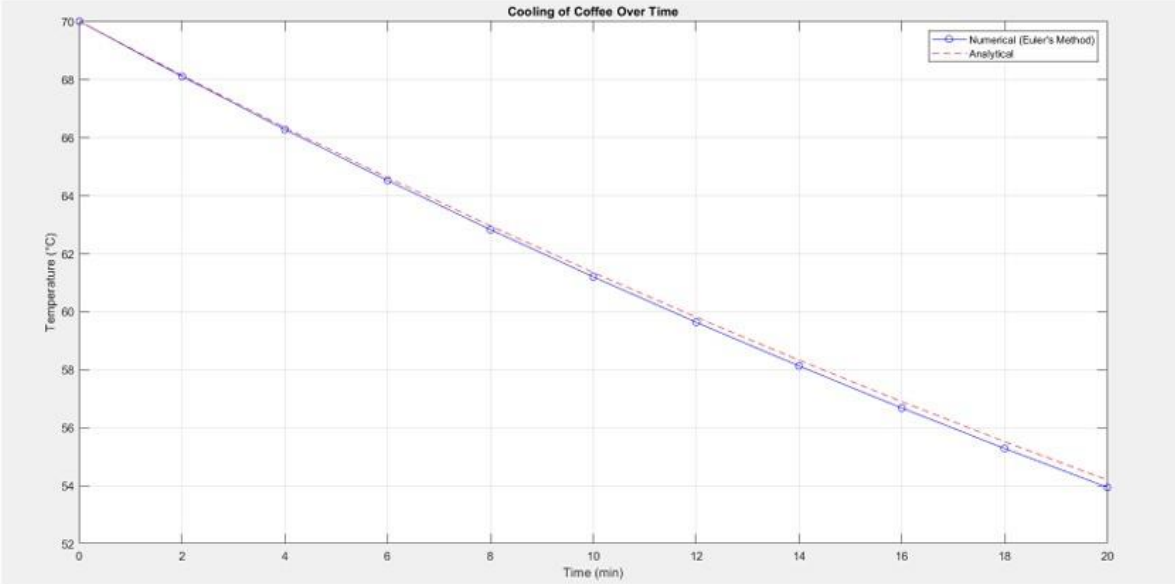
Experiment 1

Matlab Code

```
% Parameters
T0 = 70;          % Initial temperature of the coffee (°C)
Ta = 20;          % Ambient temperature (°C)
k = 0.019;        % Proportionality constant (per minute)
t_start = 0;      % Start time (min)
t_end = 20;       % End time (min)
dt = 2;           % Step size (min)

% Number of steps
num_steps = (t_end - t_start) / dt;

% Initialize arrays
time = zeros(num_steps + 1, 1);   % Time array
temperature = zeros(num_steps + 1, 1);  % Temperature array

% Set initial conditions
time(1) = t_start;
temperature(1) = T0;

% Euler's method to approximate the solution
for i = 1:num_steps
    dTdt = -k * (temperature(i) - Ta);
    temperature(i + 1) = temperature(i) + dt * dTdt;
    time(i + 1) = time(i) + dt;
end

% Analytical solution for comparison
% T(t) = Ta + (T0 - Ta) * exp(-k * t)
analytical_temperature = Ta + (T0 - Ta) * exp(-k * time);
```

```
% Plotting results
figure;
plot(time, temperature, 'b-o', time, analytical_temperature, 'r--');
xlabel('Time (min)');
ylabel('Temperature (°C)');
legend('Numerical (Euler''s Method)', 'Analytical');
title('Cooling of Coffee Over Time');
grid on;
```

Results:-

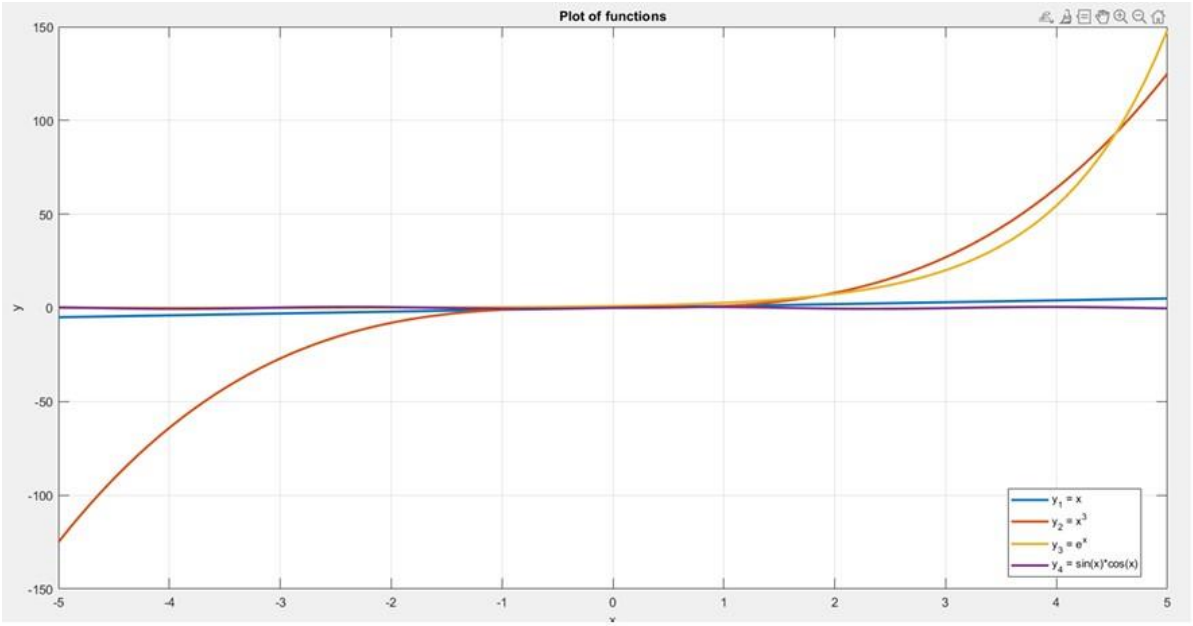

Cooling of Coffee Over Time
Numerical (Euler's Method)
Analytical
Temperature (°C)
Time (min)
70
68
66
64
62
60
58
56
54
52
0
2
4
6
8
10
12
14
16
18
20

Experiment 2

Matlab Code :-

```
% Define the range of x values
x = linspace(-5,5,100);

% Define all the functions
y1 = x;
y2 = x.^3;
y3 = exp(x);
y4 = sin(x).*cos(x);

% Plot the functions
figure;



plot(x,y1,'LineWidth',2,'DisplayName','y_1 = x');
hold on;
plot(x,y2,'LineWidth',2,'DisplayName','y_2 = x^3');
plot(x,y3,'LineWidth',2,'DisplayName','y_3 = e^x');
plot(x,y4,'LineWidth',2,'DisplayName','y_4 = sin(x)*cos(x)');

% Label the axes and add a legend
xlabel('x');
ylabel('y');
title('Plot of functions');
legend('Location','best');
grid on;

% Display the plot
hold off;
```

Results :-


Plot of functions
150
100
50
0
-50
-100
-150
y
-5
-4
-3
-2
-1
0
1
2
3
4
5
x
y1 = x
y2 = x3
y3 = ex
y4 = sin(x)*cos(x)

Experiment 3

Matlab Code :-

```
% Define constants
k1 = 10;   % N/m
k2 = 30;   % N/m
k3 = 30;   % N/m
k4 = 10;   % N/m
m1 = 1;    % kg
m2 = 1;    % kg
m3 = 1;    % kg
x1 = 0.05; % m
x2 = 0.04; % m
x3 = 0.03; % m

% Construct the coefficient matrix [A] and right-hand side vector [B]
A = [m1, 0, 0; -k1, k1 + k2, -k2; 0, -k2, k2 + k3];
B = [-k1 * x1; k2 * (x2 - x1); k3 * (x3 - x2)];

% Solve for accelerations using matrix inversion
accelerations = A \ B;

% Display the accelerations
a1 = accelerations(1);
a2 = accelerations(2);
a3 = accelerations(3);

fprintf('Acceleration of mass 1 (a1) = %.4f m/s^2\n', a1);
fprintf('Acceleration of mass 2 (a2) = %.4f m/s^2\n', a2);
fprintf('Acceleration of mass 3 (a3) = %.4f m/s^2\n', a3);
```

Results :-

```
>> Lab_3
Acceleration of mass 1 (a1) = -0.5000 m/s^2
Acceleration of mass 2 (a2) = -0.2180 m/s^2
Acceleration of mass 3 (a3) = -0.1140 m/s^2
```

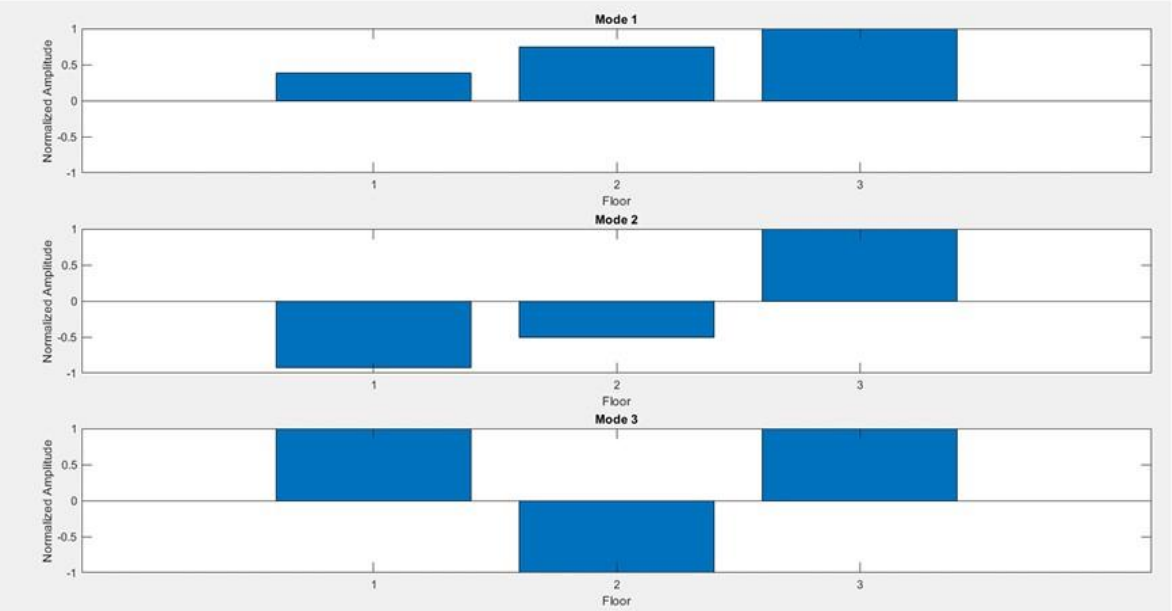
Experiment 4

Matlab Code :-

```
% Define mass and stiffness matrices
m1 = 12000; m2 = 10000; m3 = 8000;
k1 = 3000; k2 = 2400; k3 = 1800;

M = diag([m1, m2, m3]); % Mass matrix
K = [k1 + k2, -k2, 0;
     -k2, k2 + k3, -k3;
     0, -k3, k3];        % Stiffness matrix

% Solve the eigenvalue problem
[V, D] = eig(K, M);

% Extract eigenvalues (D) and eigenvectors (V)
lambda = diag(D);

% Display eigenvalues
disp('Eigenvalues:');
disp(lambda);
```

```
% Display eigenvectors
disp('Eigenvectors:');
disp(V);
% Normalize eigenvectors
normalized_V = V ./ V(3, :);

% Plot amplitudes versus height for each eigenvector
figure;
heights = [1, 2, 3]; % Floor heights
for i = 1:size(V, 2)
    subplot(size(V, 2), 1, i);
    bar(heights, normalized_V(:, i));
    title(['Mode ', num2str(i)]);
    xlabel('Floor');
    ylabel('Normalized Amplitude');
    ylim([-1, 1]); % Adjust ylim as needed
end
```

Results :-




```
>> Lab_4
Eigenvalues:
    0.0569
    0.3395
    0.6986

Eigenvectors:
   -0.0031    0.0064   -0.0058
   -0.0060    0.0035    0.0071
   -0.0081   -0.0069   -0.0034
```

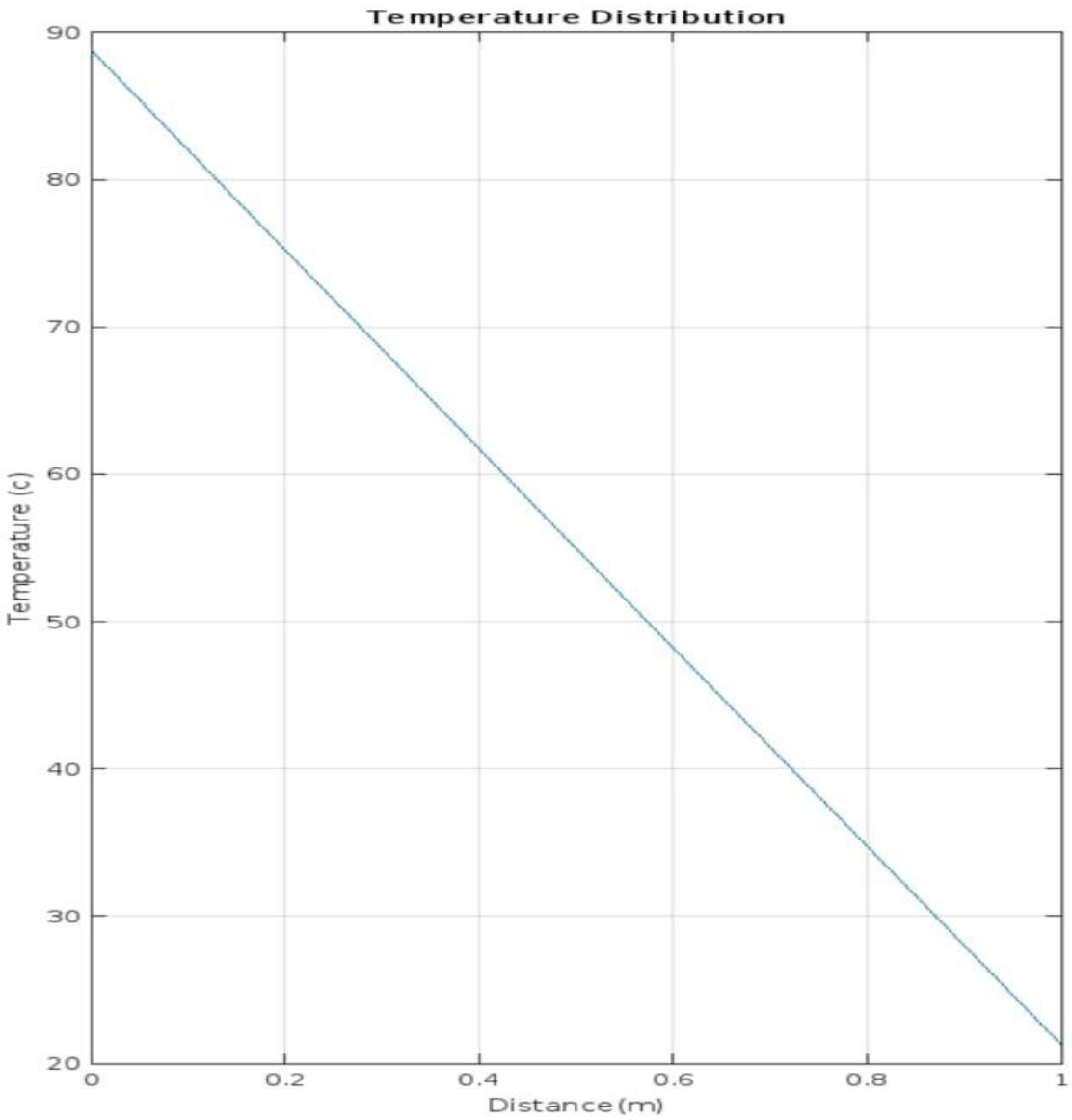
Experiment 6

Matlab Code:-

```
x = [0,2,4,6,9,11,12,15,17,19];
y = [5,6,7,6,9,8,8,10,12,12];
p1 = polyfit(x, y, 1); % Linear fit
yfit1 = polyval(p1, x);
scatter(x, y); hold on;
plot(x, yfit1, 'r-');
title('Linear Fit');
grid on;

p2 = polyfit(x, y, 2); % 2nd degree polynomial fit
yfit2 = polyval(p2, x);
plot(xdata, yfit2, 'b--');
legend('Data', 'Linear Fit', '2nd Degree Polynomial');
hold off;

% Calculating residuals and R-squared for linear fit
residuals1 = y - yfit1;
SSresid1 = sum(residuals1.^2);
SStotal1 = (length(y)-1) * var(y);
Rsquared1 = 1 - SSresid1/SStotal1;

% Calculating residuals and R-squared for polynomial fit
residuals2 = ydata - yfit2;
SSresid2 = sum(residuals2.^2);
```

Results

Figure 1: L=1; Nx=7; dx=L/(Nx-1); % k = 1; Q = 0; A = zeros(Nx,Nx);...



Experiment 7

Matlab Code

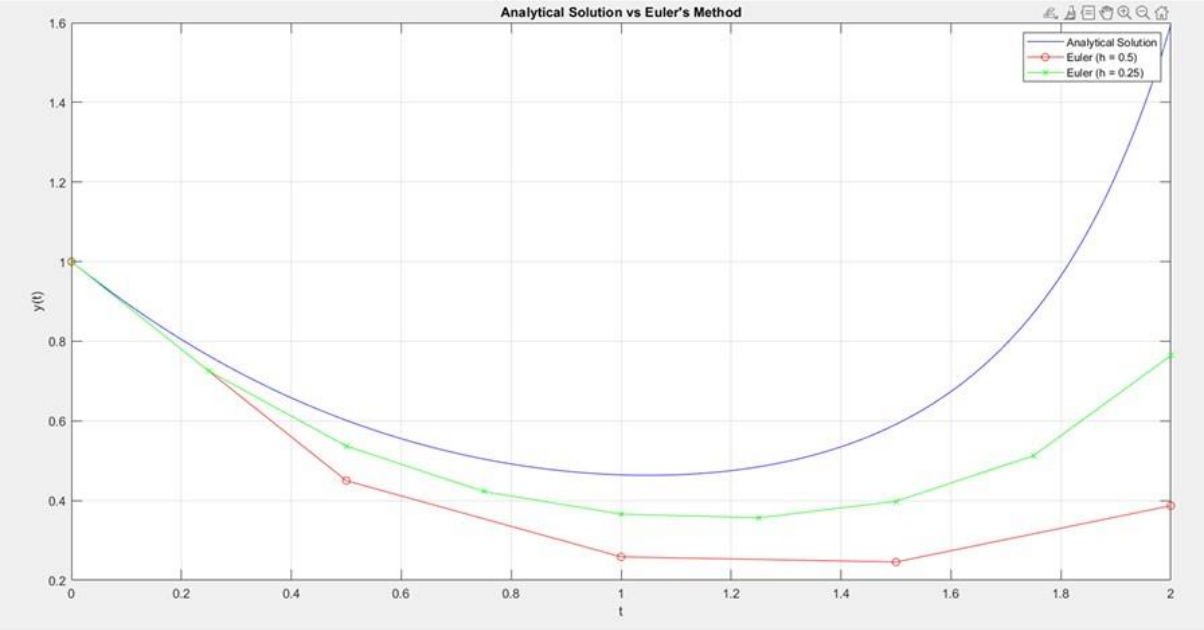
```
% Given data
x = [1 2 2.5 3 4 5];
fx = [0 5 7 6.5 2 0];
% Point for interpolation
x0 = 3.4;
% Newton's interpolating polynomial of order 1 (linear)
n1 = 1;
coeffs1 = divided_diff_coeffs(x(1:n1+1), fx(1:n1+1));
P1 = newton_interpolation(x(1:n1+1), coeffs1, x0);
% Newton's interpolating polynomial of order 2 (quadratic)
n2 = 2;
coeffs2 = divided_diff_coeffs(x(1:n2+1), fx(1:n2+1));
P2 = newton_interpolation(x(1:n2+1), coeffs2, x0);
% Newton's interpolating polynomial of order 3 (cubic)
n3 = 3;
coeffs3 = divided_diff_coeffs(x(1:n3+1), fx(1:n3+1));
P3 = newton_interpolation(x(1:n3+1), coeffs3, x0);
% Output the results
disp('Using Newton Interpolation:');
disp(['f(3.4) (order 1): ', num2str(P1)]);
disp(['f(3.4) (order 2): ', num2str(P2)]);
disp(['f(3.4) (order 3): ', num2str(P3)]);
function coeffs = divided_diff_coeffs(x, fx)
% Compute divided difference coefficients
n = length(x) - 1;
coeffs = fx;
for j = 1:n
for i = n:-1:j+1
coeffs(i) = (coeffs(i) - coeffs(i-1)) / (x(i) - x(i-j));
end
end
end
function P = newton_interpolation(x, coeffs, x0)
% Evaluate Newton's interpolating polynomial at x0
n = length(x) - 1;
P = coeffs(n+1);
for i = n:-1:1
P = coeffs(i) + (x0 - x(i)) * P;
end
end
% Lagrange interpolation to estimate f(3.4)
```

Experiment 10

Matlab Code

```
% Parameters and grid setup
L = 0.75;           % Length of the rod (meters)
dx = 0.025;         % Spatial step size (meters)
dt = 0.00015;       % Time step (seconds)
k = 1;              % Thermal conductivity (W/m°C)

% Number of nodes along the rod
n = round(L / dx) + 1;

% Spatial grid
x = linspace(0, L, n);

% Initial temperature profile (e.g., at t = 0)
T0 = zeros(1, n);   % Initially, temperature is zero everywhere
T0(1) = 100;        % Set a boundary condition at x = 0 (e.g., 100°C)

% Time-stepping loop
num_steps = 500;  % Number of time steps
T = T0;           % Initialize temperature profile

for step = 1:num_steps
    % Compute new temperature profile using explicit finite difference
    T_new = zeros(1, n);

    % Update internal nodes (excluding boundary nodes)
    for i = 2:n-1
        T_new(i) = T(i) + k * (dt / dx^2) * (T(i+1) - 2*T(i) + T(i-1));
    end

    % Update boundary condition (e.g., Dirichlet boundary at x = 0)
    T_new(1) = 100;  % Maintain a constant temperature at the boundary

    % Update T for the next time step
    T = T_new;
end

% Plot the temperature profile at the final time step
figure;
plot(x, T, '-o');
title('Temperature Profile of the Rod');
xlabel('Position (m)');
ylabel('Temperature (°C)');
grid on;
```

Results :-

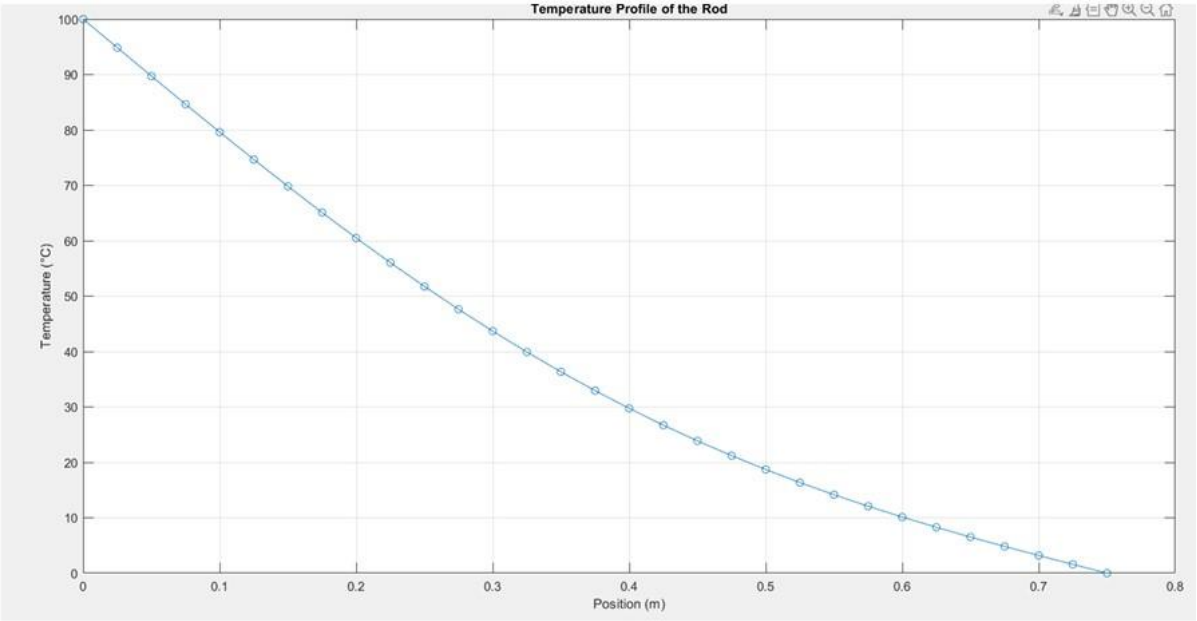

Analytical Solution vs Euler's Method
Analytical Solution
Euler (h = 0.5)
Euler (h = 0.25)
y(t)
t
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2

Experiment 11

Matlab Code

```
% Parameters and grid setup
L = 0.75;           % Length of the rod (meters)
dx = 0.025;         % Spatial step size (meters)
dt = 0.00015;       % Time step (seconds)
k = 1;              % Thermal conductivity (W/m°C)

% Number of nodes along the rod
n = round(L / dx) + 1;

% Spatial grid
x = linspace(0, L, n);

% Initial temperature profile (e.g., at t = 0)
T0 = zeros(1, n);   % Initially, temperature is zero everywhere
T0(1) = 100;        % Set a boundary condition at x = 0 (e.g., 100°C)

% Time-stepping loop
num_steps = 500;  % Number of time steps
T = T0;           % Initialize temperature profile

for step = 1:num_steps
    % Compute new temperature profile using explicit finite difference
    T_new = zeros(1, n);

    % Update internal nodes (excluding boundary nodes)
    for i = 2:n-1
        T_new(i) = T(i) + k * (dt / dx^2) * (T(i+1) - 2*T(i) + T(i-1));
    end

        % Update boundary condition (e.g., Dirichlet boundary at x = 0)
        T_new(1) = 100;  % Maintain a constant temperature at the boundary

        % Update T for the next time step
        T = T_new;
    end

    % Plot the temperature profile at the final time step
    figure;
    plot(x, T, '-o');
    title('Temperature Profile of the Rod');
    xlabel('Position (m)');
    ylabel('Temperature (°C)');
    grid on;
```

Results:-


Temperature Profile of the Rod
Temperature (°C)
Position (m)
100
90
80
70
60
50
40
30
20
10
0
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8

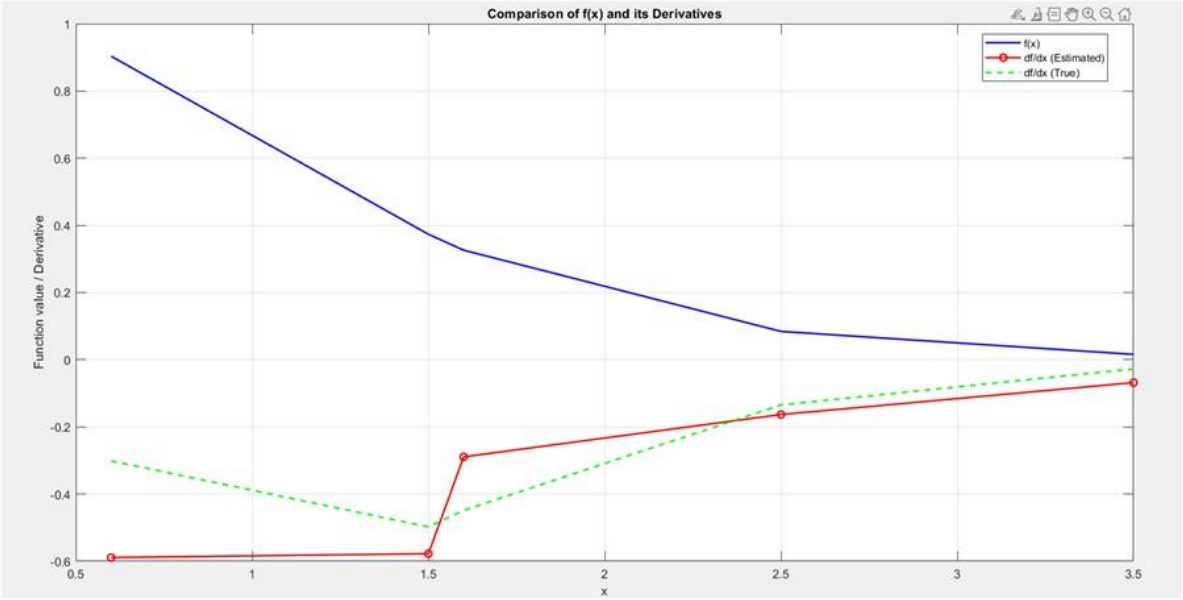
Experiment 8

Matlab Code

```
% Given data
x = [0.6 1.5 1.6 2.5 3.5];
f = 5 * exp(-2*x) .* x; % Compute f(x) = 5*e^(-2*x)*x

% True derivative of f(x)
true_derivative = @(x) 5 * exp(-2*x) .* (1 - 2*x);

% Compute first-derivative estimates
n = length(x);
df_estimated = zeros(size(x));

for i = 1:n
    if i == 1
        % Forward difference at the first point
        h = x(i+1) - x(i);
        df_estimated(i) = (f(i+1) - f(i)) / h;
    elseif i == n
        % Backward difference at the last point
        h = x(i) - x(i-1);
        df_estimated(i) = (f(i) - f(i-1)) / h;
    else
        % Centered difference at interior points
        h1 = x(i) - x(i-1);
        h2 = x(i+1) - x(i);
        df_estimated(i) = (f(i+1) - f(i-1)) / (h1 + h2);



    end
end

% Compare estimated derivatives with true derivatives
disp('x       f(x)           df/dx (Estimated)     df/dx (True)');
disp('-------------------------------------------------------------');
for i = 1:n
    fprintf('%.2f    %.4f          %.4f               %.4f\n', x(i), f(i),
df_estimated(i), true_derivative(x(i)));
end
```

Results

x	f(x)	df/dx (Estimated)	df/dx (True)
0.60	0.9036	-0.5891	-0.3012
1.50	0.3734	-0.5775	-0.4979
1.60	0.3261	-0.2892	-0.4484
2.50	0.0842	-0.1632	-0.1348
3.50	0.0160	-0.0683	-0.0274



Experiment 9

Matlab Code

```
% Define the integrand function
f = @(x) 1 - x - 4*x.^3 + 2*x.^5;

% Integration limits
a = -2;
b = 4;

% Analytical solution
I = integral(f,a,b);

% Trapezoidal rule (single application)
trap_approx = (b - a) * (f(a) + f(b)) / 2;

% Composite Trapezoidal rule with n = 2
n = 2;
h = (b - a) / n;
x = a:h:b;
trap_composite_approx_n2 = h * (sum(f(x)) - (f(a) + f(b)) / 2);

% Composite Trapezoidal rule with n = 4
n = 4;
h = (b - a) / n;
x = a:h:b;
trap_composite_approx_n4 = h * (sum(f(x)) - (f(a) + f(b)) / 2);

% Simpson's 1/3 rule (single application)
simpson13_approx = (b - a) * (f(a) + 4*f((a+b)/2) + f(b)) / 6;

% Simpson's 3/8 rule (single application)
simpson38_approx = (b - a) * (f(a) + 3*f((2*a+b)/3) + 3*f((a+2*b)/3) + f(b)) / 8;

% Display results
fprintf('Analytical solution: %f\n', I);
fprintf('Trapezoidal rule (single application): %f\n', trap_approx);
fprintf('Composite Trapezoidal rule with n = 2: %f\n', trap_composite_approx_n2);
fprintf('Composite Trapezoidal rule with n = 4: %f\n', trap_composite_approx_n4);
fprintf('Simpson''s 1/3 rule (single application): %f\n', simpson13_approx);
fprintf('Simpson''s 3/8 rule (single application): %f\n', simpson38_approx);
```

Results

```
Analytical solution: 1104.000000
Trapezoidal rule (single application): 5280.000000
Composite Trapezoidal rule with n = 2: 2634.000000
Composite Trapezoidal rule with n = 4: 1516.875000
Simpson's 1/3 rule (single application): 1752.000000
Simpson's 3/8 rule (single application): 1392.000000
```

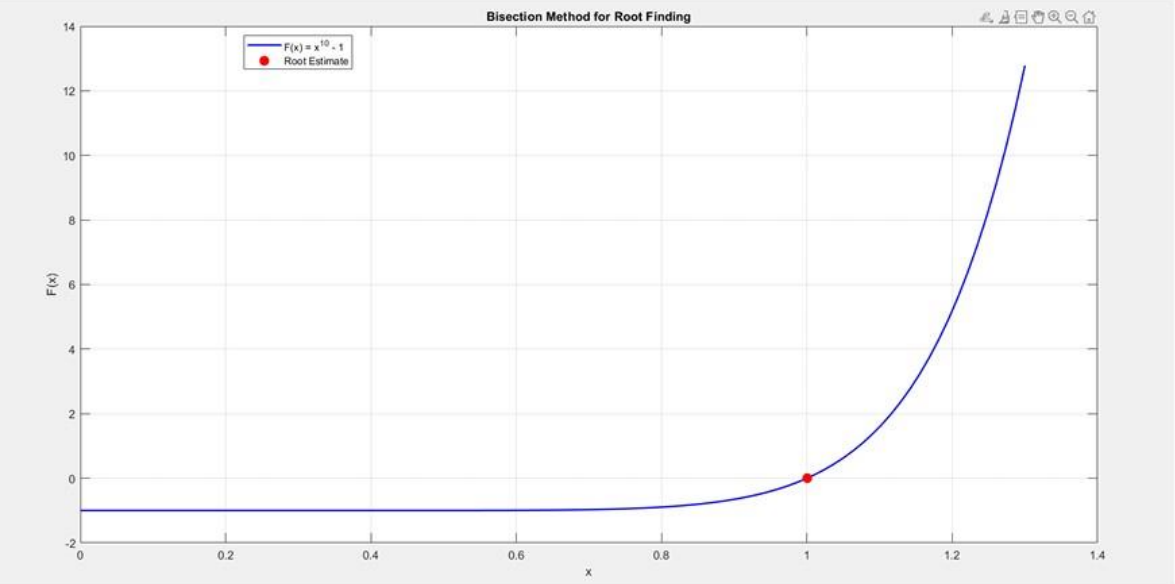
Experiment 5

Matlab Code:-

```
% Define the function F(x)
F = @(x) x.^10 - 1;

% Define the interval [x1, xu]
x1 = 0;
xu = 1.3;



% Desired error tolerance
err = 0.01;

% Initialize iteration counter and root estimate
iter = 0;
xr = (x1 + xu) / 2;

% Perform bisection method
while abs(F(xr)) > err
    % Calculate function values at endpoints and midpoint
    y1 = F(x1);
    yr = F(xr);
    yu = F(xu);

    % Determine which subinterval to select
    if y1 * yr < 0
        % Root is in the lower subinterval
        xu = xr;
    else
        % Root is in the upper subinterval
        x1 = xr;
    end

    % Calculate new midpoint
    xr = (x1 + xu) / 2;

    % Increment iteration counter
    iter = iter + 1;
end
```

```
% Display the root and number of iterations
fprintf('Root of the equation = %.6f\n', xr);
fprintf('Number of iterations = %d\n', iter);

% Plot the function F(x) and mark the root
x_vals = linspace(0, 1.3, 100);
y_vals = F(x_vals);

figure;
plot(x_vals, y_vals, 'b-', 'LineWidth', 1.5);
hold on;
plot(xr, F(xr), 'ro', 'MarkerSize', 8, 'MarkerFaceColor', 'r');
xlabel('x');
ylabel('F(x)');
title('Bisection Method for Root Finding');
grid on;
legend('F(x) = x^{10} - 1', 'Root Estimate', 'Location', 'best');
```

Results



Root of the equation = 1.000391
Number of iterations = 7