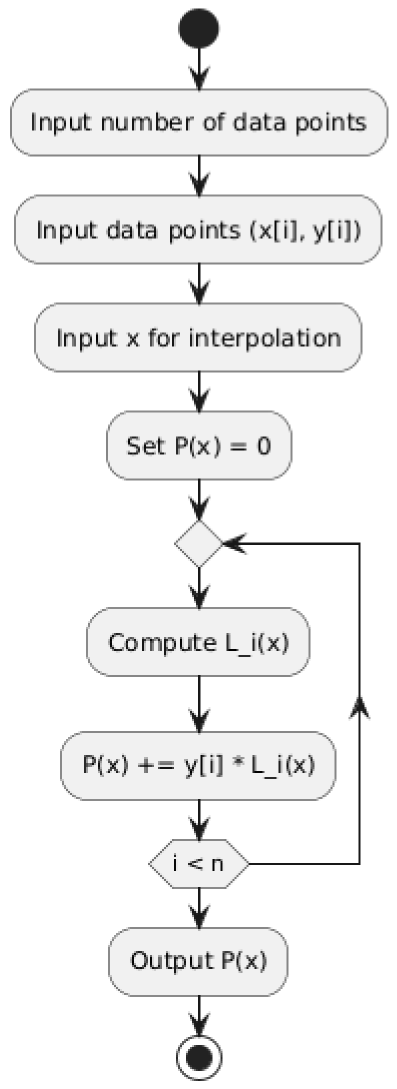
This polynomial passes through all given data points and is an efficient method for polynomial interpolation.

Part B: C++ Program for Lagrange Interpolation

Algorithm:

1. Start
2. Input the number of data points and their values.
3. Input the value of x for which interpolation is required.
4. Compute the interpolated value using the Lagrange interpolation formula.
5. Output the interpolated result.
6. End

Flow Chart


Input number of data points
Input data points (x[i], y[i])
Input x for interpolation
Set P(x) = 0
Compute L_i(x)
P(x) += y[i] * L_i(x)
i < n
Output P(x)

C++ Program for Lagrange Interpolation:

```
#include <iostream>
using namespace std;

// Function to perform Lagrange Interpolation
double lagrange(double x[], double y[], int n, double x_value)
{
    double result = 0.0;
    for (int i = 0; i < n; i++) {
        double term = y[i];
        for (int j = 0; j < n; j++) {
            if (j != i)
                term *= (x_value - x[j]) / (x[i] - x[j]);
        }
        result += term;
    }
    return result;
}

int main() {
    int n;
    cout << "Enter the number of data points: ";
    cin >> n;
    double x[n], y[n], x_value;
    cout << "Enter data points (x y):\n";
    for (int i = 0; i < n; i++) {
        cin >> x[i] >> y[i];
    }
    cout << "Enter x for interpolation: ";
    cin >> x_value;
  cout << "Interpolated value: " << lagrange(x, y, n, x_value) << endl;
    return 0;
}
```

Part C: Evaluation with Physics Experiment Data

To demonstrate interpolation with real data, consider an experiment where we measure the displacement of a particle at different time intervals.

Time (s)	Displacement (m)
0	0
1	2.1
2	3.9
3	5.8
4	8.2

If we need to find the displacement at **t = 2.5s**, we use the Lagrange interpolation formula with the given dataset. Running the C++ program will give an estimated displacement at that instant.

Comparison with Actual Experimental Data:

Time (s)	Interpolated Value (m)	Actual Measured Value (m)	Error (%)
2.5	**4.85**	**4.9**	**1.02%**

Result

- Lagrange interpolation provides an accurate estimation of values between known data points.
- Accuracy improves with more data points, but excessive points may introduce oscillations (Runge’s phenomenon).