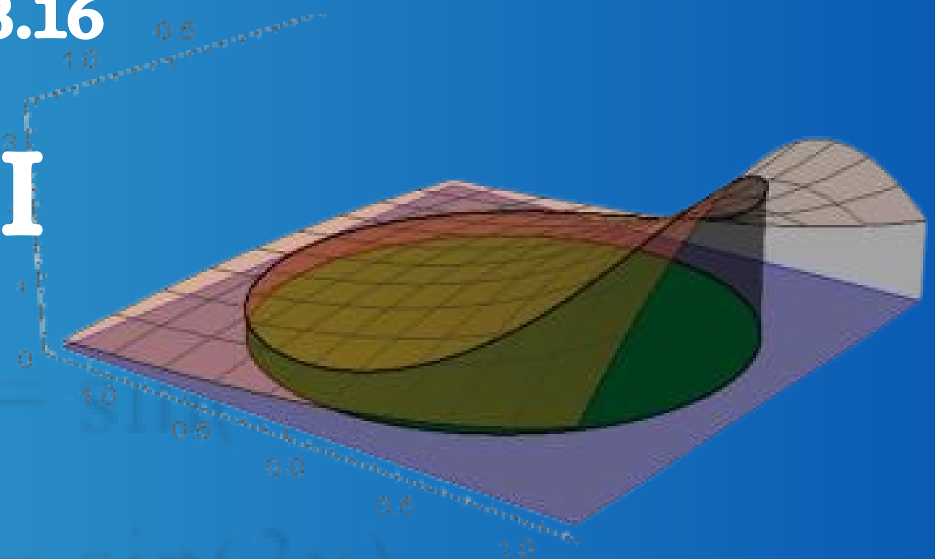
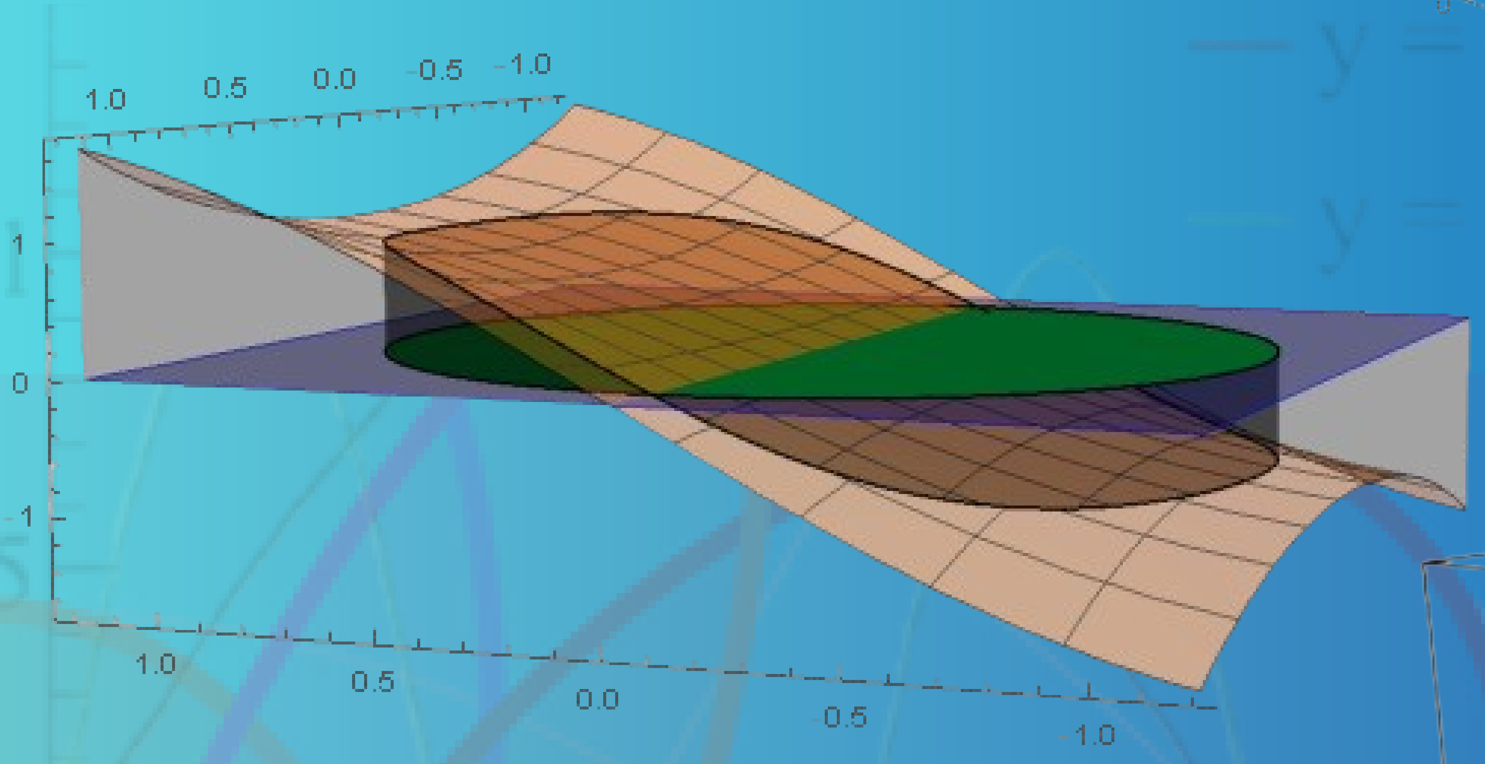
DEVELOPMENT OF MATHEMATICS PRACTICAL LABORATORY MANUAL AS PER THE ITEP SYLLABUS

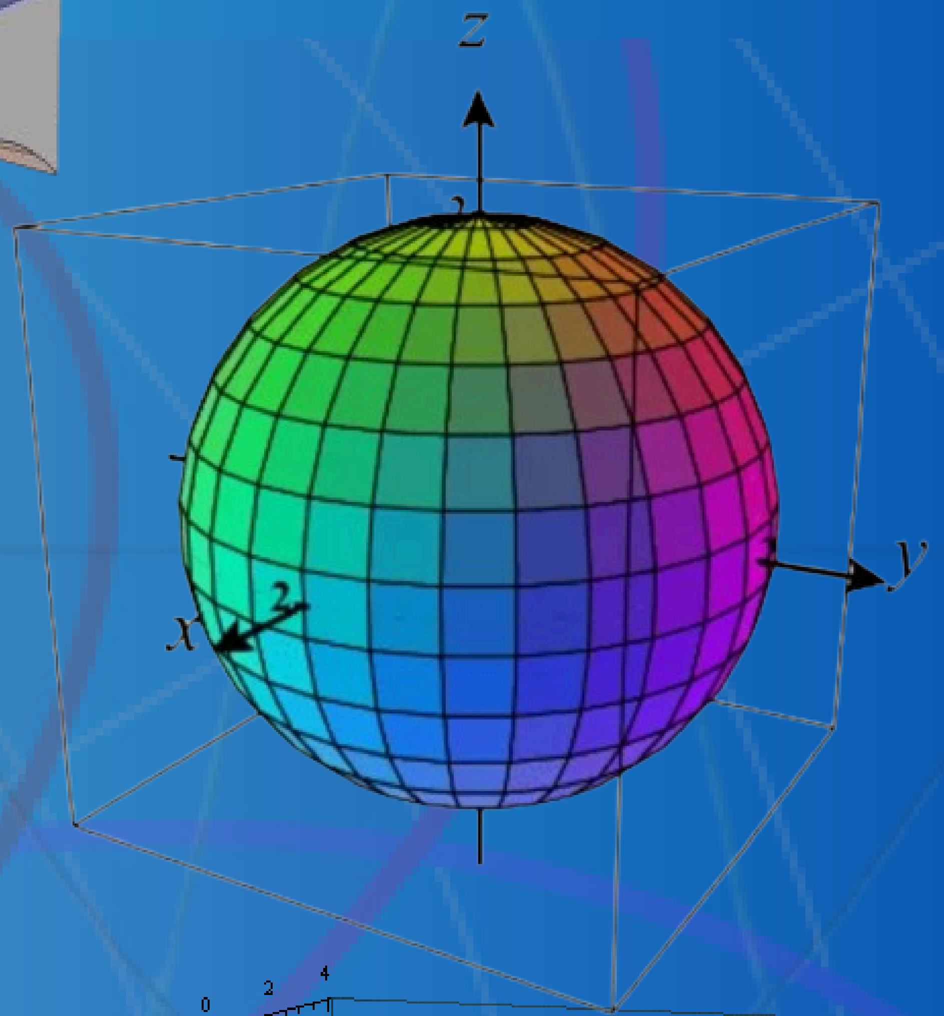
2024-25

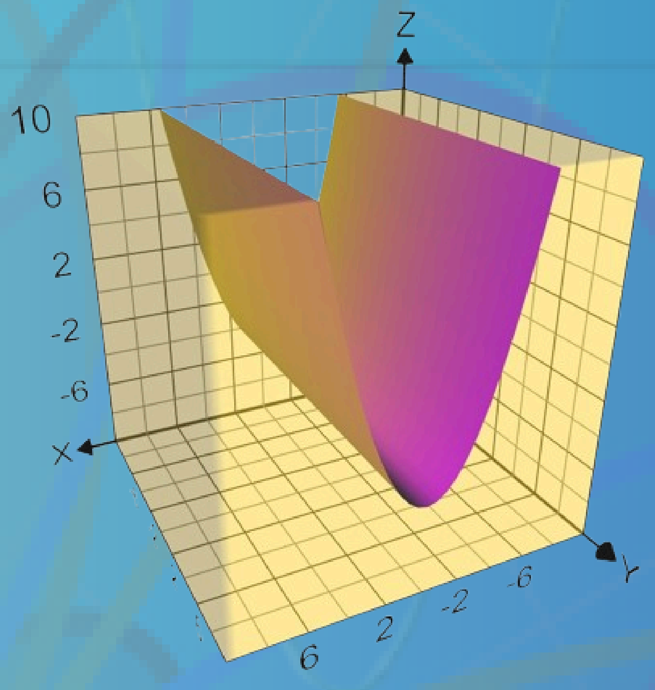
PAC Code 23.16

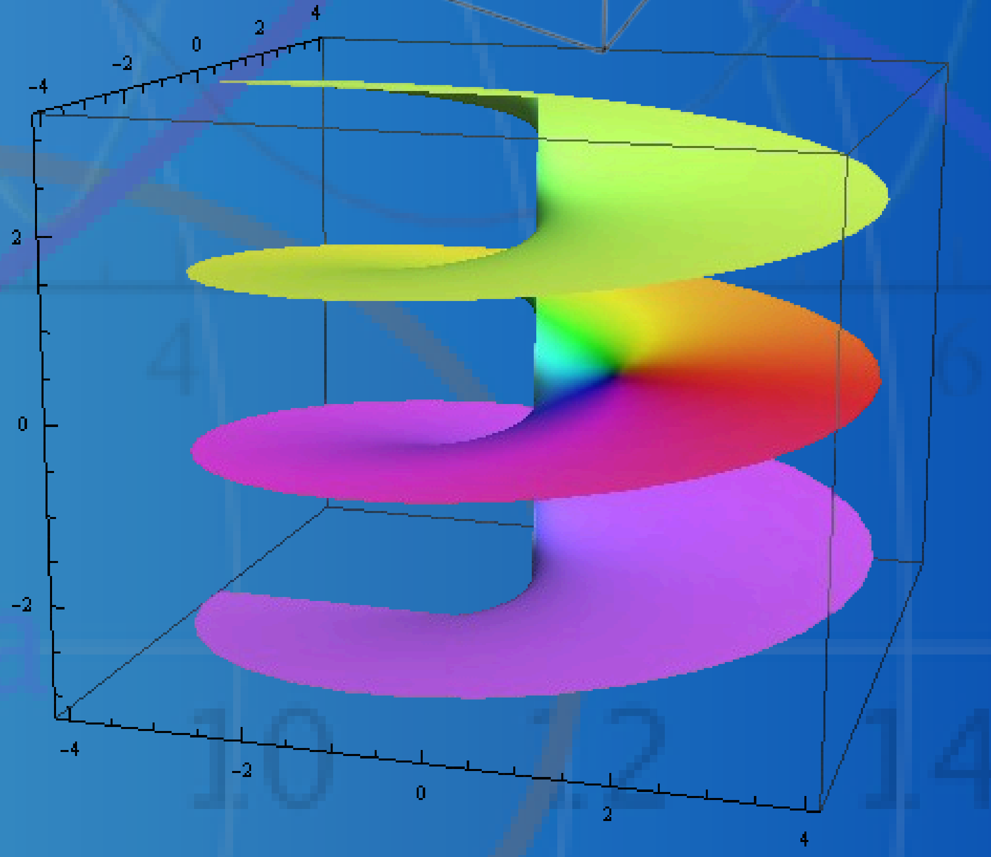
Part - II

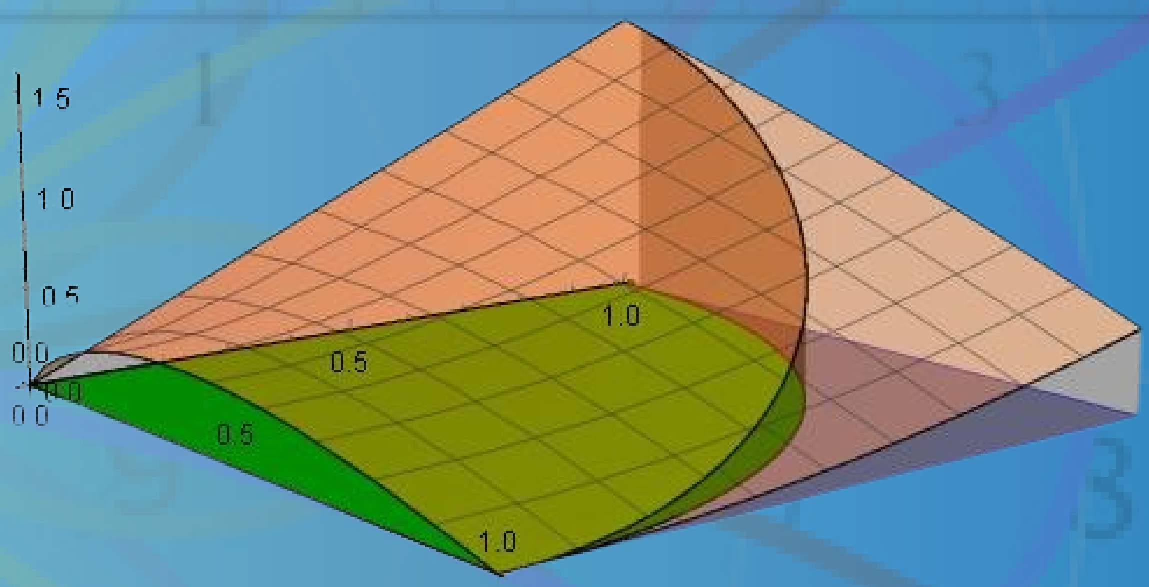












Dr. Ashwani Kumar Garg
Mr. Aji Thomas
Dr. Rajat Kaushik

REGIONAL INSTITUTE OF EDUCATION
National Council of Educational Research and Training
Shyamala Hills, Bhopal, 102002

DEVELOPMENT OF MATHEMATICS PRACTICAL LABORATORY MANUAL AS PER THE ITEP SYLLABUS

PART-II

विद्यया ऽमृतमश्नुते

एन सी ई आर टी
NCERT

Dr. Ashwani Kumar Garg
Associate Professor, Mathematics, RIE, NCERT, Bhopal

Mr. Aji Thomas
Assistant Professor, Mathematics, RIE, NCERT, Bhopal

Dr. Rajat Kaushik
Assistant Professor, Mathematics, RIE, NCERT, Bhopal

REGIONAL INSTITUTE OF EDUCATION, BHOPAL
National Council of Educational Research and Training

Manual Development Committee

Advisory Committee

1. Prof. Dinesh Prasad Saklani, Director, NCERT
2. Prof. Jaydip Mandal, Principal, RIE Bhopal
3. Prof. Sunita Farkiya, Head, Department of Education in Science and Mathematics, NIE, NCERT, New Delhi
4. Prof. Chitra Singh, Head, Department of Extension Education, RIE Bhopal
5. Prof. Rashmi Singhai, Head, Department of Education in Science and Mathematics, RIE Bhopal

Manual Development Team

1. Prof. Saurabh Kumar Shrivastava, Deptt. of Mathematics, IISER Bhopal
2. Prof. K.B. Subramaniam, Retired faculty, RIE , NCERT, Bhopal
3. Dr. Kameshwar Rao, Retired faculty, RIE ,NCERT, Bhopal
4. Dr. Sourav Das, Assistant Professor, NIT Jamshedpur
5. Dr. Kuldeep Singh Yadav, Assistant Professor in Mathematics, MANIT Bhopal
6. Dr. Subhas Khajanchi, Assistant Professor, Dept. of Mathematics, Presidency University Kolkata
7. Dr. Pushpendra Kumar, Assistant Professor Mathematics, MANIT Bhopal
8. Dr. Saurabh Kapoor, Assistant Professor, RIE, NCERT, Bhubaneswar
9. Dr. Birendra Singh, Assistant Professor, AIMT, Lucknow
10. Dr. Shimpi Singh Jadon, Assistant Professor, Rajkiya Engineering College, Kannauj, U.P.
11. Dr. Anurag Shukla, Assistant Professor, Rajkiya Engineering College Kannauj, U.P.

12. Dr. Dheerendra Mishra, Assistant Professor in Mathematics MANIT, Bhopal
13. Dr. Mohit Kumar, Assistant Professor, GLA University, Mathura
14. Dr. Anil Kumar Shukla, Assistant Professor of Mathematics, VIT Bhopal University Sehore, M.P.
15. Mr. Jay Kishore Sahani, Assistant Professor in Mathematics, Department of Mathematics, D.A.V. PG College, Siwan
16. Dr. Rishikesh Kumar, Assistant Professor, NIE, NCERT, New Delhi
17. Dr. Namrata Tripathi, Assistant professor, Govt. College, Phanda, Bhopal
18. Mr. Abhinav Gupta, Research Scholar in Mathematics, MANIT Bhopal
19. Mr. Sanjeev Kumar, Research Scholar, AMU, Aligarh
20. Mr. Pawan Kumar, Research Scholar, AMU
21. Mr. Gurmeet Saini, Research Scholar, Rajkiya Engineering College Kannauj, U.P.
22. Dr. Anjali, Post doc fellow, IIT Bombay
23. Mr. Yogesh Trivedi, Research Scholar, BITS, Goa
24. Ms. Priya Srivastava, Research Scholar, MANIT Prayagraj
25. Mr. Nitish Kumar Mahala, Research Scholar in Mathematics, MANIT, Bhopal
26. Ms. Bhavana Singh, Research Scholar in Mathematics, MANIT Bhopal
27. Ms. Sakshi Raghuwanshi, Dept. of Mathematics, JPF, RIE, Bhopal
28. Dr. Rajat Kaushik, Assistant Professor in Mathematics, RIE, NCERT, Bhopal
29. Mr. Aji Thomas, Assistant Professor in Mathematics, RIE, NCERT, Bhopal
30. Dr. Ashwani Kumar Garg, Associate Professor in Mathematics, RIE, NCERT, Bhopal

Preface

The *Development of Mathematics Practical Laboratory Manual as per the ITEP Syllabus* is designed to bridge the gap between theoretical mathematics and its practical applications through computational tools for easy handling the manual is divided into two parts (Part I and Part II). With rapid advancements in computers and mathematical software, technology has become an integral part of mathematical education. While traditional methods relied heavily on manual calculations, the present era demands that students develop computational and analytical skills to engage effectively with mathematical concepts.
Visualization plays a crucial role in mathematics learning. Modern mathematical software allows students to perform complex calculations quickly and provides graphical representations that enhance their understanding of abstract concepts. By investing time in learning the basic syntax of these tools, students can significantly improve their efficiency in problem-solving and analysis. This manual serves as a supplementary text for Practical Mathematics in the Integrated Teacher Education Programme (ITEP), offering a structured learning experience that enables students to analyze and classify mathematical concepts independently. The manual presents a sequence of examples and exercises that reinforce understanding and enhance problem-solving skills.
The primary objective of this manual is to acquaint students with various mathematical software tools, develop an understanding of the practical applications of mathematics, and enable them to integrate mathematics with other scientific disciplines. It aims to enhance analytical and problem-solving skills while preparing future educators to incorporate computational tools in teaching and research. Each chapter of this manual follows a structured approach, including conceptual explanations, recommended software applications, step-by-step solutions, and exercises to reinforce learning. The systematic organization of content ensures that students not only gain theoretical knowledge but also develop hands-on experience in applying computational tools to solve mathematical problems.
As computational mathematics continues to play a crucial role in education and research, this manual serves as an essential resource for students, bridging the gap between theoretical learning and real-world applications. The increasing relevance of artificial intelligence, machine learning, and data science has further highlighted the importance of computational

skills in problem-solving. This manual prepares students for the future by providing them with the expertise to navigate mathematical challenges using modern technological tools. Designed to be a valuable academic resource, this manual supports students through progressive learning, hands-on practice, and analytical problem-solving. By the end of this manual, students will have acquired a solid foundation in computational mathematics, proficiency in mathematical software, and the ability to apply mathematics to practical situations. This first part of the module introduces fundamental concepts, while the subsequent parts explore advanced problem-solving techniques and visualization tools. Through continuous engagement and exploration, students will develop the ability to analyze, interpret, and solve mathematical problems with confidence and efficiency. It is hoped that this manual will inspire learners to explore mathematics beyond traditional boundaries and embrace technology as a powerful tool for mathematical discovery.

Place: RIE, Bhopal **Program Coordinators**
Date: 31.01.2025

Acknowledgement

At the outset, we extend our deepest appreciation and gratitude to Prof. Dinesh Prasad Saklani, Director, NCERT, for his unwavering trust in us, his invaluable guidance, and his constant encouragement. His support has been instrumental in the successful development of the *Mathematics Practical Laboratory Manual as per the ITEP syllabus.*
We express our profound thanks to Prof. Jaydip Mandal, Principal, RIE Bhopal, for his visionary leadership, continuous motivation, and steadfast involvement in the conception and realization of this laboratory manual. We also extend our sincere gratitude to the esteemed members of the Program Advisory Committee for their approval and support in sanctioning this project.
We take this opportunity to place on record our sincere appreciation for Prof. Sunita Farkiya, Head, Department of Education in Science and Mathematics, NIE, NCERT, New Delhi, Prof. Rashmi Singhai, Head, Department of Education in Science and Mathematics, RIE Bhopal, and Prof. Chitra Singh, Head, Department of Extension Education, RIE Bhopal, for their invaluable contributions and insightful inputs throughout the development process. Our heartfelt thanks also go to the Principals of RIEs, the Mathematics faculty of various constituent units of NCERT, and the faculty of RIE Bhopal for their continued academic support.
We are deeply grateful to all the distinguished resource persons who played a crucial role in the development of this manual. Their expertise and dedication have significantly enriched this work. Our appreciation also extends to the Junior Project Fellows (JPFs) who contributed to various aspects of this project with diligence and enthusiasm. We sincerely acknowledge the efforts of the administrative staff, whose commitment and efficiency have ensured the smooth execution of this endeavur.
Lastly, we extend our gratitude to all individuals and institutions who, directly or indirectly, contributed to the successful completion of this program. Your invaluable support and collaboration have played a crucial role in making this initiative a reality.

Place: RIE, Bhopal **Program Coordinators**
Date: 31.01.2025

Contents

Chapter 1

Operation Research

Practical No. 1

Aim

To plot a graph of linear programming problem using MATLAB.

Problem

Plot the feasible region of the following LPP

$$\text{Max } Z = 3x_1 + 2x_2$$

$$\begin{aligned} \text{Subject to:} \quad & x_1 + x_2 \leq 4 \\ & x_1 + 2x_2 \leq 6 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Algorithm

Step 1: Initialize the value of $x1$ and $x2$ using the "linspace" command i.e.
$x1$=linspace$(0, 6, 5)$; $x2$=linspace$(0, 6, 5)$;
Step 2: Plot both the constraints using the "plot" command
plot $(x1, 4 - x1$, "g", "DisplayName", "$x1 + x2 <= 4$") ;
plot($x1, \frac{(6-x1)}{2}$, "b", "DisplayName", "$x1 + 2 \times x2 <= 6$");
Step 3: Set the limit of axes through the command xlim and ylim i.e.
xlim($[0, 6]$); ylim($[0, 6]$);
Step 4: Shading the feasible region with the help of "fill" command
fill($[0, 4, 2, 0], [0, 0, 2, 3]$, "$g$", "FaceAlpha",1, "DisplayName", "Feasible Region");
Step 5: Plot the vertices
plot($[0, 4, 2, 0], [0, 0, 2, 3]$, "$ro$", "DisplayName", "Vertices");

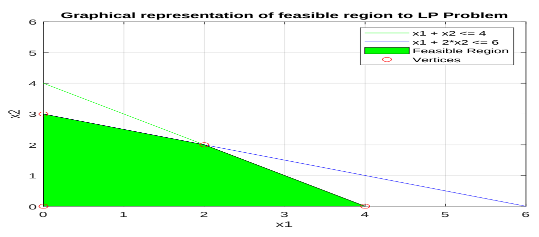
Step 6: Labeling the $x1$ and $x2$ axes
xlabel(“$x1$”); ylabel(“$x2$”);

Program

```
x1 = linspace(0, 6, 5); x2 = linspace(0, 6, 5);
plot(x1, 4 - x1, ‘g’, ‘DisplayName’, ‘x1 + x2 <= 4’);
hold on;
plot(x1, (6 - x1)/2, ‘b’, ‘DisplayName’, ‘x1 + 2*x2 <= 6’);
xlim([0, 6]);ylim([0, 6]);
fill([0, 4, 2, 0], [0, 0, 2, 3], ‘g’, ‘FaceAlpha’, 1,
‘DisplayName’, ‘Feasible Region’);
plot([0, 4, 2, 0], [0, 0, 2, 3], ‘ro’, ‘DisplayName’, ‘Vertices’);
xlabel(‘x1’);ylabel(‘x2’);
legend(‘show’);
title(‘Graphical representation of feasible region to LP Problem’);
grid on;
hold off;
```

Output

The graphical representation of the feasible region to given LPP is:



Exercise Problem

Plot the feasible region of the following LPP

$$\text{Max } Z = 6x_1 + 4x_2$$

$$\begin{aligned} \text{Subject to:} \quad & 2x_1 + 3x_2 \leq 120 \\ & 2x_1 + x_2 \leq 60 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Practical No. 2

Aim

To solve a linear programming problem using MATLAB.

Problem

Solve the following LPP

$$\text{Max } Z = 3x_1 + 2x_2$$

$$\begin{aligned} \text{Subject to:} \quad & x_1 + x_2 \leq 4 \\ & x_1 + 2x_2 \leq 6 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Algorithm

```
Step 1: Enter the coefficients of objective function and constraints
        $f = [-3, -2];$
        $A = [1, 1; 1, 2];$
        $b = [4; 6];$
        $lb = [0, 0];$
Step 2: To solve the LPP using linprog command
        $[x, Z] = \text{linprog}(f, A, b, [], [], lb);$
Step 3: Display the results using fprintf command
        fprintf("Unique optimal solution is:");
        fprintf("$x1$ = %.2$f$, $x2$ = %.2$f\backslash n$", x(1), x(2));
        fprintf("Maximum Z = %.2$f\backslash n$", -Z);
```

Program

```
f = [-3, -2];
A = [1, 1;1, 2];
b = [4; 6];
lb = [0, 0];
[x, Z] = linprog(f, A, b, [], [], lb);
fprintf('Unique optimal solution is:');
fprintf('x1 = %.2f, x2 = %.2f\n', x(1), x(2));
fprintf('Maximum Z = %.2f\n', -Z);
```

Output

The output of the above program is

```
Optimal solution found.

Unique optimal solution is: x1 = 4.00, x2 = 0.00
Maximum Z = 12.00
```

Exercise Problem

Solve the following LPP

$$\text{Max } Z = 5x_1 + 3x_2$$

$$\begin{aligned}\text{Subject to:}\quad & x_1 + x_2 \leq 9\\ & 2x_1 + 3x_2 \leq 7\\ & x_1, x_2 \geq 0\end{aligned}$$

Practical No. 3

Aim

To plot a graph of linear programming problem with unbounded solution using MATLAB.

Problem

Solve the following LPP graphically

$$\text{Max } Z = 3x_1 + 2x_2$$

$$\text{Subject to:} \quad \begin{aligned} x_1 - x_2 &\geq 2 \\ x_1 + x_2 &\geq 4 \\ x_1, x_2 &\geq 0 \end{aligned}$$

Algorithm

Step 1: Initialize x_1
$x1 = linspace(0, 10, 100);$
Step 2: Plot the lines of the constraints
plot $(x1, x1 - 2, `r', `\text{DisplayName}', `x1 - x2 >= 2');$
hold on;
plot $(x1, 4 - x1, `b', `\text{DisplayName}', `x1 + x2 >= 4');$
Step 3: Set the limit of axes through the command xlim and ylim i.e.
xlim([0, 10]);ylim([0, 10]);
Step 4: Shading the feasible region with the help of "fill" command
fill$([2, 10, 10, 4], [0, 8, 10, 0], `g',' `\text{FaceAlpha}', 1, ``\text{DisplayName}", `\text{Feasible Region}');$
Step 5: Plot the vertices
plot$([0, 4, 2, 0], [0, 0, 2, 3], ``ro", ``\text{DisplayName}", ``\text{Vertices}");$
Step 6: Labeling the x_1 and x_2 axes
xlabel$(``x_1");$ ylabel$(``x_2");$

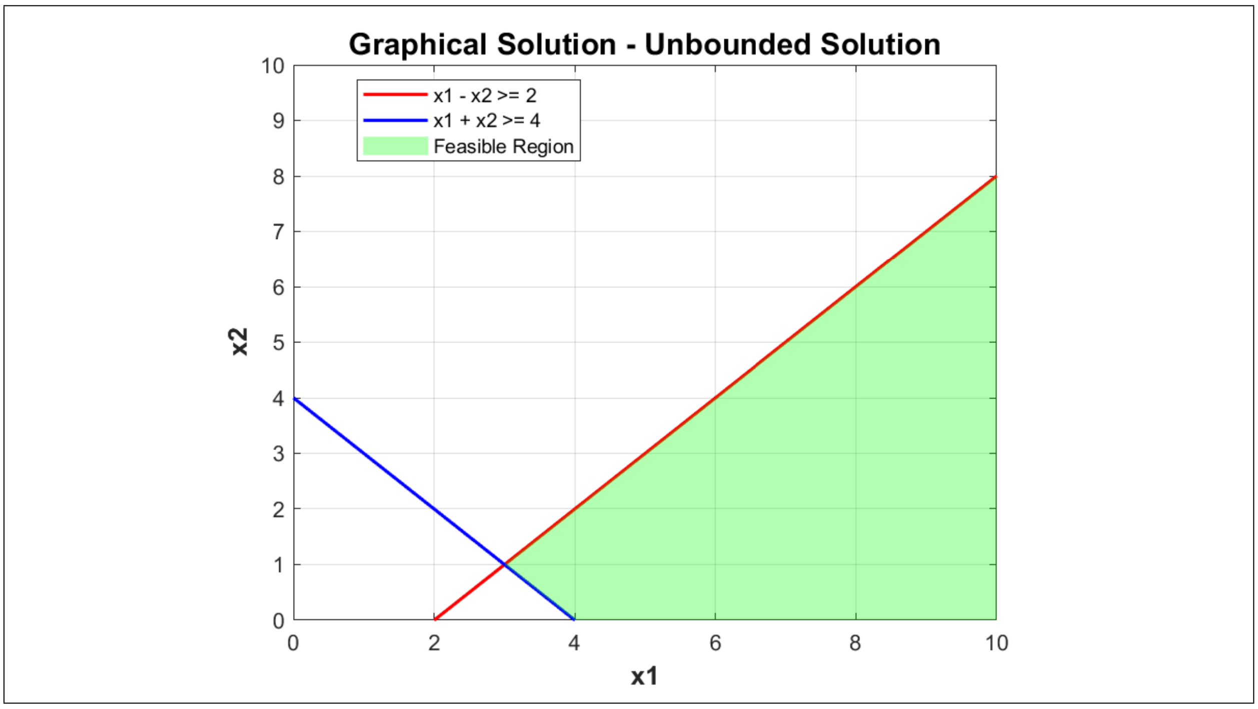
Program

```
x1 = linspace(0, 10, 100);
x2_1 = x1 - 2;      % x1 - x2 >= 2
x2_2 = 4 - x1;      % x1 + x2 >= 4
% Plot the constraints
figure;
plot(x1, x2_1, 'r', 'LineWidth', 1.5, 'DisplayName', 'x1 - x2 >= 2'); hold on;
plot(x1, x2_2, 'b', 'LineWidth', 1.5, 'DisplayName', 'x1 + x2 >= 4');
xlim([0, 10]);
ylim([0, 10]);
x_fill = [4, 3, 10, 10]; % x-coordinates of the boundary
y_fill = [0, 1, 8, 0];  % y-coordinates of the boundary
% Shade the feasible region
fill(x_fill, y_fill, 'g', 'FaceAlpha', 0.3, 'EdgeColor', 'none', 'DisplayName', '
% Add labels and legend
xlabel('x1', 'FontSize', 12, 'FontWeight', 'bold');
ylabel('x2', 'FontSize', 12, 'FontWeight', 'bold');
```

```
title('Graphical Solution - Unbounded Solution', 'FontSize', 14, 'FontWeight', 'bold');
legend('Location', 'best');
grid on;
hold off;
```

Output

The graphical representation of the unbounded solution.



Exercise Problem

Plot the following LPP

$$\text{Min } Z = 40x_1 + 80x_2$$

$$\begin{aligned}\text{Subject to:}\quad & 72x_1 + 12x_2 \geq 216\\ & 6x_1 + 24x_2 \geq 72\\ & 4x_1 + 20x_2 \geq 200\\ & x_1, x_2 \geq 0\end{aligned}$$

Practical No. 4

Aim

To plot a graph of linear programming problem with no feasible solution using MATLAB.

Problem

Plot the following LPP graphically

$$\text{Max } Z = 3x_1 + 2x_2$$

$$\begin{aligned} \text{Subject to:} \quad & x_1 + x_2 \leq 4 \\ & x_1 + x_2 \geq 6 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Algorithm

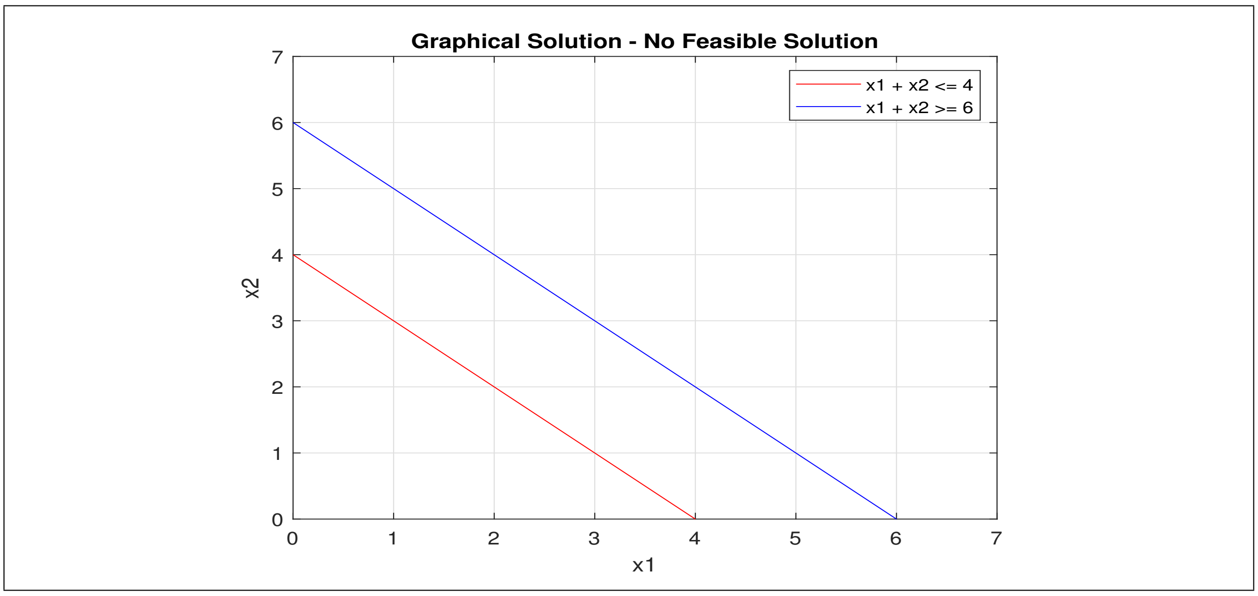
```
Step 1: Initialize $x_1$
        x1 = linspace(0, 7, 100);
Step 2: Plot the lines of the constraints
        plot $(x1, 4 - x1, `r', `DisplayName', `x1 + x2 <= 4')$;
        hold on;
        plot $(x1, 6 - x1, `b', `DisplayName', `x1 + x2 >= 6')$;
Step 3: Set the limit of axes through the command xlim and ylim i.e.
        xlim([0, 7]);ylim([0, 7]);
Step 6: Labeling the $x_1$ and $x_2$ axes
        xlabel(“$x_1$”); ylabel(“$x_2$”);
```

Program

```
x1 = linspace(0, 7, 100);
plot(x1, 4 - x1, ‘r’, ‘DisplayName’, ‘x1 + x2 <= 4’); hold on;
plot(x1, 6 - x1, ‘b’, ‘DisplayName’, ‘x1 + x2 >= 6’);
xlim([0, 7]);
ylim([0, 7]);
xlabel(‘x1’);
ylabel(‘x2’);
legend(‘show’);
title(‘Graphical Solution - No Feasible Solution’);
grid on;
hold off;
```

Output

The graph shows the no feasible solution.



Exercise Problem

Plot the following LPP

$$\text{Min } Z = 5x_1 + 8x_2$$

$$\begin{aligned} \text{Subject to:} \quad & 4x_1 + 6x_2 \leq 24 \\ & 4x_1 + 8x_2 \leq 40 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Practical No. 5

Aim

To plot a graph of linear programming problem with alternative solutions using MATLAB.

Problem

Plot the following LPP graphically

$$\text{Max } Z = 3x_1 + 2x_2$$

$$\begin{aligned} \text{Subject to:} \quad & x_1 + 2x_2 \leq 8 \\ & x_1 + x_2 \leq 6 \\ & x_1, x_2 \geq 0 \end{aligned}$$

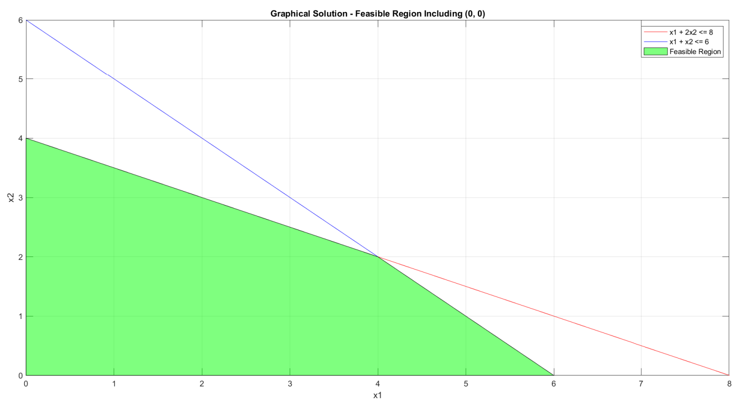
Algorithm

```
Step 1: Initialize the $x_1$
        x1 = linspace(0, 8, 100);
Step 2:Plot the lines of the constraints
        $plot(x1, (8 - x1)/2, `r', `DisplayName', `x1 + 2x2 <= 8');$
        hold on;
        $plot(x1, 6 - x1, `b', `DisplayName', `x1 + x2 <= 6');$
Step 3: Set the limit of axes through the command xlim and ylim i.e.
        xlim([0, 10]);ylim([0, 10]);
Step 4: Shading the feasible region with the help of "fill" command
        $fill([0, 6, 4, 0], [4, 0, 2, 4], `g', `FaceAlpha', 0.3, `DisplayName', `Feasible Region';$
Step 5: Labeling the $x_1$ and $x_2$ axes
        xlabel("$x_1$"); ylabel("$x_2$");
```

Program

```
x1 = linspace(0, 8, 100);

% Plot the constraint lines
plot(x1, (8 - x1) / 2, 'r', 'DisplayName', 'x1 + 2x2 <= 8'); hold on;
plot(x1, 6 - x1, 'b', 'DisplayName', 'x1 + x2 <= 6');

% Set axis limits for better visualization
xlim([0, 8]);
ylim([0, 6]);

% Define the feasible region vertices (including (0, 0))
x_feasible = [0, 0, 4, 6, 0];
y_feasible = [0, 4, 2, 0, 0];

% Fill the feasible region completely
fill(x_feasible, y_feasible, 'g', 'FaceAlpha', 0.5, 'DisplayName', 'Feasible Regi

% Label axes and add legend
xlabel('x1');
ylabel('x2');
legend('show');
title('Graphical Solution - Feasible Region Including (0, 0)');
grid on;

hold off;
```

Output



Exercise Problem

Plot the following LPP

$$\text{Min } Z = 2x_1 + 4x_2$$

$$\begin{aligned} \text{Subject to:} \quad & x_1 + 2x_2 \leq 5 \\ & x_1 + x_2 \leq 4 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Practical No. 6

Aim

To solve the linear programming problem through simplex method using MATLAB.

Problem

Solve the following LPP using simplex method.

$$\text{Max } Z = 5x_1 + 4x_2$$

$$\begin{aligned} \text{Subject to:} \quad & x_1 + 2x_2 \leq 5 \\ & 2x_1 + x_2 \leq 8 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Algorithm

Step 1: Input the objective function coefficients by using the following commands
disp('Enter the objective function coefficients:");
$c(1)$ = input('Enter Coefficient 1: ");
$c(2)$ = input('Enter Coefficient 2: ");
Step 2: In MATLAB
$c = -c$;
Step 3: Input the coefficients for the constraints
A(1,1) = input('Enter first coefficient of constraint 1: ');
A(1,2) = input('Enter second coefficient of constraint 1: ');
A(2,1) = input('Enter first coefficient of constraint 2: ');
A(2,2) = input('Enter second coefficient of constraint 2: ');
Step 4: Input the right-hand side values for the constraints and non-negativity constraints
b(1) = input('Enter right-hand side for constraint 1: ');
b(2) = input('Enter right-hand side for constraint 2: ');
lb = [0; 0];
Step 5: Maximize the problem using linprog command and also display the results
options = optimoptions('linprog', 'Algorithm', 'dual-simplex');
[x, fval] = linprog(c, A, b, [], [], lb, [], options);
disp('Optimal values for x:');
disp(x);
disp('Optimal value of the objective function Z:');
disp(-fval);

Program

```
disp('Enter the objective function coefficients:');
c(1) = input('Enter Coefficient 1: ');
c(2) = input('Enter Coefficient 2: ');
c = -c;
disp('Enter the coefficients for the constraints (A matrix):');
% Enter after converting constraints with "less than and equal to" sign
A(1,1) = input('Enter first coefficient of constraint 1: ');
A(1,2) = input('Enter second coefficient of constraint 1: ');
A(2,1) = input('Enter first coefficient of constraint 2: ');
A(2,2) = input('Enter second coefficient of constraint 2: ');
disp('Enter the right-hand side values for the constraints (b vector):');
b(1) = input('Enter right-hand side for constraint 1: ');
b(2) = input('Enter right-hand side for constraint 2: ');
lb = [0; 0];
options = optimoptions('linprog', 'Algorithm', 'dual-simplex');
[x, fval] = linprog(c, A, b, [], [], lb, [], options);
disp('Optimal values for x:');
disp(x);
disp('Optimal value of the objective function Z:');
disp(-fval);
```

Output

```
Enter the objective function coefficients:
Enter Coefficient 1: 5
Enter Coefficient 2: 4
Enter the coefficients for the constraints (A matrix):
Enter first coefficient of constraint 1: 1
Enter second coefficient of constraint 1: 2
Enter first coefficient of constraint 2: 2
Enter second coefficient of constraint 2: 1
Enter the right-hand side values for the constraints (b vector):
Enter right-hand side for constraint 1: 5
Enter right-hand side for constraint 2: 8
Optimal solution found.
Optimal values for x:
3.6667, 0.6667
Optimal value of the objective function Z:
21
```

Exercise Problem

Plot the following LPP

$$\text{Max } Z = 3x_1 - x_2$$

$$\begin{aligned} \text{Subject to:} \quad & x_1 - x_2 \leq 3 \\ & 2x_1 - x_2 \leq 0 \\ & 2x_1 - x_2 \geq 12 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Practical No. 7

Aim

To solve the linear programming problem using simplex method through MATLAB.

Problem

Solve the following LPP using simplex method.

$$\text{Max } Z = 25x_1 + 30x_2$$

$$\begin{aligned} \text{Subject to:} \quad & x_1 + 2x_2 \leq 9 \\ & 2x_1 + x_2 \leq 7 \\ & 3x_1 + 2x_2 \leq 5 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Algorithm

```
Step 1: Define a function file in MATLAB with help of c, A, and b
        function []=Simplex(c, A, b);
Step 2: Initialize c, A, and b
        $[m, n] = size(A);$
        $A = [A, eye(m)];$
        $c = [c, zeros(1, m)];$
Step 3: Check the condition $tableau(end, 1 : end - 1) < 0$
Step 4: Using for loop in side the while condition to calculate the values of the table
Step 5: Display the results.
```

Program

```
% Instructions to run this program:
% step1: Enter the inputs in coomand window by using semicolomn as a separator
% for example: c=[25, 30]; A=[1,2; 2,1; 3,2]; b=[9;7;5];
% step2: Type simplex(c, A, b) and  then press enter.
function []=Simplex(c, A, b)
    [m, n] = size(A);
    A = [A, eye(m)];
    c = [c, zeros(1, m)];
    tableau = [A, b; -c, 0];
    while any(tableau(end, 1:end-1) < 0)
        [~, pivotCol] = min(tableau(end, 1:end-1));
        ratios = tableau(1:end-1, end) ./ tableau(1:end-1, pivotCol);
        ratios(ratios < 0) = Inf;
        [~, pivotRow] = min(ratios);
        pivotElement = tableau(pivotRow, pivotCol);
        tableau(pivotRow, :) = tableau(pivotRow, :) / pivotElement;
        for i = [1:pivotRow-1, pivotRow+1:m+1]
            tableau(i, :) = tableau(i, :) - tableau(i, pivotCol)
            * tableau(pivotRow, :);
        end
    end
    optSolution = zeros(1, n);
    for i = 1:n
        if any(tableau(1:end-1, i) == 1) &&
        all(tableau(1:end-1, i) == 0 | tableau(1:end-1, i) == 1)
            optSolution(i) = tableau(find(tableau(1:end-1, i) == 1), end);
        end
    end
    optValue = -tableau(end, end);
    disp('Optimal Solution:');
    disp(optSolution);
    disp(['Optimal Value: ', num2str(optValue)]);
end
```

Output

```
>> c = [25, 30];
>> A = [1, 2; 2, 1; 3, 2];
>> b = [9; 7; 5];
>> Simplex(c, A, b)
Optimal Solution:
         0    2.5000
Optimal Value: -75
```

Exercise Problem

Plot the following LPP

$$\text{Max } Z = 100x_1 + 80x_2$$

$$\begin{aligned} \text{Subject to:} \quad & 6x_1 + 4x_2 \leq 7200 \\ & 2x_1 + 4x_2 \leq 4000 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Practical No. 8

Aim

To solve the linear programming problem through Two-phase method using MATLAB code.

Problem

Solve the following LPP using simplex method.

$$\text{Max } Z = 5x_1 + 4x_2$$

$$\begin{aligned} \text{Subject to:} \quad & x_1 + 2x_2 \leq 5 \\ & 2x_1 + x_2 \leq 8 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Algorithm

Step 1: Define the coefficients of the objective function in Phase 1.
Step 2: Enter the system of constraints with artificial variables.
Step 3: Run the phase 1 using the linprog command and find the values of op.
Step 4: Evaluate the x_phase1 and $fval\_phase1$ values by using c_phase1, A, b, and op.
Step 5: Display the results of Phase 1, x_phase1, $fval\_phase1$.
Step 6: Check the condition
Step 7: Minimize the original objective function by using the following command
$[x\_phase2, fval\_phase2] = \text{linprog}(c\_phase2, A, b, [], [], lb, [], op)$
Step 8: Display the results of optimal solution for the original problem

Program

```
c_phase1 = [0; 0; 0; 1; 1];
A = [1, 2, 2, 0, 0;
     3, -1, 3, 1, 0];
b = [6; 8];
lb = zeros(5, 1);
op = optimoptions('linprog', 'Algorithm', 'dual-simplex', 'Display', 'iter');
[x_phase1, fval_phase1] = linprog(c_phase1, A, b, [], [], lb, [], op);
disp('Phase 1 solution (artificial variables):');
disp(x_phase1);
disp('Phase 1 objective function value (should be 0 if feasible):');
disp(fval_phase1);
if fval_phase1 > 0
    disp('No feasible solution exists.');
else
    disp('Feasible solution found. Proceeding to Phase 2...');
    c_phase2 = [-3; -2; 0; 0; 0];
    [x_phase2, fval_phase2] = linprog(c_phase2, A, b, [], [], lb, [], op)
    % Display Phase 2 results (optimal solution for the original problem)
    disp('Optimal values for x1, x2, s1, s2, a1:');
    disp(x_phase2);
    disp('Optimal value of the objective function Z:');
    disp(-fval_phase2);
end
```

Output

```
Phase 1 solution (artificial variables):
     0
     0
     0
     0
     0

Phase 1 objective function value (should be 0 if feasible):
     0

Feasible solution found. Proceeding to Phase 2...

 Iter      Time            Fval  Primal Infeas    Dual Infeas
    0     0.001    0.000000e+00   0.000000e+00   1.428720e+00
    2     0.011   -1.228571e+01   0.000000e+00   0.000000e+00

Optimal solution found.

Optimal values for x1, x2, s1, s2, a1:
    3.1429
    1.4286
         0
         0
         0

Optimal value of the objective function Z:
   12.2857
```

Exercise Problem

Plot the following LPP

$$\text{Min } Z = x_1 - 2x_2 - 3x_3$$

$$\begin{aligned} \text{Subject to:} \quad & -2x_1 + x_2 + 3x_3 = 2 \\ & 2x_1 + 3x_2 + 4x_3 = 1 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

Practical No. 9

Aim

To solve the linear programming problem with no feasible solution using MATLAB.

Problem

Solve the following LPP using simplex method.

$$\text{Max } Z = 3x_1 + 2x_2$$

$$\begin{aligned} \text{Subject to:} \quad & x_1 + x_2 \leq 1 \\ & -x_1 - x_2 \leq -3 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Algorithm

Step 1: Enter the value of c, A, and b.
$c = [-3; -2]; A = [11; -1-1]; b = [1; -3];$
Step 2: Evaluate the value of x, fval, and output using the following command
options = optimoptions($'linprog',' Algorithm',' dual{-}simplex',' Display',' iter'$); $[x, fva$
linprog $(c, A, b, [], [], lb, [], options)$;
Step 3: Check the condition using if else statement
Step 4: Display the results.

Program

```
c = [-3; -2];
A = [1 1; -1 -1];
b = [1; -3];
lb = [0; 0];
options = optimoptions(‘linprog’,‘Algorithm’,‘dual-simplex’,‘Display’,‘iter’);
[x, fval, exitflag, output] = linprog(c, A, b, [], [], lb, [], options);
if exitflag == -2
    disp(‘The problem has no feasible solution.’);
elseif exitflag == 1
    disp([‘Optimal solution found: ’, num2str(fval)]);
else
    disp(‘The problem could not be solved for some other reason.’);
end
```

Output

```
Linprog stopped because no point satisfies the constraints.
The problem has no feasible solution.
```

Exercise Problem

Plot the following LPP

$$\text{Max } Z = x_1 + x_2$$

$$\begin{aligned} \text{Subject to:} \quad & x_1 - x_2 \leq -1 \\ & -x_1 + x_2 \leq 0 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Practical No. 10

Aim

To solve the linear programming problem with unbounded solution using MATLAB.

Problem

Check the uniqueness of the LPP using MATLAB.

$$\text{Max } Z = 2x_1 + 1x_2$$

$$\begin{aligned} \text{Subject to:} \quad & -x_1 + x_2 \leq -2 \\ & x_1 - 2x_2 \leq 2 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Algorithm

Step 1: Enter the values of c, A, and b
Step 2: Solve the problem using the optimoptions command
Step 3: Calculate the value of x, fval, exitflag, and output using linprog command
Step 4: Display the results using the disp command

Program

```
c = [-2; -1];
A = [-1 1; 1 -2];
b = [-2; 2];
lb = [0; 0];
options = optimoptions('linprog','Algorithm','dual-simplex','Display','iter');
[x, fval, exitflag, output] = linprog(c, A, b, [], [], lb, [], options);
if exitflag == -3
    disp('The problem is unbounded.');
elseif exitflag == 1
    disp(['Optimal solution found: ', num2str(fval)]);
else
    disp('The problem could not be solved for some other reason.');
end
```

Output

```
Iter      Time            Fval  Primal Infeas    Dual Infeas
    0      0.001    0.000000e+00   2.378414e+00   1.783811e+00
    1      0.001   -4.000000e+00   2.378414e+00   0.000000e+00
    2      0.001   -4.000000e+00   0.000000e+00   2.973018e+00
    3      0.001   -4.000000e+00   0.000000e+00   2.973018e+00
The problem is unbounded.
```

Exercise Problem

Plot the following LPP

$$\text{Max } Z = 60x_1 + 20x_2$$

$$\begin{aligned} \text{Subject to:} \quad & 2x_1 + 4x_2 \geq 120 \\ & 8x_1 + 6x_2 \geq 240 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Z X

Practical No. 11

Aim

Check the uniqueness of the LPP using MATLAB.

Problem

Solve the following LPP.

$$\text{Max } Z = x_1 + x_2$$

$$\begin{aligned} \text{Subject to:} \quad & x_1 + x_2 \leq 4 \\ & x_1 \leq 3 \\ & x_2 \leq 3 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Algorithm

Step 1: Enter the objective function coeffiecients.
Step 2: Enter the constraints coefficients.
Step 3: Calculate the values of x, fval using the lingprog command.
Step 4: Check the condition using if else statement.
Step 5: Display the results using the disp command.

Program

```
c = [-1; -1];
A = [1 1; 1 0; 0 1];
b = [4; 3; 3];
lb = [0; 0];
options = optimoptions(‘linprog’,‘Algorithm’,‘dual-simplex’,‘Display’,‘iter’);
[x, fval, exitflag, output] = linprog(c, A, b, [], [], lb, [], options);
if exitflag == 1
    disp([‘Optimal solution found: Z = ’, num2str(-fval)]);
    disp([‘Optimal point: x1 = ’, num2str(x(1)), ’, x2 = ’, num2str(x(2))]);
    disp(‘Note: There may be infinitely many solutions along the line segment of the feasib
elseif exitflag == -3
    disp(‘The problem is unbounded.’);
elseif exitflag == -2
    disp(‘The problem has no feasible solution.’);
else
    disp(‘The problem could not be solved for some other reason.’);
end
```

Output

```
 Iter      Time            Fval  Primal Infeas    Dual Infeas
    0     0.001    0.000000e+00    0.000000e+00   1.414201e+00
    1      0.001   -3.999964e+00   1.000000e+00   0.000000e+00
    2      0.001   -4.000000e+00   0.000000e+00   0.000000e+00

Optimal solution found.

Optimal solution found: Z = 4
Optimal point: x1 = 3, x2 = 1
Note: There may be infinitely many solutions along the line segment
of the feasible region.
```

Exercise Problem

Solve the following LPP.

$$\text{Max } Z = 5x_1 + 9x_2$$

$$\begin{aligned}\text{Subject to:}\quad & 3x_1 + 5x_2 \leq 7\\ & 2x_1 \leq 3\\ & 8x_2 \leq 3\\ & x_1, x_2 \geq 0\end{aligned}$$

Practical No. 12

Aim

Create the simplex table of simplex algorithm for LPP using MATLAB.

Problem

Find the simplex table for the following LPP.

$$\text{Max } Z = x_1 + x_2$$

$$\begin{aligned}\text{Subject to:}\quad & x_1 + x_2 \leq 4\\ & x_1 \leq 3\\ & x_2 \leq 3\\ & x_1, x_2 \geq 0\end{aligned}$$

Algorithm

Step 1: Enter the c, A, and b.
Step 2: Create table using the following command.

tableau = [A; c];
Step 3: Display the results using disp command.

Program

```
c = [-3 -5 0 0 0];
A = [2 3 1 0 12;
     1 1 0 1 5];
tableau = [A; c];
disp('Initial Simplex Tableau:')
disp(tableau)
```

Output

```
>> Problem_12
Initial Simplex Tableau:
     2     3     1     0    12
     1     1     0     1     5
    -3    -5     0     0     0
```

Exercise Problem

Find the simplex table for the following LPP.

$$\text{Max } Z = 2x_1 + 3x_2$$

$$\begin{aligned} \text{Subject to:} \quad & 3x_1 + 5x_2 \leq 8 \\ & 7x_1 \leq 4 \\ & 4x_2 \leq 3 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Practical No. 13

Aim

Check the uniqueness of an LPP using MATLAB.

Problem

Solve the following LPP.

$$\text{Max } Z = 3x_1 + 2x_2$$

$$\begin{aligned} \text{Subject to:} \quad & 4x_1 + 5x_2 \leq 6 \\ & 3x_1 + 2x_2 \leq 12 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Algorithm

Step 1: Enter the coefficient of c, A, and b.
Step 2: Using the pre-defined optimoptions to solve the LPP.
Step 3: Evaluate the x, fval, and output using if else statement.
Step 4: Display results.

Program

```
c = [-3; -2];
A = [4, 5; 3, 2];
b = [6; 12];
op = optimoptions(‘linprog’, ‘Algorithm’, ‘dual-simplex’, ‘Display’, ‘iter’);
[x, fval, exitflag, output] = linprog(c, A, b, [], [], [0; 0], [], op);
if exitflag == 1
    disp(‘Optimal solution found:’);
    disp(x);
    disp(‘Maximum Z:’);
    disp(-fval);
elseif exitflag == -3
    disp(‘The problem is unbounded.’);
elseif exitflag == -2
    disp(‘No feasible solution exists.’);
else
    disp(‘Multiple optimal solutions exist.’);
end
```

Output

```
Iter      Time            Fval  Primal Infeas    Dual Infeas
    0     0.001    0.000000e+00   0.000000e+00   9.164285e-01
    1     0.001   -1.200000e+01   7.400828e+00   0.000000e+00
    2     0.002   -4.500000e+00   0.000000e+00   0.000000e+00
```

```
Optimal solution found.

Optimal solution found:
    1.5000
         0

Maximum Z:
    4.5000
```

Exercise Problem

Solve the following LPP.

$$\text{Max } Z = 4x_1 + 5x_2$$

$$\begin{aligned} \text{Subject to:} \quad & 3x_1 + 8x_2 \leq 10 \\ & 7x_1 + 4x_2 \leq 5 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Practical No. 14

Aim

To plot a graph of feasible region and solve an LPP with three constraints using MATLAB.

Problem

Solve the following LPP.

$$\text{Max } Z = x + y$$

$$\begin{aligned} \text{Subject to:} \quad & x + y \leq 4 \\ & y \leq 3 \\ & x \leq 2 \\ & x, y \geq 0 \end{aligned}$$

Algorithm

Step 1: Initialize x and y.
Step 2: Define the constraints.
Step 3: Plots the line of the constraints.

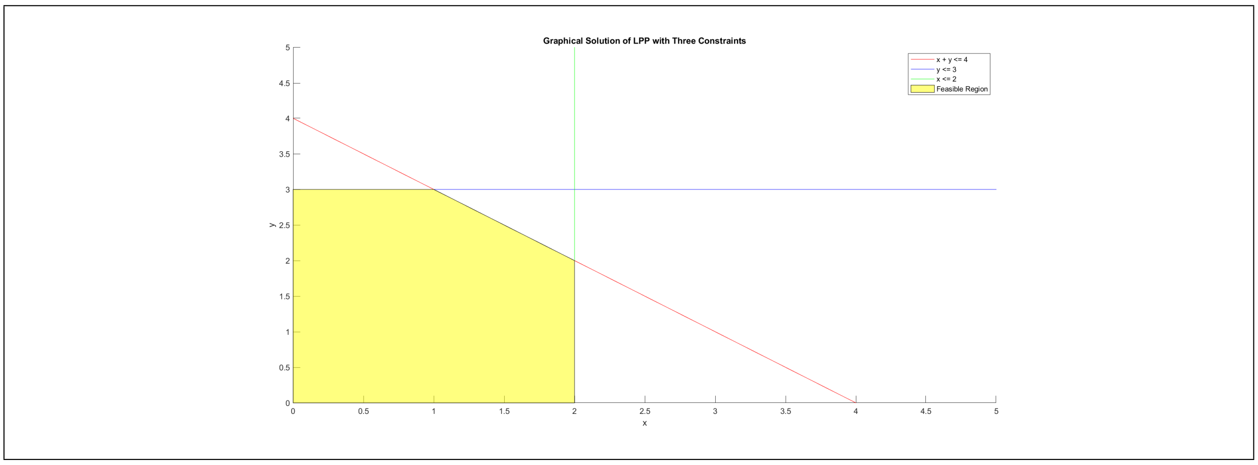
Step 4: Enter the corner points.
Step 5: Calculate the values of max_Z using for loop statement.
Step 6: Print the results using fprintf command.

Program

```
x = linspace(0, 5, 100);
y = linspace(0, 5, 100);
y1 = 4 - x;         % From x + y <= 4, rearrange as y = 4 - x
y2 = 3 * ones(size(x));  % From y <= 3, create a horizontal line y = 3
x2 = 2 * ones(size(y));  % From x <= 2, create a vertical line x = 2
figure;
hold on;
plot(x, y1, ‘r’, ‘DisplayName’, ‘x + y <= 4’); % Constraint 1
plot(x, y2, ‘b’, ‘DisplayName’, ‘y <= 3’);     % Constraint 2
plot(x2, y, ‘g’, ‘DisplayName’, ‘x <= 2’);     % Constraint 3
% Set axis limits for better visualization
axis([0 5 0 5]);
% Redefine feasible region to exclude above (1, 3) and (2, 2)
fill([0, 0, 1, 2, 2], [0, 3, 3, 2, 0], ‘y’, ‘FaceAlpha’, 0.5, ‘DisplayName’, ‘Fea
% Label axes and plot
xlabel(‘x’);
ylabel(‘y’);
legend show;
title(‘Graphical Solution of LPP with Three Constraints’);
% Calculate Z at each corner point (new region)
corner_points = [0 0; 0 3; 1 3; 2 2];
Z = 3 * corner_points(:,1) + 2 * corner_points(:,2);

% Display the results
fprintf(‘Corner Points and Z values:\n’);
for i = 1:size(corner_points, 1)
    fprintf(‘Point (x, y) = (%.2f, %.2f), Z = %.2f\n’, corner_points(i,1), corner
end
% Find the optimal solution
[max_Z, max_index] = max(Z);
optimal_point = corner_points(max_index, :);
fprintf(‘\nOptimal Solution:\n’);
fprintf(‘x = %.2f, y = %.2f, Maximum Z = %.2f\n’, optimal_point(1), optimal_point
hold off;
```

Output

```
Corner Points and Z values:
Point (x, y) = (0.00, 0.00), Z = 0.00
Point (x, y) = (2.00, 2.00), Z = 10.00
Point (x, y) = (2.00, 3.00), Z = 12.00
Point (x, y) = (1.00, 3.00), Z = 9.00

Optimal Solution:
x = 2.00, y = 3.00, Maximum Z = 12.00
```



Exercise Problem

Solve the following LPP.

$$\text{Max } Z = 2x + 3y$$

$$\begin{aligned} \text{Subject to:} \quad & 3x + 5y \leq 9 \\ & y \leq 5 \\ & x \leq 8 \\ & x, y \geq 0 \end{aligned}$$

Practical No. 15

Aim

To plot a graph of feasible region of an LPP with three variables using MATLAB.

Problem

Solve the following LPP.

$$\text{Max } Z = x_1 + x_2 + x_3$$

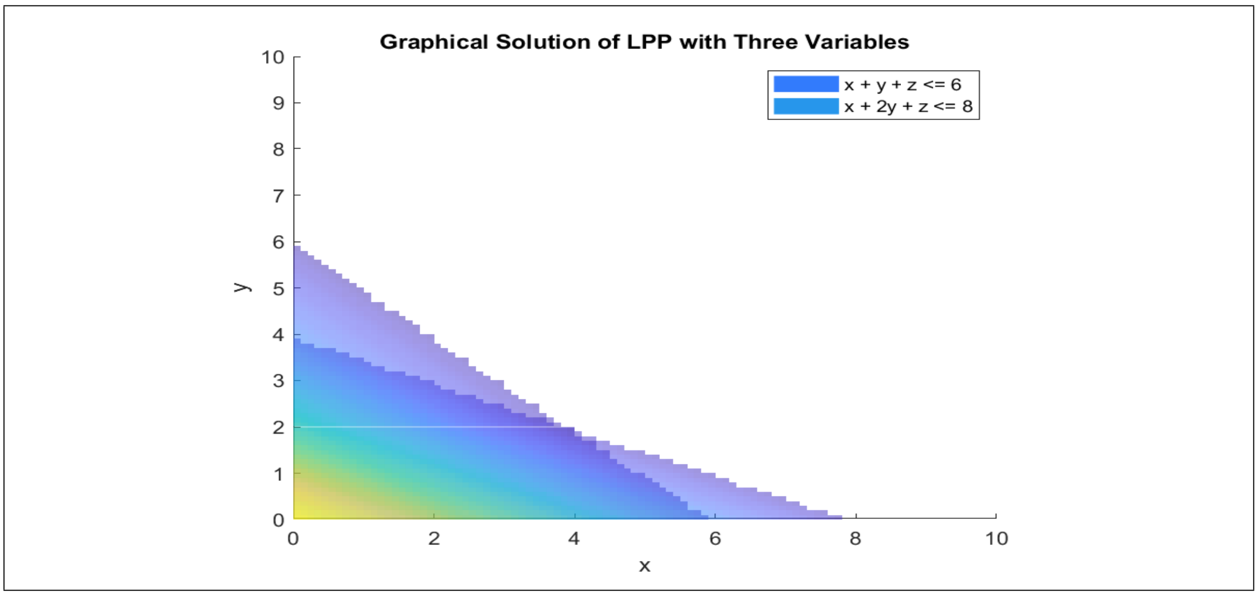
$$\text{Subject to:} \quad \begin{aligned} &x_1 + x_2 + x_3 \leq 6 \\ &x_1 + 2x_2 + x_3 \leq 8 \\ &x_1, x_2, x_3 \geq 0 \end{aligned}$$

Algorithm

Step 1: Initialize the x and y value using meshgrid command.
Step 2: Enter the constraints of three variables.
Step 3: Plot the lines of the constraints.
Step 4: Display the graph.

Program

```
[x, y] = meshgrid(0:0.1:10, 0:0.1:10);
z1 = 6 - x - y;
z2 = 8 - x - 2*y;
z1(z1 < 0) = NaN;
z2(z2 < 0) = NaN;
figure;
hold on;
surf(x, y, z1, ‘FaceAlpha’, 0.5, ‘EdgeColor’, ‘none’, ‘DisplayName’,
‘x + y + z <= 6’);
surf(x, y, z2, ‘FaceAlpha’, 0.5, ‘EdgeColor’, ‘none’, ‘DisplayName’,
‘x + 2y + z <= 8’);
axis([0 10 0 10 0 10]);
xlabel(’x’);
ylabel(’y’);
zlabel(’z’);
legend;
title(’Graphical Solution of LPP with Three Variables’);
hold off;
```



Output

Exercise Problem

Solve the following LPP.

$$\text{Max } Z = 2x_1 + 5x_2 + 7x_3$$

$$\begin{aligned} \text{Subject to:} \quad & 3x_1 + 2x_2 + x_3 \leq 9 \\ & 4x_1 + 7x_2 + x_3 \leq 10 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

Practical No. 16

Aim

To solve the LPP through Big-M method using MATLAB.

Problem

Solve the following LPP.

$$\text{Max } Z = 3x_1 + 2x_2$$

$$\begin{aligned} \text{Subject to:} \quad & 3x_1 + 4x_2 \leq 4 \\ & 5x_1 + x_2 \geq 6 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Algorithm

```
Step 1: Initialize the Big-M value
        $M = 1000$;
Step 2: Enter the coefficients of the objective function.
Step 3: Enter the coefficient matrix for the constraints.
Step 4: Evaluate the x, fval using the linprog i.e.
        $[x, fval] = linprog(c, A, b, [], [], lb, [], options)$;
Step 5: Display the results using disp command
```

Program

```
M = 1000;
c = [-3; -2; 0; 0; M];
A = [3, 4, 5, 0, 0;
     5, 1, 0, -1, 1];
b = [4; 6];
lb = zeros(5, 1);
options = optimoptions('linprog', 'Algorithm', 'dual-simplex', 'Display', 'iter')
[x, fval] = linprog(c, A, b, [], [], lb, [], options);
disp('Optimal values for x1, x2, s1, s2, a1:');
disp(x);
disp('Optimal value of the objective function Z:');
disp(-fval);
```

Output

```
 Iter      Time            Fval  Primal Infeas    Dual Infeas
    0         0    0.000000e+00   0.000000e+00   9.999998e-01
    1     0.001   -4.000000e+00   0.000000e+00   0.000000e+00

Optimal solution found.

Optimal values for x1, x2, s1, s2, a1:
    1.3333
         0
         0
    0.6667
         0

Optimal value of the objective function Z:
     4
```

Exercise Problem

Solve the following LPP.

$$\text{Max } Z = 5x_1 + 6x_2$$

$$\begin{aligned} \text{Subject to:} \quad & 4x_1 + 8x_2 \leq 6 \\ & 3x_1 + 2x_2 \geq 7 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Practical No. 17

Aim

To solve the real-world problem through LPP using MATLAB.

Problem

Solve the following LPP.
A company produces three products (P1, P2, and P3), each requiring different amounts of three resources (R1, R2, and R3). The resource constraints are as follows: R1 has a maximum of 100 units, R2 has a maximum of 80 units, and R3 has a maximum of 60 units. The profit per unit of P1, P2, and P3 is Rupees 40, 30, and 50, respectively. The company wants to determine the optimal number of units of each product to produce in order to maximize profit, while satisfying the resource constraints. Formulate and solve the linear programming problem to find the optimal production quantities and maximum profit.

$$\text{Max } Z = 40x_1 + 30x_2 + 50x_3$$

$$\begin{aligned} \text{Subject to:} \quad & 2x_1 + x_2 + 3x_3 \leq 100 \\ & x_1 + 2x_2 + 2x_3 \leq 80 \\ & x_1 + x_2 + x_3 \leq 60 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

Algorithm

Step 1: Enter the c, A, and b.
Step 2: Solve the problem using optimoptions.
Step 3: Calculate the values of x, fval using the following
$[x, fval, exitflag, output] = linprog(c, A, b, [], [], lb, [], options);$
Step 4: Display the results.

Program

```
% Coefficients of the objective function
c = [-40; -30; -50; 0; 0; 0];  % Include slack variables with 0 coefficients

% Coefficient matrix for constraints (A matrix)
A = [2, 1, 3, 1, 0, 0;    % Coefficients for the first constraint (with slack)
     1, 2, 2, 0, 1, 0;    % Coefficients for the second constraint (with slack)
     1, 1, 1, 0, 0, 1];   % Coefficients for the third constraint (with slack)

% Right-hand side of the constraints
b = [100; 80; 60];

% Lower bounds for decision variables and slack variables (all >= 0)
lb = zeros(6, 1);  % 6 variables (3 decision + 3 slack)

% Use linprog to solve the problem
options = optimoptions('linprog', 'Algorithm', 'dual-simplex', 'Display', 'iter')
[x, fval, exitflag, output] = linprog(c, [], [], A, b, lb, [], options);

% Display the results
if exitflag == 1
    disp('Optimal solution found:')
    disp(['Units of Product P1 (x1) = ', num2str(x(1))])
    disp(['Units of Product P2 (x2) = ', num2str(x(2))])
    disp(['Units of Product P3 (x3) = ', num2str(x(3))])
    disp(['Slack variable for Constraint 1 (s1) = ', num2str(x(4))])
    disp(['Slack variable for Constraint 2 (s2) = ', num2str(x(5))])
    disp(['Slack variable for Constraint 3 (s3) = ', num2str(x(6))])
    disp(['Maximum Profit = ', num2str(-fval)])  % Negate to get max profit
else
    disp('The problem does not have an optimal solution.')
end
```

Output

```
 Iter      Time            Fval  Primal Infeas    Dual Infeas
    0     0.015    0.000000e+00   0.000000e+00   2.500000e+01
    2     0.015   -2.200000e+03   0.000000e+00   0.000000e+00

Optimal solution found.

Optimal solution found:
Units of Product P1 (x1) = 40
Units of Product P2 (x2) = 20
Units of Product P3 (x3) = 0
Slack variable for Constraint 1 (s1) = 0
Slack variable for Constraint 2 (s2) = 0
Slack variable for Constraint 3 (s3) = 0
Maximum Profit = 2200
```

Exercise Problem

A company produces three products (P1, P2, and P3), each requiring different amounts of three resources (R1, R2, and R3). The resource constraints are as follows: R1 has a maximum of 1000 units, R2 has a maximum of 800 units, and R3 has a maximum of 600 units. The profit per unit of P1, P2, and P3 is Rupees 400, 300, and 500, respectively. The company wants to determine the optimal number of units of each product to produce in order to maximize profit, while satisfying the resource constraints. Formulate and solve the linear programming problem to find the optimal production quantities and maximum profit.

Practical No. 18

Aim

To solve a LPP with minimization objective function using MATLAB.

Problem

Solve the following LPP.

$$\text{Min } Z = 50x_1 + 80x_2$$

$$\begin{aligned} \text{Subject to:} \qquad & -3x_1 - 2x_2 \leq -120 \\ & -x_1 - 4x_2 \leq -80 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Algorithm

Step 1: Enter the c, A, and b.
Step 2: Calculate the values of x, fval using the following.
$[x, fval, exitflag, output] = linprog(c, A, b, [], [], lb, [])$;
Step 3: To find the values of minimum cost using the if else statement.
Step 4: Display the results.

Program

```
c = [50; 80];
A = [-3, -2;
     -1, -4];
b = [-120; -80];
lb = [0; 0];
[x, fval, exitflag, output] = linprog(c, A, b, [], [], lb, []);
if exitflag == 1
    disp('Optimal solution found:');
    disp(['Units of x1 = ', num2str(x(1))]);
    disp(['Units of x2 = ', num2str(x(2))]);
    disp(['Minimum Cost = ', num2str(fval)]);
else
    disp('The problem does not have an optimal solution.');
end
```

Output

```
>>
Units of x1 = 32
Units of x2 = 12
Minimum Cost = 2560
```

Exercise Problem

Solve the following LPP.

$$\text{Min } Z = 70x_1 + 100x_2$$

$$\begin{aligned}\text{Subject to:}\quad & -5x_1 - 7x_2 \leq -130\\ & -3x_1 - 4x_2 \leq -90\\ & x_1, x_2 \geq 0\end{aligned}$$

Practical No. 19

Aim

To plot the graph of a LPP with minimization objective function using MATLAB.

Problem

Solve the following LPP.

$$\text{Min } Z = 50x_1 + 80x_2 + 40x_3$$

$$\begin{aligned}\text{Subject to:}\quad & x_1 + x_2 + x_3 \leq 30\\ & 2x_1 + x_2 + 3x_3 \leq 60\\ & x_1 + 2x_2 + x_3 \leq 40\\ & x_1, x_2, x_3 \geq 0\end{aligned}$$

Algorithm

Step 1: Enter the c, A, and b.
Step 2: Calculate the values of x, fval using the following
$[x, fval, exitflag, output] = linprog(c, A, b, [], [], lb, []);$
Step 3: To find the values of minimum cost using the if else statement.
Step 4: Display the results.

Program

```
[x1, x2] = meshgrid(0:1:30, 0:1:30);
x3_1 = 30 - x1 - x2;
x3_2 = (60 - 2*x1 - x2) / 3;
x3_3 = 40 - x1 - 2*x2;
x3_1(x3_1 < 0) = NaN;
x3_2(x3_2 < 0) = NaN;
x3_3(x3_3 < 0) = NaN;
figure;
hold on;
surf(x1, x2, x3_1, ‘FaceAlpha’, 0.5, ‘EdgeColor’, ‘none’, ‘DisplayName’,
‘x1 + x2 + x3 <= 30’);
surf(x1, x2, x3_2, ‘FaceAlpha’, 0.5, ‘EdgeColor’, ‘none’, ‘DisplayName’,
‘2x1 + x2 + 3x3 <= 60’);
surf(x1, x2, x3_3, ‘FaceAlpha’, 0.5, ‘EdgeColor’, ‘none’, ‘DisplayName’,
‘x1 + 2x2 + x3 <= 40’);
axis([0 30 0 30 0 30]);
xlabel(’x1’);
ylabel(’x2’);
zlabel(’x3’);
legend;
title(‘Feasible Region for Minimization Problem’);
hold off;
```

Output

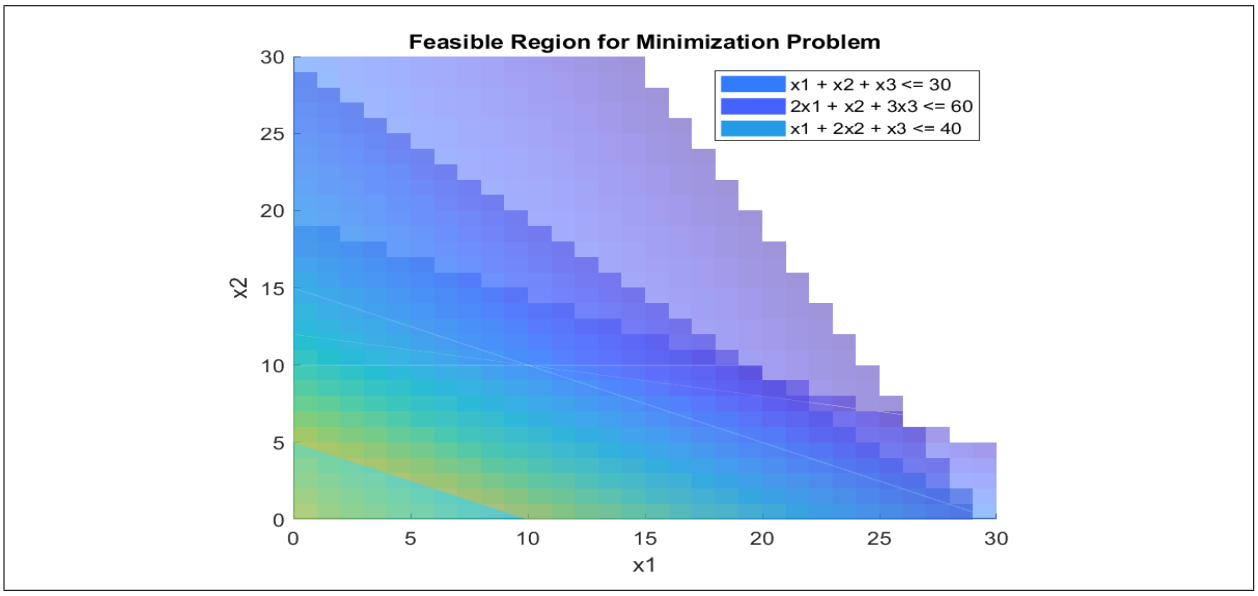
Exercise Problem

Solve the following LPP.

$$\text{Min } Z = 50x_1 + 80x_2 + 40x_3$$

$$\begin{aligned} \text{Subject to:} \quad & x_1 + x_2 + x_3 \leq 50 \\ & 3x_1 + 2x_2 + 5x_3 \leq 80 \\ & x_1 + 3x_2 + 2x_3 \leq 40 \\ & x_1, x_2, x_3 \geq 0 \end{aligned}$$

Practical No. 20



Aim

To solve a LPP with minimization objective function using MATLAB.

Problem

Solve the following LPP.

$$\text{Min } Z = x_1 + x_2$$

$$\text{Subject to:} \quad -x_1 + x_2 \leq -1$$
$$x_1, x_2 \geq 0$$

Algorithm

Step 1: Enter the c, A, and b.
Step 2: Calculate the values of x, fval using the following
$[x, fval, exitflag, output] = linprog(c, A, b, [], [], lb, []);$
Step 3: To find the values of minimum cost using the if else statement.
Step 4: Display the results.

Program

```
c = [1; 1];
A = [-1, 1];
b = [-1];
lb = [0; 0];
```

```
[x, fval, exitflag, output] = linprog(c, A, b, [], [], lb, []);
if exitflag == 1
    disp('Optimal solution found:');
    disp(['x1 = ', num2str(x(1))]);
    disp(['x2 = ', num2str(x(2))]);
    disp(['Minimum Cost = ', num2str(fval)]);
elseif exitflag == -2
    disp('The problem is unbounded.');
else
    disp('The problem does not have an optimal solution.');
end
```

Output

```
>>
Optimal solution found:
x1 = 1
x2 = 0
Minimum Cost = 1
```

Exercise Problem

Solve the following LPP.

$$\text{Min } Z = 2x_1 + 3x_2$$

$$\begin{aligned} \text{Subject to:} \quad & x_1 - x_2 \geq 1 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Chapter 2

Mathematical Statistics

Chapter 3

Mathematical Modeling

Practical No. 1

Aim

To model the population growth of a species over time using exponential growth and analyze the solution using MATLAB.

Problem

Given an initial population size, predict the population after a certain time period using the exponential growth model.

Theory

The population growth can be modeled by the differential equation:

$$\frac{dP}{dt} = rP, \quad P(0) = P_0,$$

where

- $P(t)$ is the population at time t
- r is the growth rate

The solution to this differential equation is

$$P(t) = P_0 e^{rt}$$

Algorithm

Step 1: Initialize initial population P_0 and the growth rate r,
Step 2: Define the time period over which the population is to be calculated,
Step 3: Use the exponential growth formula to calculate the population at each time step $P(t) = P_0e^{rt}$,
Step 4: Plot the population over time,
Step 5: Define and calculate r.

Program

```
% Parameters
P0 = 100; % Initial population
r = 0.05; % Growth rate
t = 0:0.1:10; % Time vector

% Population at time t
P = P0 * exp(r * t);

% Plotting the result
figure;
plot(t, P);
xlabel('Time');
ylabel('Population');
title('Exponential Population Growth');
grid on;
```

Outcome

This program will generate a plot showing the exponential growth of the population over time. The x-axis represents time, and the y-axis represents the population size. You should see a curve that starts at the initial population P_0 and grows exponentially as time increases.

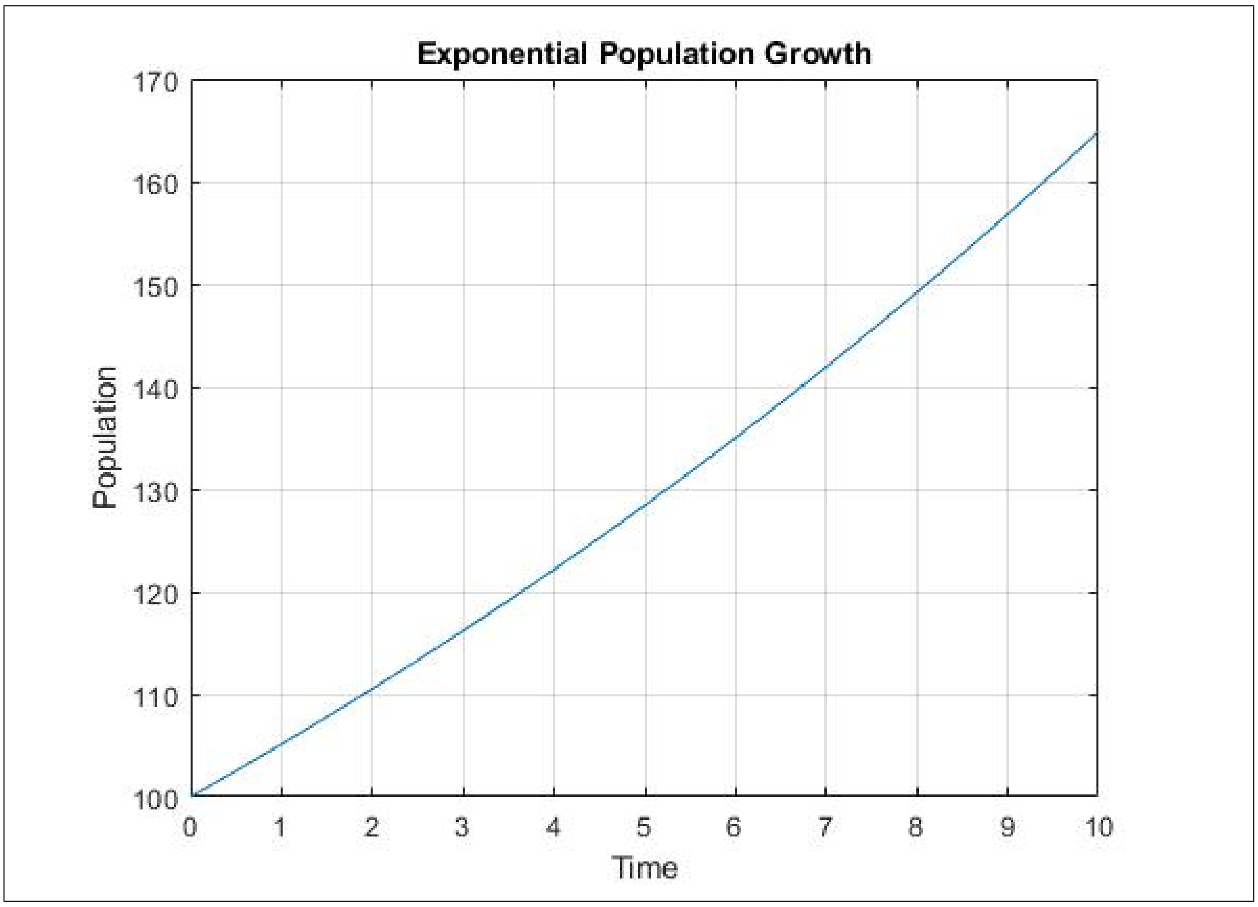
Exercise Problem

Consider an initial population size $P_0 = 150$. Using the exponential growth model, predict the population after 15 years with three different growth rates: $r = 0.03$, $r = 0.06$, and $r = 0.09$. Calculate the population size at $t = 15$ years for each growth rate. Plot the population growth over time for all three growth rates on the same graph. Interpret how the population changes with different growth rates and discuss which growth rate has the most significant impact on long-term population growth.

Practical No. 2

Aim

To model the distance traveled by an object moving at constant velocity.



Problem

Given a constant velocity, find the distance an object travels over time.

Theory

With constant velocity, the distance traveled is given by:

$$d = v \cdot t,$$

where

- d is the distance traveled,
- v is the constant velocity,
- t is the time.

Algorithm

Step 1: Define the constant velocity v,
Step 2: Use the formula to calculate d over time,

Step 3: Calculate and store values for each time step,
Step 4: Plot distance over time.

Program

```
    % Constant Velocity Model

v = 60;             % Constant velocity in km/h
T = 10;             % Total time in hours

% Time vector from 0 to T
time = linspace(0, T, 100);

% Calculate distance at each time point
distance = v * time;

% Plotting the distance traveled
figure;
plot(time, distance, 'r-', 'LineWidth', 2);
title('Constant Velocity Model');
xlabel('Time (hours)');
ylabel('Distance (km)');
grid on
```

Outcome

The output of the program is as follows

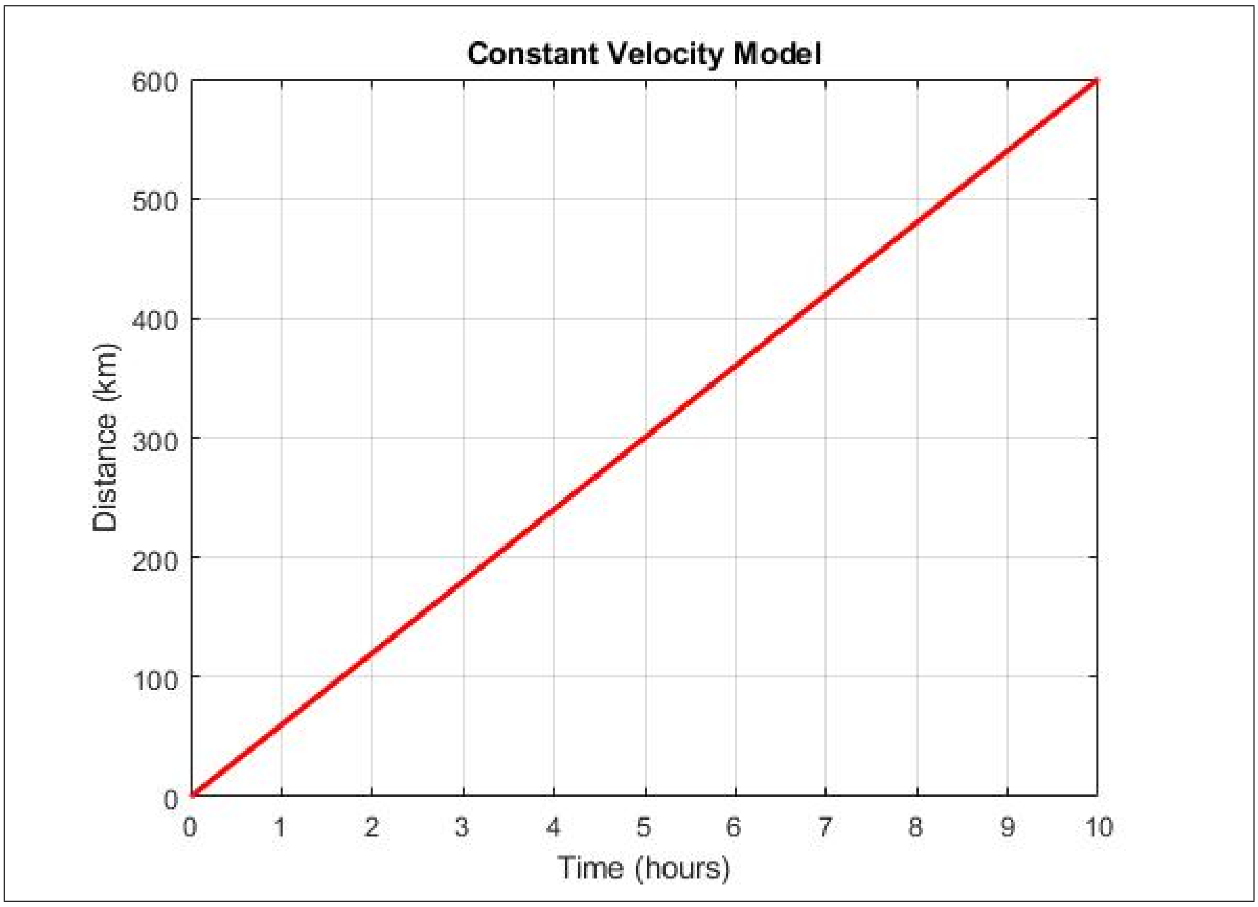
Exercise Problem

An object is moving at a constant velocity. Calculate the distance it travels over 15 seconds at three different constant velocities: $v = 10$ m/s, $v = 20$ m/s, and $v = 30$ m/s. Calculate the distance traveled for each velocity over time $t = 0$ to $t = 15$ seconds. Plot the distance over time for each velocity on the same graph. Interpret how changing velocity affects the slope of the distance-time graph and the overall distance covered.

Practical No. 3

Aim

To model the decay of a quantity over time, such as radioactive decay or the decrease in population with a constant decay rate.



Problem

Given an initial quantity and a constant decay rate, simulate the decay over time.

Theory

For exponential decay, the quantity decreases over time as

$$Q(t) = Q_0 e^{-kt},$$

where

- $Q(t)$ is the quantity at time t
- Q_0 is the initial quantity
- k is the decay constant

Algorithm

Step 1: Define initial quantity Q_0 and the decay constant k,
Step 2: Use the decay formula to calculate $Q(t)$ over time,

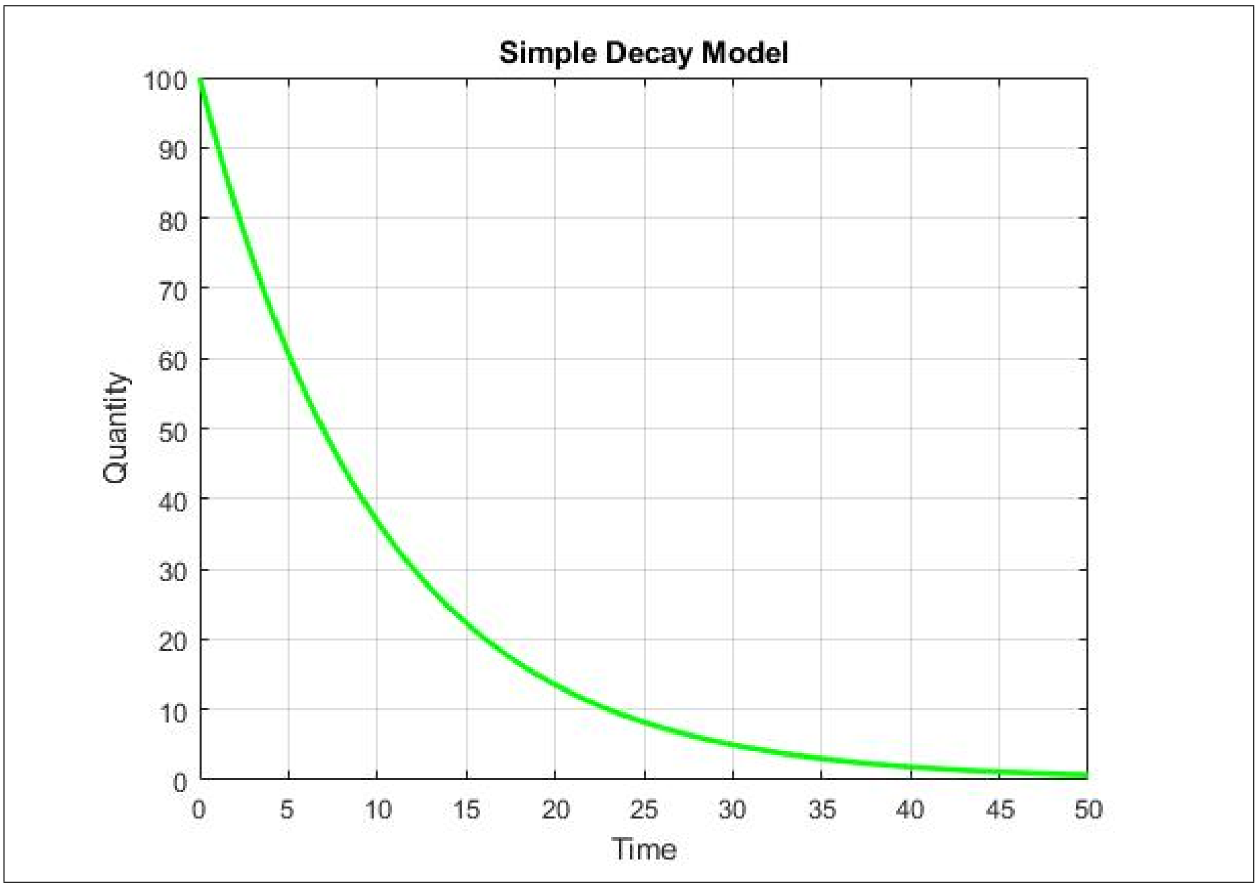
Step 3: Calculate and store the values for each time step,
Step 4: Plot the decay over time.

Program

```
    % Simple Decay Model

Q0 = 100;           % Initial quantity
k = 0.1;            % Decay constant
T = 50;             % Total time period

% Time vector from 0 to T
time = linspace(0, T, 100);

% Calculate quantity at each time point
Q = Q0 * exp(-k * time);

% Plotting the decay
figure;
plot(time, Q, ‘g-’, ‘LineWidth’, 2);
title(‘Simple Decay Model’);
xlabel(‘Time’);
ylabel(‘Quantity’);
grid on
```

Output

The output of the program is as follows



Exercise Problem

Given an initial quantity $Q_0 = 200$, calculate the decay over time for three different decay constants: $k = 0.05$, $k = 0.1$, and $k = 0.2$. Simulate the decay over a time period from $t = 0$ to $t = 20$ and calculate the quantity $Q(t)$ for each decay constant at each time step. Plot the decay curves on the same graph to compare how different values of k affect the decay rate. Interpret the differences between the curves, noting how the value of k influences the decay rate and the time taken for the quantity to decrease significantly.

Practical No. 4

Aim

To model the free fall of an object under gravity and analyze the motion using MATLAB

Problem

Given the initial height and initial velocity of an object, predict its position and velocity over time as it falls under the influence of gravity.

Theory

The motion of a freely falling object can be described by the equations:

$$v(t) = v_0 + gt,$$

$$y(t) = y_0 + v_0 t + \frac{1}{2} g t^2,$$

where

- $y(t)$ is the position at time t
- $v(t)$ is the velocity at time t
- y_0 is the initial height
- v_0 is the initial velocity
- g is the acceleration due to gravity (approximately $9.8\,\mathrm{m/s^2}$)

Algorithm

Step 1: Define the initial height y_0, initial velocity v_0, and the acceleration due to gravity g,
Step 2: Define the period over which the motion is to be calculated,
Step 3: Use the equations of motion to calculate the position and velocity at each time step,
Step 4: Plot the position and velocity over time.

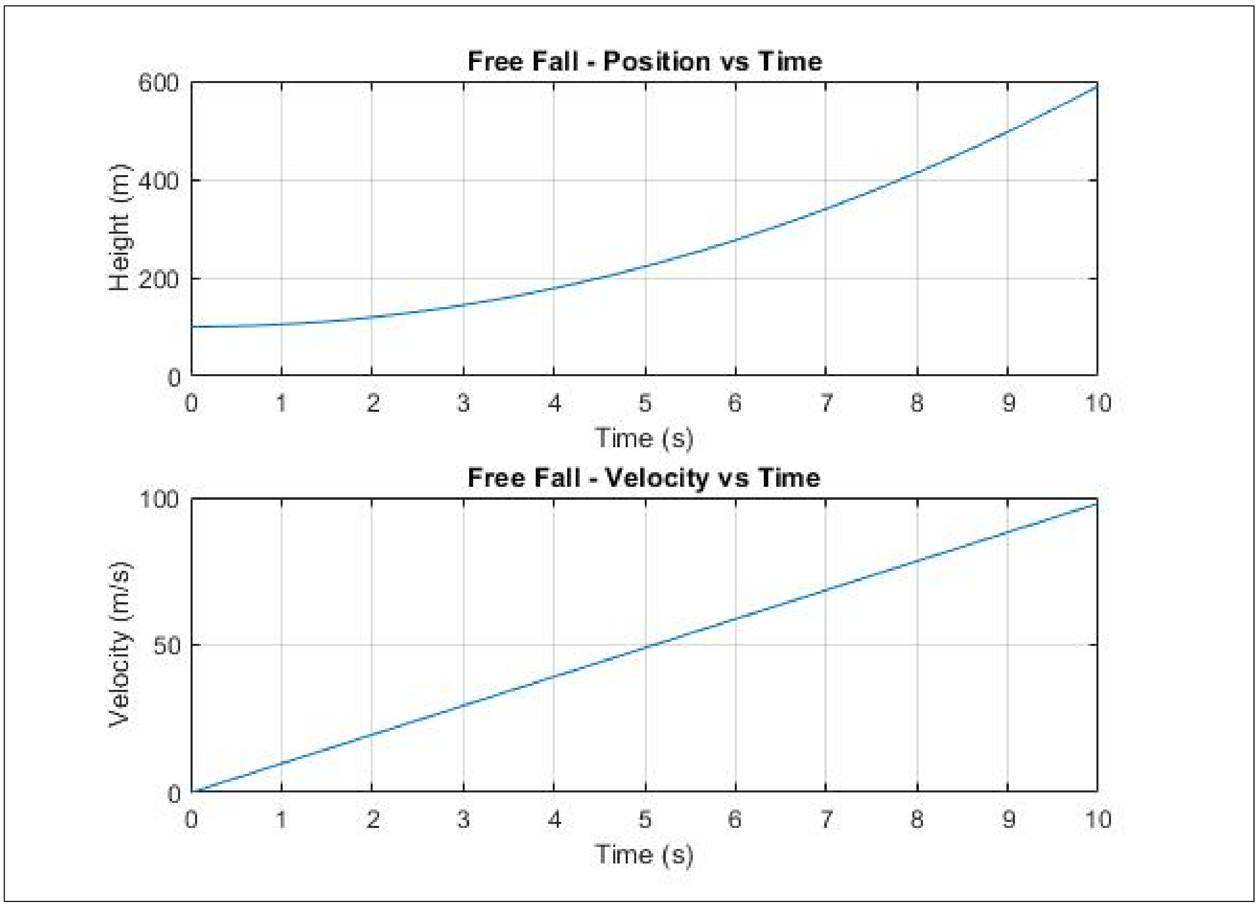
Program

```
    % Parameters
y0 = 100; % Initial height (meters)
v0 = 0; % Initial velocity (m/s)
g = 9.8; % Acceleration due to gravity (m/s^2)
t = 0:0.1:10; % Time vector (seconds)

% Position and velocity at time t
v = v0 + g * t;
y = y0 + v0 * t + 0.5 * g * t.^2;
```

```
% Plotting the results
figure;
subplot(2,1,1);
plot(t, y);
xlabel('Time (s)');
ylabel('Height (m)');
title('Free Fall - Position vs Time');
grid on;

subplot(2,1,2);
plot(t, v);
xlabel('Time (s)');
ylabel('Velocity (m/s)');
title('Free Fall - Velocity vs Time');
grid on
```

Output

The output of the program is as follows



Exercise Problem

An object is dropped from an initial height of $y_0 = 200$ meters under the influence of gravity ($g = 9.8$ m/s). Calculate and plot the position and velocity over 10 seconds for the initial velocities: $v_0 = 0$ m/s; $v_0 = 10$ m/s; $v_0 = -10$ m/s. Calculate the position $y(t)$ and velocity $v(t)$ for each initial velocity over time. Plot the position and velocity for each case on separate graphs to compare the motion. Analyze the effect of different initial velocities on the time taken to reach the ground.

Practical No. 5

Aim

To model the population growth with a carrying capacity, where growth slows as the population approaches this limit.

Problem

Given an initial population, growth rate, and carrying capacity, predict the population over time.

Theory

The logistic growth model accounts for population limits by introducing a carrying capacity K. The differential equation is given by

$$\frac{dP}{dt} = rP\left(1 - \frac{P}{K}\right) \qquad \text{with } P(0) = P_0,$$

where

- $P(t)$ is the population at time t,
- r is the growth rate,
- K is the carrying capacity.

The solution is of the form

$$P(t) = \frac{K}{1 + \left(\frac{K-P_0}{P_0}\right)e^{-rt}},$$

where P_0 is the initial population.

Algorithm

Step 1: Define the initial population P_0, growth rate r, and carrying capacity K,
Step 2: Set up a time vector over which to evaluate the model,
Step 3: Calculate $P(t)$ for each time point using the logistic formula,
Step 4: Plot $P(t)$ against time to visualize logistic growth.

Program

```
    % Logistic Growth Model

P0 = 50;              % Initial population
r = 0.1;              % Growth rate (10%)
K = 500;              % Carrying capacity
T = 50;               % Total time period (years)
% Time vector from 0 to T
time = linspace(0, T, 100);
% Calculate population at each time point using logistic formula
P = K ./ (1 + ((K - P0) / P0) * exp(-r * time));
% Plotting the logistic growth
figure;
plot(time, P, ‘g-’, ‘LineWidth’, 2);
title(‘Logistic Growth Model’);
xlabel(‘Time (years)’);
ylabel(‘Population’);
grid on;
```

Output

This plot will show an S-shaped (sigmoid) curve, where the population initially grows quickly, then slows down as it approaches the carrying capacity K.

Exercise Problem

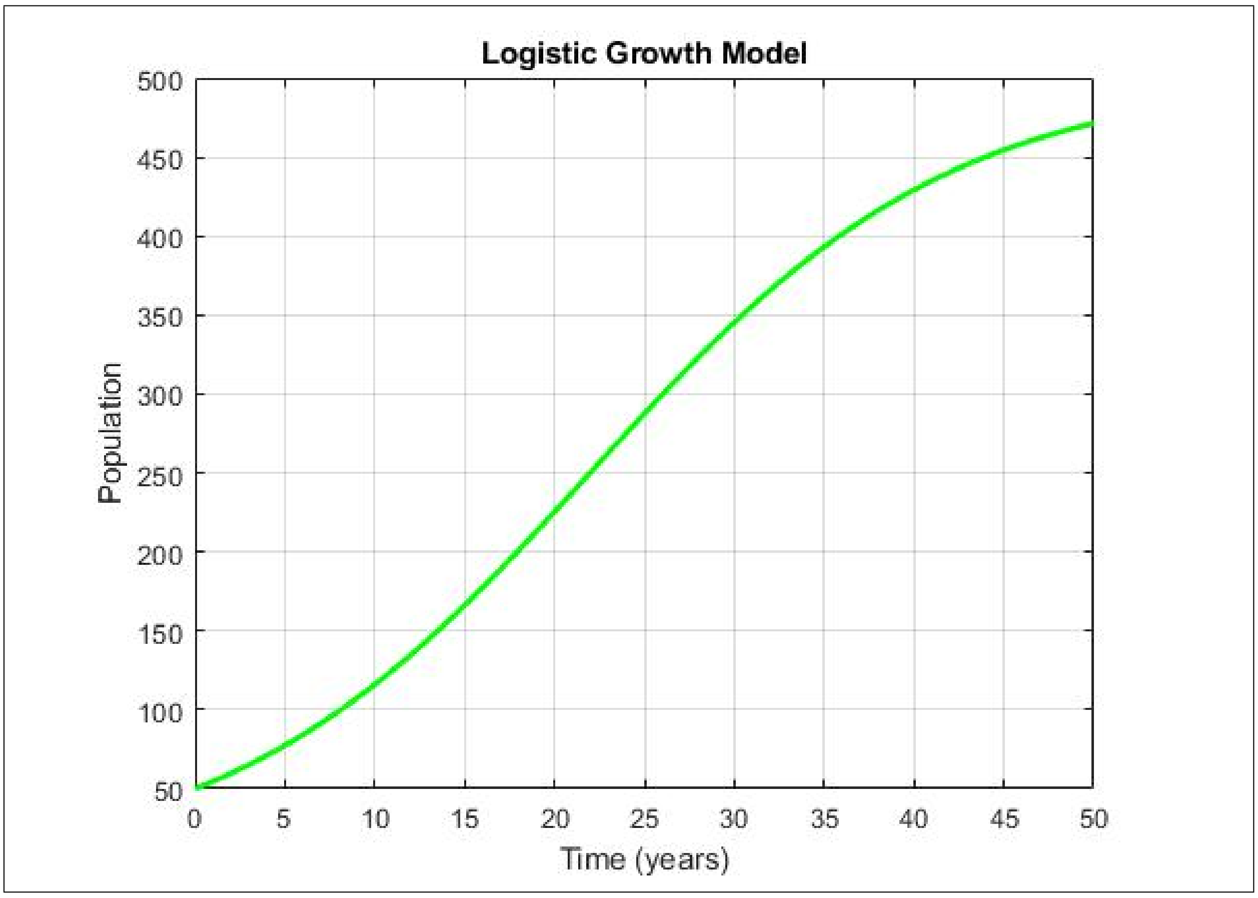
An environment has a population with an initial size of $P_0 = 50$ and a growth rate $r = 0.08$. Predict the population over time for three different carrying capacities: $K = 300$, $K = 500$, and $K = 1000$.

For each carrying capacity, calculate and plot the population over 50 years and observe the differences in growth behavior.

Practical No. 6

Aim

To model linear and nonlinear growth and decay using first-order ordinary differential equations (ODEs) and analyze the solutions using MATLAB.



Problem

Given an initial quantity, we aim to model its growth or decay over time using both linear and nonlinear equations. We will compare the results and visualize the solutions.

Theory

Linear Growth and Decay

The linear growth/decay model is described by the equation:

$$\frac{dQ}{dt} = kQ, \qquad \text{with} \quad Q(0) \ = \ Q_0$$

where

- $Q(t)$ is the quantity at time t
- k is the constant rate of growth (if $k > 0$) or decay (if $k < 0$)

The solution to this differential equation is

$$Q(t) = Q_0 e^{kt},$$

where Q_0 is the initial quantity.

Nonlinear Growth and Decay

One common nonlinear model is the logistic growth model

$$\frac{dQ}{dt} = rQ\left(1 - \frac{Q}{K}\right), \qquad \text{with} \quad Q(0) \ = \ Q_0,$$

where

- $Q(t)$ is the quantity at time t,
- r is the intrinsic growth rate,
- K is the carrying capacity.

The solution to this differential equation is

$$Q(t) = \frac{KQ_0e^{rt}}{K + Q_0(e^{rt} - 1)},$$

where Q_0 is the initial quantity.

Algorithm

Step 1: Define the initial quantity Q_0 and parameters k, r, and K,
Step 2: Define the time period over which the quantity is to be calculated,
Step 3: Use the respective equations to calculate the quantity at each time step for linear and nonlinear models,
Step 4: Plot the quantities over time for comparison.

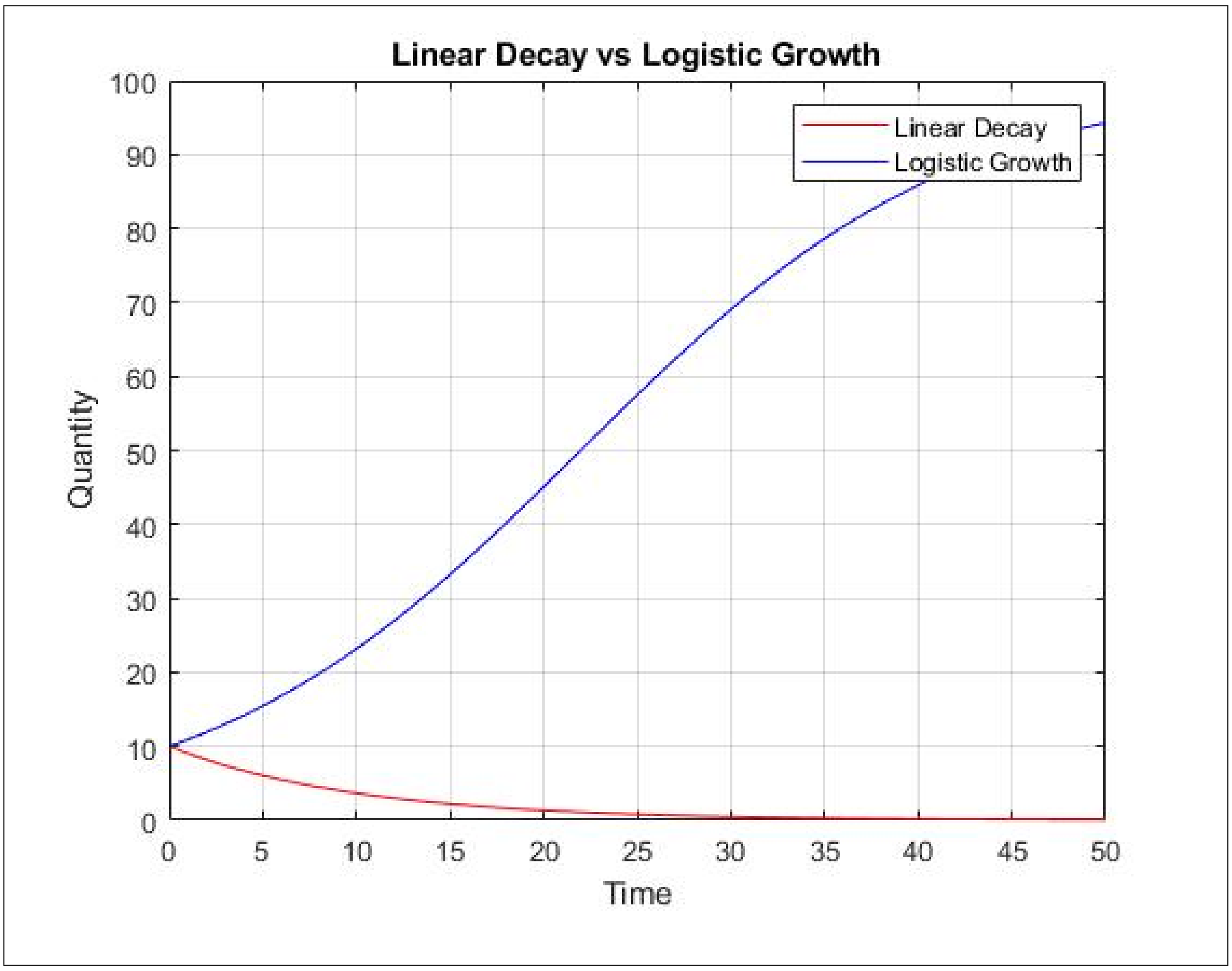
Program

```
% Parameters
Q0 = 10; % Initial quantity
k = -0.1; % Decay constant for linear model
r = 0.1; % Growth rate for nonlinear model
K = 100; % Carrying capacity for nonlinear model
t = 0:0.1:50; % Time vector

% Linear model (Exponential decay)
Q_linear = Q0 * exp(k * t);

% Nonlinear model (Logistic growth)
Q_nonlinear = K * Q0 * exp(r * t) ./ (K + Q0 * (exp(r * t) - 1));

% Plotting the results
```

```
figure;
plot(t, Q_linear, ‘r’, ’DisplayName’, ‘Linear Decay’);
hold on
plot(t, Q_nonlinear, ‘b’, ‘DisplayName’, ‘Logistic Growth’);
xlabel(‘Time’);
ylabel(‘Quantity’);
title(‘Linear Decay vs Logistic Growth’);
legend;
grid on;
```

Output

MATLAB programs will generate a plot showing the linear decay and logistic growth of the quantity over time. The x-axis represents time, and the y-axis represents the quantity. The linear decay will show an exponential decrease, while the logistic growth will show an S-shaped curve that levels off as it approaches the carrying capacity K.



Exercise Problem

Suppose you are tracking two quantities:

Quantity A: Represents the decay of a chemical substance over time with an initial amount of $Q_0 = 50$ and a decay rate $k = -0.03$.

Quantity B: Represents the growth of a biological population in an environment with limited resources, starting with an initial population $Q_0 = 50$, growth rate $r = 0.07$, and carrying capacity $K = 200$.

For each quantity: calculate and plot the values over 80 units of time. Also, analyze the different behaviors shown in the graph.

Practical No. 7

Aim

To model population growth with a carrying capacity, where growth slows as the population approaches this limit.

Problem

Given an initial population, growth rate, and carrying capacity, predict the population over time.

Theory

The logistic growth model accounts for population limits by introducing a carrying capacity K. The differential equation is

$$\frac{dP}{dt} = rP\left(1 - \frac{P}{K}\right), \qquad \text{with} \quad P(0) = P_0,$$

where

- $P(t)$ is the population at time t,
- r is the growth rate,
- K is the carrying capacity.

The solution is of the form

$$P(t) = \frac{K}{1 + \left(\frac{K-P_0}{P_0}\right)e^{-rt}},$$

where P_0 is the initial population.

Algorithm

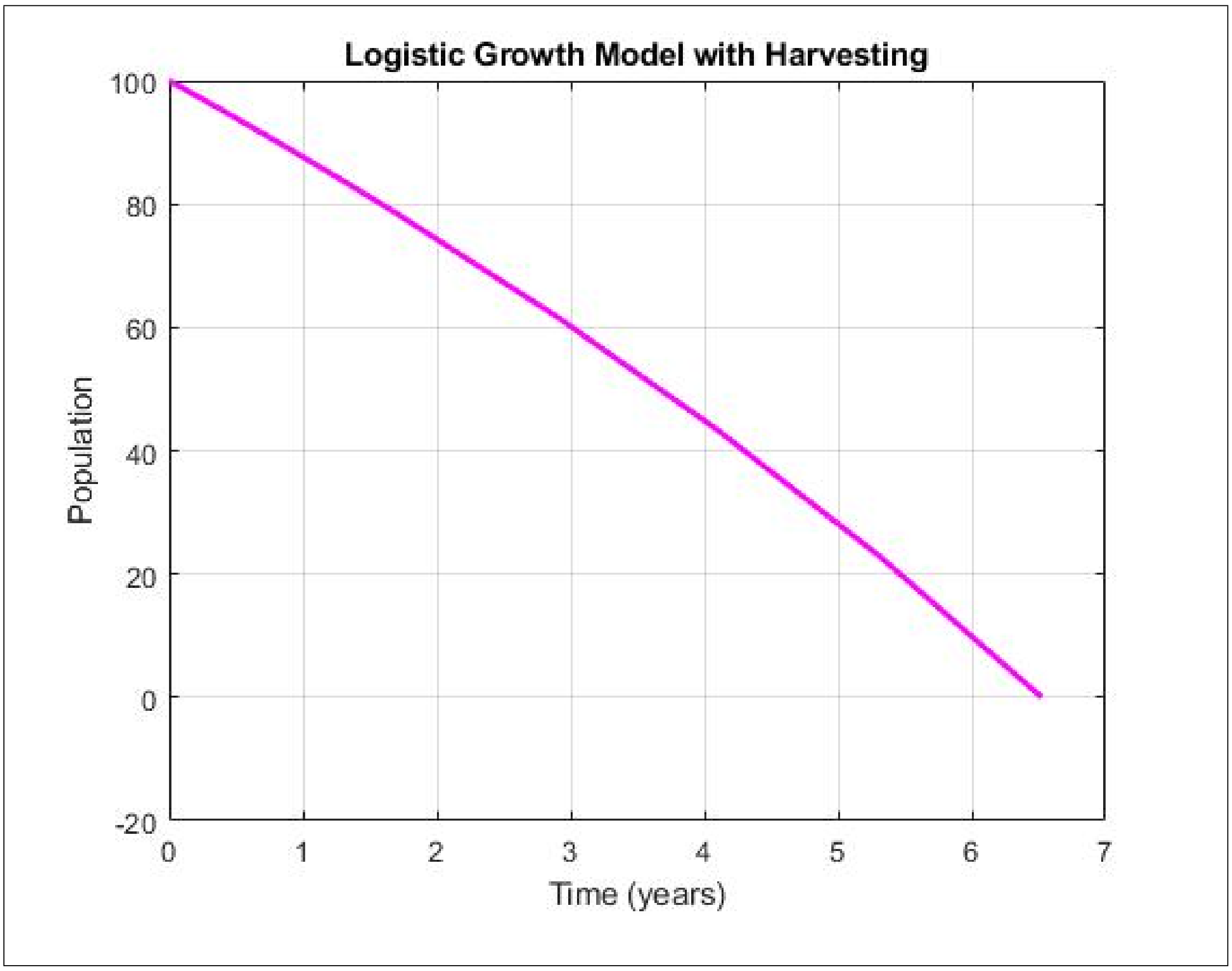
Step 1: Define the initial population P_0, growth rate r, and carrying capacity K,
Step 2: Set up a time vector over which to evaluate the model,
Step 3: Calculate $P(t)$ for each time point using the logistic formula,
Step 4: Plot $P(t)$ against time to visualize logistic growth.

Program

```
% Parameters for the Logistic Growth Model with Harvesting
P0 = 100;            % Initial population
r = 0.1;             % Growth rate (10%)
K = 500;             % Carrying capacity
h = 20;              % Harvesting rate
T = 50;              % Total time period (years)

% Define the differential equation
dPdt = @(t, P) r * P * (1 - P / K) - h;

% Define options to stop integration when population goes below zero
options = odeset('Events', @(t, P) eventFcn(P));

% Solve using ode45 with event handling
[time, P] = ode45(dPdt, [0 T], P0, options);

% Plotting the population growth with harvesting
figure;
plot(time, P, ‘m-’, ‘LineWidth’, 2);
title(‘Logistic Growth Model with Harvesting’);
xlabel(‘Time (years)’);
ylabel(‘Population’);
grid on;

% Event function to stop integration when population reaches zero
function [value, isterminal, direction] = eventFcn(P)
    value = P;          % Condition to check (population reaches zero)
    isterminal = 1;     % Stop the integration
    direction = -1;     % Detect when P is decreasing through zero
end
```

Output

The plot will show an S-shaped (sigmoid) curve, where the population initially grows quickly, then slows down as it approaches the carrying capacity K.



Exercise Problem

Given a population with an initial size of $P_0 = 100$, growth rate $r = 0.1$, and carrying capacity $K = 500$, explore how varying harvesting rates impact the population over time. Find different values for the harvesting rate h; $h = 10$, $h = 20$, $h = 30$. For each harvesting rate, plot the population over 50 years and analyze the behavior of the population.

Practical No. 8

Aim

To model the spread of an infectious disease using the SIR model and analyze the solution using MATLAB.

Problem

Write down a mathematical model for SIR model.

Theory

The SIR model describes the dynamics of infectious diseases as follows

$$\frac{dS}{dt} = -\beta SI,$$

$$\frac{dI}{dt} = \beta SI - \gamma I,$$

$$\frac{dR}{dt} = \gamma I,$$

with

$$S(0) = S_0, \quad I(0) = I_0, \quad R(0) = R_0,$$

where

- $S(t)$ is the number of susceptible individuals at time t,
- $I(t)$ is the number of infectious individuals at time t,
- $R(t)$ is the number of recovered individuals at time t,
- β is the infection rate,
- γ is the recovery rate.

Algorithm

Step 1: Define the parameters β, γ and the initial numbers S_0, I_0, R_0,
Step 2: Define the time vector over which the system will be simulated,
Step 3: Use the SIR model equations to define the system,
Step 4: Use MATLAB's ODE solver (ode45) to solve the system of equations,
Step 5: Plot the numbers of susceptible, infectious, and recovered individuals over time.

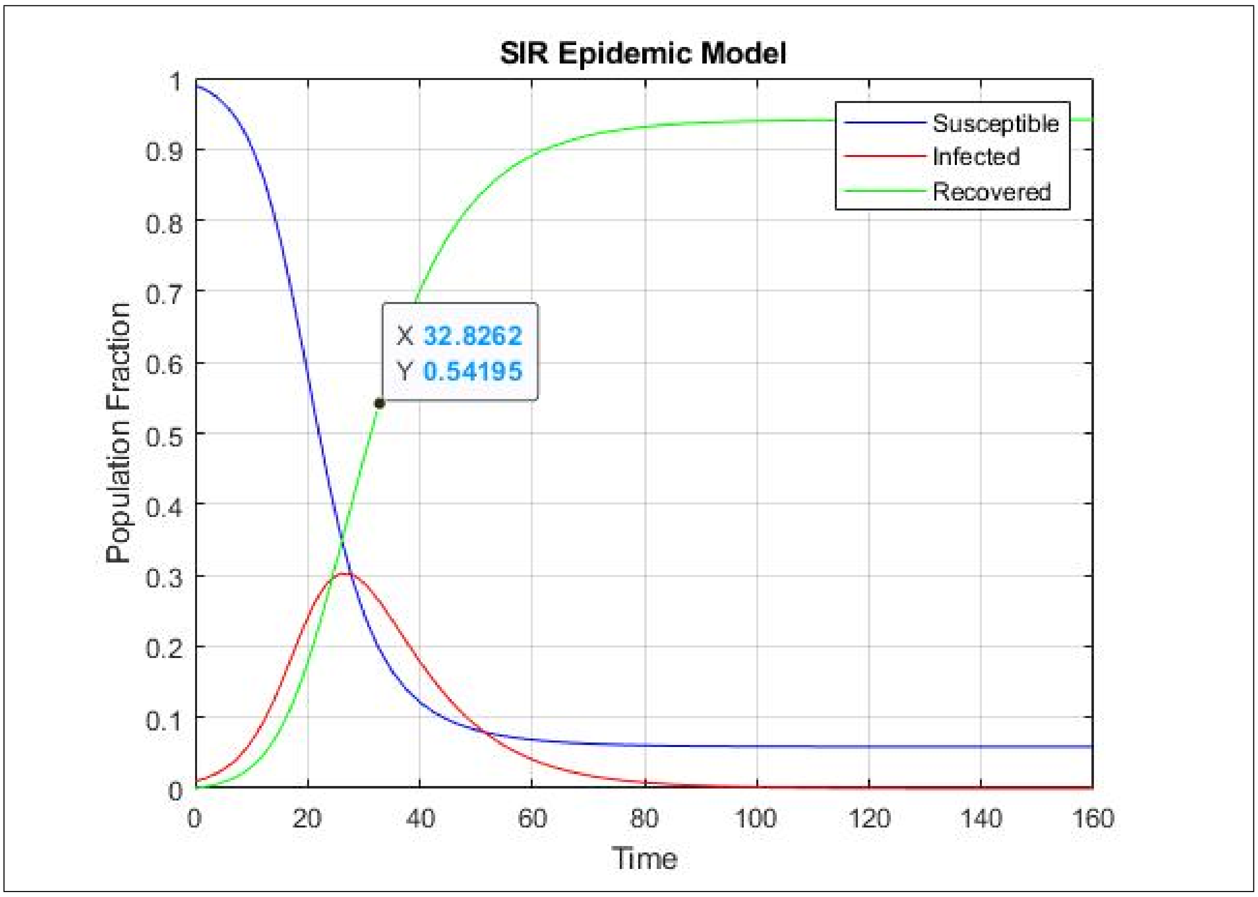
Program

```
% Parameters
beta = 0.3;    % Infection rate
gamma = 0.1;   % Recovery rate
initial_values = [0.99, 0.01, 0];  % Initial S, I, R values
T = 160;
```

```
sir_model = @(t, Y) [-beta * Y(1) * Y(2); beta * Y(1) * Y(2) - gamma * Y(2);
gamma * Y(2)];

[time, SIR] = ode45(sir_model, [0 T], initial_values);

plot(time, SIR(:,1), 'b', time, SIR(:,2), 'r', time, SIR(:,3), 'g');
title('SIR Epidemic Model');
xlabel('Time');
ylabel('Population Fraction');
legend('Susceptible', 'Infected', 'Recovered');
grid on;
```

Output

Plot showing the progression of susceptible, infected, and recovered populations, with the infection peaking and then declining.



Exercise Problem

Modeling the spread of an infectious disease in a population using the SIR (Susceptible, Infected, Recovered) model. Consider the following parameters: $\beta = 0.4$ (the rate at which individuals are infected by others), $\gamma = 0.1$ (the rate at which infected individuals recover) with initial population conditions: $S_0 = 0.95$, $I_0 = 0.05$, $R_0 = 0$.

Practical No. 9

Aim

To model the interaction between predator-prey populations using a system of ordinary differential equations and analyze the stability of the system using MATLAB.

Problem

Write down a mathematical model for Lotka-Volterra predator-prey system and analyze the system's stability.

Theory

The Lotka-Volterra predator-prey system is described as the system of following differential equations as follows

$$\frac{dx}{dt} = \alpha x - \beta xy,$$

$$\frac{dy}{dt} = \delta xy - \gamma y,$$

with initial values

$$x(0) \;=\; x_0, \;\; y(0) \;=\; y_0,$$

where

- $x(t)$ is the prey population at time t,
- $y(t)$ is the predator population at time t,
- α is the growth rate of prey in the absence of predators,
- β is the predation rate,
- γ is the natural death rate of predators,
- δ represents the activation rate of predators.

Algorithm

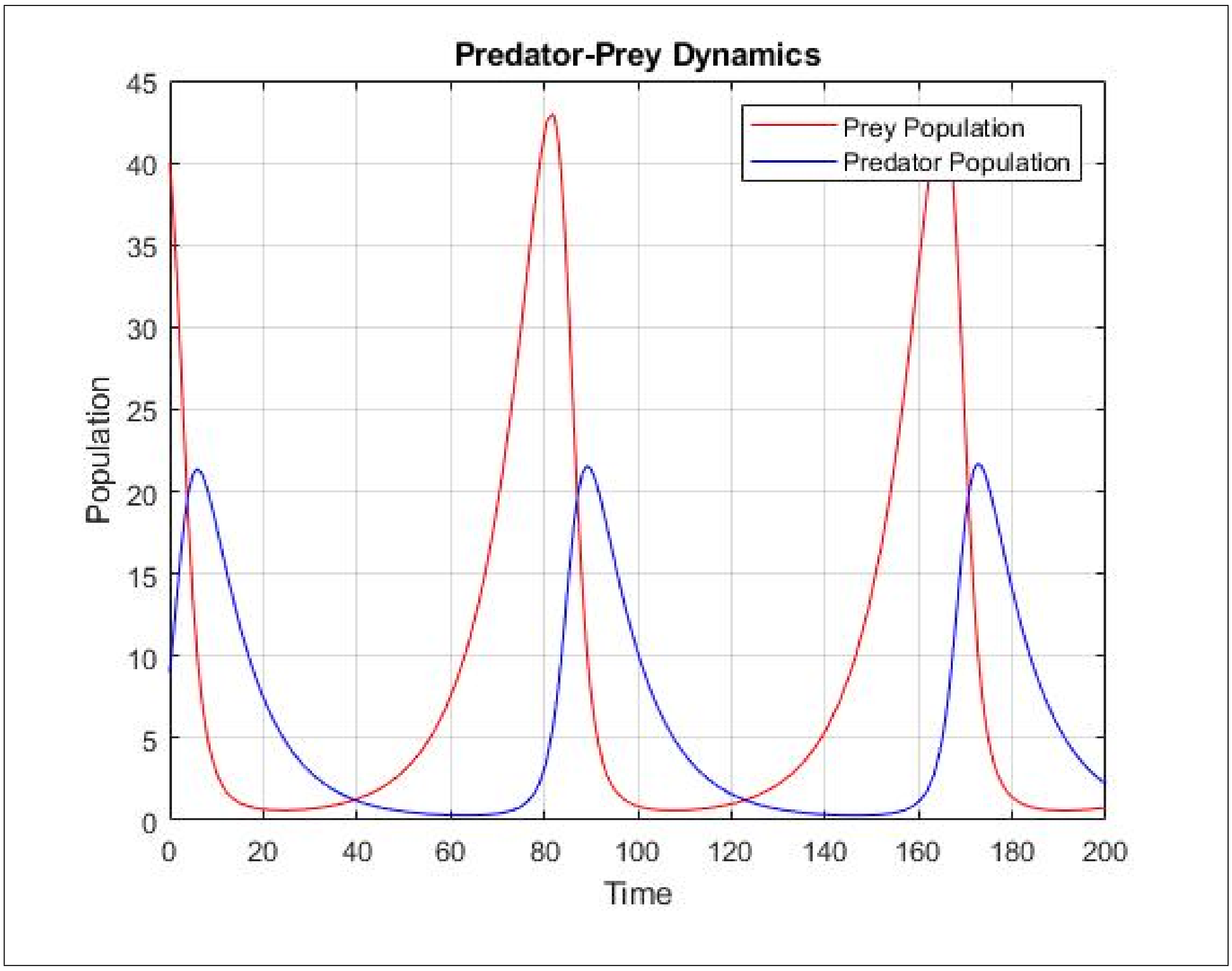
Step 1: Define the parameters α, β, γ, δ with initial values x_0, y_0,
Step 2: Define the time vector over which the system will be simulated,
Step 3: Use the Lotka-Volterra equations to define the system,
Step 4: Use MATLAB's ODE solver (ode45) to solve the system of equations,
Step 5: Plot the population of prey and predators over time.

Program

```
% Parameters
alpha = 0.1; % Prey growth rate
beta = 0.02; % Predation rate
gamma = 0.1; % Predator death rate
delta = 0.01; % Predator reproduction rate

% Initial conditions
x0 = 40; % Initial prey population
y0 = 9; % Initial predator population

% Time vector
tspan = [0 200];

% Lotka-Volterra equations
lotka_volterra = @(t, xy) [alpha * xy(1) - beta * xy(1) * xy(2);
                           delta * xy(1) * xy(2) - gamma * xy(2)];

% Solve the system of equations
[t, xy] = ode45(lotka_volterra, tspan, [x0, y0]);

% Plotting the results
figure;
plot(t, xy(:,1), ‘r’, ‘DisplayName’, ‘Prey Population’);
hold on;
plot(t, xy(:,2), ‘b’, ‘DisplayName’, ‘Predator Population’);
xlabel(‘Time’);
ylabel(‘Population’);
title(‘Predator-Prey Dynamics’);
legend;
grid on;
```

Output

MATLAB programs will generate a plot showing the populations of prey and predators over time. The x-axis represents time, and the y-axis represents the population size. The plot will show the oscillatory behavior typical of predator-prey dynamics, where the populations of prey and predators rise and fall periodically.



Exercise Problem

You are tasked with modeling the interaction between predator and prey populations over time using the Lotka-Volterra equations. Given the initial populations of prey and predators, simulate their interaction and analyze the stability of the system for the parameters: prey growth rate (α) = 0.1, predation rate coefficient (β) = 0.02, predator death rate (γ) = 0.1, predator reproduction rate (δ) = 0.01, with initial values $x_0 = 50$ and $y_0 = 10$.

Practical No. 10

Aim

To model the initial spread of an infection in a population where no one is immune.

Problem

Given a small initial infection rate and transmission rate, predict the number of infected individuals over time.

Theory

The basic infection model is defined as follows

$$\frac{dI}{dt} = \beta I \left(1 - \frac{I}{N}\right), \qquad \text{with} \quad I(0) = I_0,$$

where

- $I(t)$ is the number of infected individuals at time t,
- β is the transmission rate,
- N is the population.

Algorithm

Step 1: Define I_0, β, N,
Step 2: Set up a time vector,
Step 3: Use ode45 in MATLAB,
Step 4: Plot the spread of infection.

Program

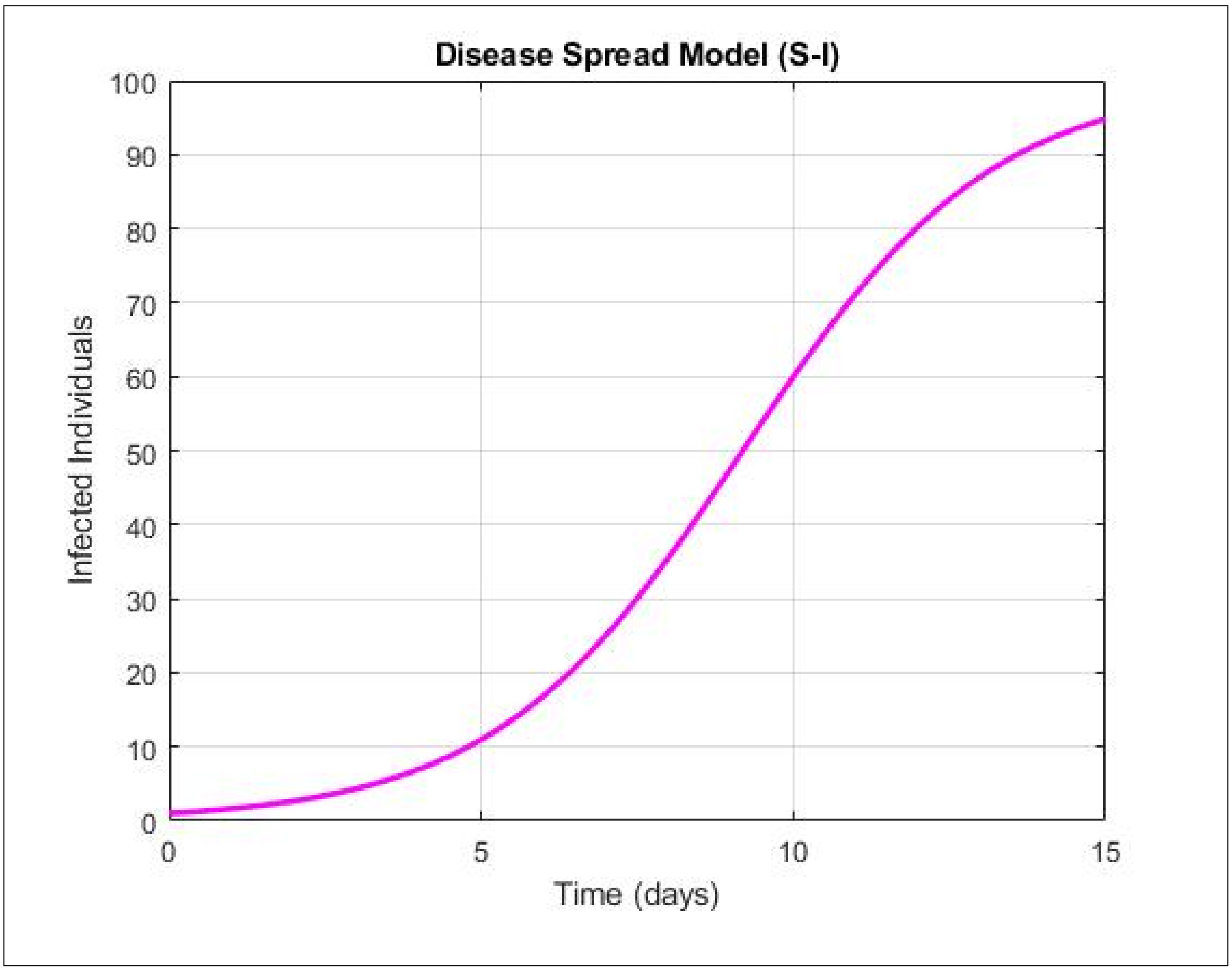
```
 I0 = 1;         % Initial infected population
beta = 0.5;     % Transmission rate
N = 100;        % Total population
T = 15;         % Time span

dIdt = @(t, I) beta * I * (1 - I / N);

[time, I] = ode45(dIdt, [0 T], I0);

plot(time, I, ‘m-’, ‘LineWidth’, 2);
title(‘Disease Spread Model (S-I)’);
xlabel(‘Time (days)’);
ylabel(‘Infected Individuals’);
grid on;
```

Output

An initial rapid increase in infection, slowing as it approaches the total population N.

Exercise Problem

You are tasked with modeling the initial spread of an infection in a population using the given infection model. The infection starts with a small number of initially infected individuals and spreads over time with parameters: initial number of infected individuals (I_0) = 2, transmission rate (β) = 0.3, total population (N) = 200, simulation time (T) = 20 days. Using MATLAB, simulate the spread of the infection over time with the given parameters. Plot the number of infected individuals versus time. What do you observe about the infection spread in the early stages (first 5 days)? Does the rate of infection growth appear exponential or logistic?



Practical No. 11

Aim

Calculate the growth of an investment with annual compounding interest over discrete intervals.

Problem

Given an initial investment, annual interest rate, and number of years, calculate the investment value at the end of each year.

Theory

The investment grows by a fixed percentage each year is given by

$$I_{n+1} = I_n(1 + r), \qquad \text{with} \quad I(0) = I_0,$$

where r is the annual interest rate.

Algorithm

Step 1: Define I_0, r and the number of years N,
Step 2: Use a loop to calculate the investment value at each interval,
Step 3: Plot the discrete growth over time.

Program

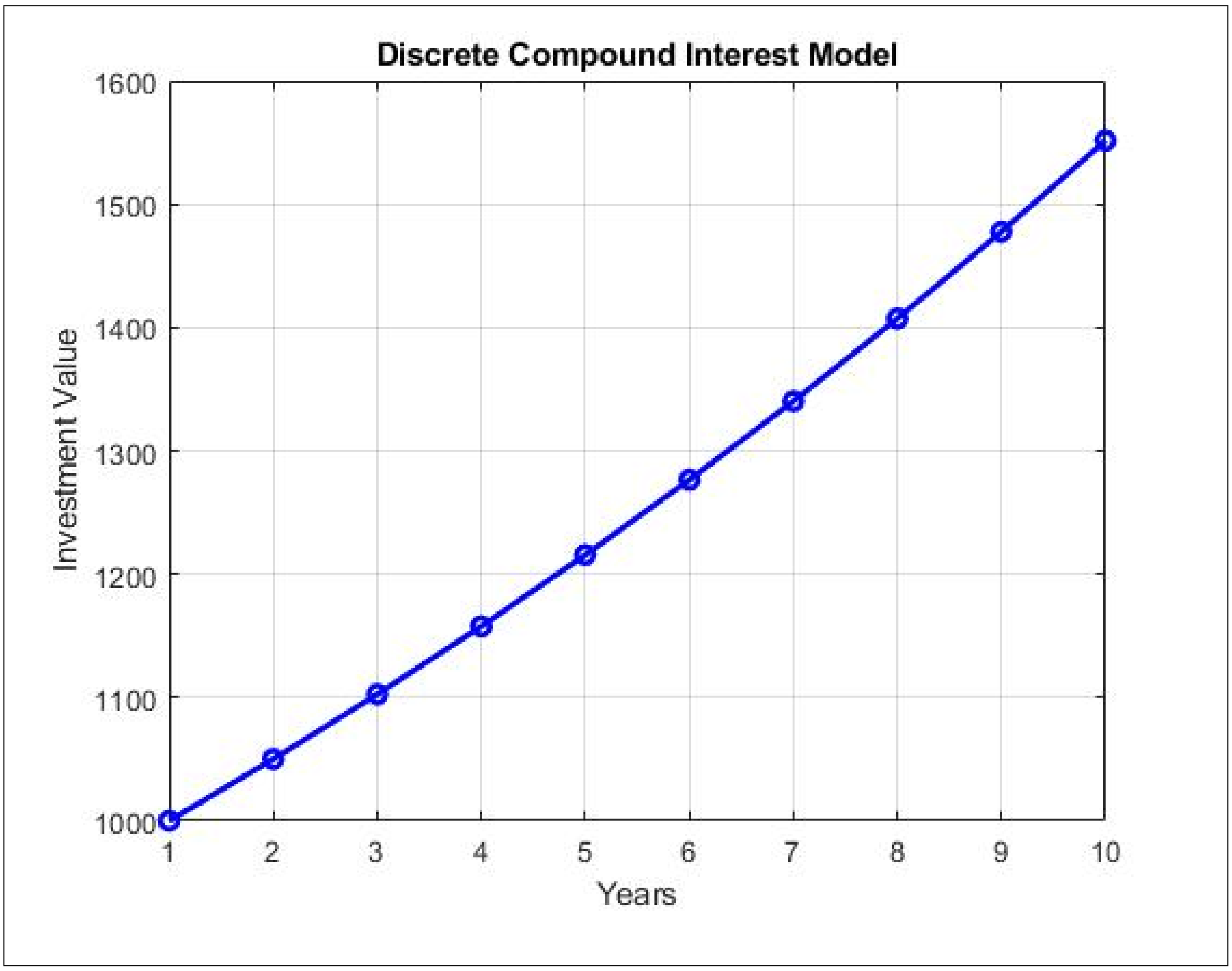
```
 I0 = 1000;    % Initial investment
r = 0.05;     % Interest rate (5%)
N = 10;       % Number of years

I = zeros(1, N);
I(1) = I0;

for n = 2:N
    I(n) = I(n-1) * (1 + r);
end

plot(1:N, I, ‘bo-’, ‘LineWidth’, 2);
title(‘Discrete Compound Interest Model’);
xlabel(‘Years’);
ylabel(‘Investment Value’);
grid on;
```

Output

A stepwise plot showing the investment growing each year, with each step higher due to the compounding effect.

Exercise Problem

You are working with a financial advisor to calculate the growth of an investment over a period of time. The initial parameters for the investment are as follows: initial investment $(I_0) = 1500$, annual interest rate $(r) = 4\%$ (0.04), number of years $(N) = 20$. Use the formula for annual compounding interest to compute the value of the investment at the end of each year over 20 years. Write a MATLAB program to calculate and store the investment values for each year. Plot the investment growth over the 20 years as a stepwise graph.



Practical No. 12

Aim

To analyze the stability of the equilibrium point for the predator-prey system using Jacobian analysis.

Problem

Given a predator-prey model, determine the stability of the systems equilibrium points.

Theory

The Jacobian matrix for the system

$$\frac{dx}{dt} = \alpha x - \beta xy$$

$$\frac{dy}{dt} = \delta xy - \gamma y$$

is

$$J = \begin{bmatrix} \alpha - \beta y & -\beta x \\ \delta y & \delta x - \gamma \end{bmatrix}$$

Evaluating this matrix at equilibrium points allows us to analyze stability.

Algorithm

Step 1: Find the equilibrium points,
Step 2: Calculate the Jacobian matrix,
Step 3: Determine eigenvalues of the Jacobian at the equilibrium point,
Step 4: Analyze stability based on the sign of the eigenvalues.

Program

```
syms x y alpha beta delta gamma;
J = [alpha - beta * y, -beta * x; delta * y, delta * x - gamma];
equilibrium = [gamma/delta, alpha/beta];
J_eq = subs(J, {x, y}, equilibrium);
eigenvalues = eig(J_eq);

disp('Jacobian at equilibrium:');
disp(J_eq);
disp('Eigenvalues:');
disp(eigenvalues);
```

Output

```
 Jacobian at equilibrium:

[                  0, -(beta*gamma)/delta]
[ (alpha*delta)/beta,                   0]

Eigenvalues:

  (-alpha)^(1/2)*gamma^(1/2)
 -(-alpha)^(1/2)*gamma^(1/2)
```

The Jacobian matrix and eigenvalues at the equilibrium point, helping determine stability (eigenvalues with negative real parts indicate stability).

Exercise Problem

The Lotka-Volterra predator-prey system is given by

$$\frac{dx}{dt} = \alpha x - \beta xy,$$
$$\frac{dy}{dt} = \delta xy - \gamma y,$$

where $x(t)$ represents the prey population and $y(t)$ is the predator population at time t, α represents the growth rate of the prey, β is the predation coefficient, γ is the natural death rate of the predators and δ designates the reproduction rate of predators.
Solve for the equilibrium points of the predator-prey system by setting $\frac{dx}{dt} = 0$ and $\frac{dy}{dt} = 0$. Express the equilibrium points in terms of the parameters α, β, γ, and δ.

Practical No. 13

Aim

The aim of the model is to the relationship between supply and demand in an economic market.

Problem

We are using a system of ordinary differential equations (ODEs) to model the interaction between supply S and demand D, with the assumption that the rate of change of supply and demand are functions of the price P.

Theory

The supply S and demand D rates can be expressed by the following system of differential equations:

$$\frac{dS}{dt} = a(P - P_0),$$
$$\frac{dD}{dt} = -b(P - P_0),$$

with

$$S(0) = S_0, \quad D(0) = D_0,$$

where

- P is the price,
- P_0 is the equilibrium price,

- a and b are positive constants representing sensitivity rates of supply and demand to price changes.

The supply increases if $P > P_0$ (price above equilibrium), while demand decreases. Conversely, if $P < P_0$, supply decreases, and demand increases as the market tries to reach equilibrium.

Algorithm

Step 1: Define constants a, b, and P_0,
Step 2: Set initial values for supply S and demand D,
Step 3: Define differential equations. Use ode45 in MATLAB,
Step 4: Solve and plot.

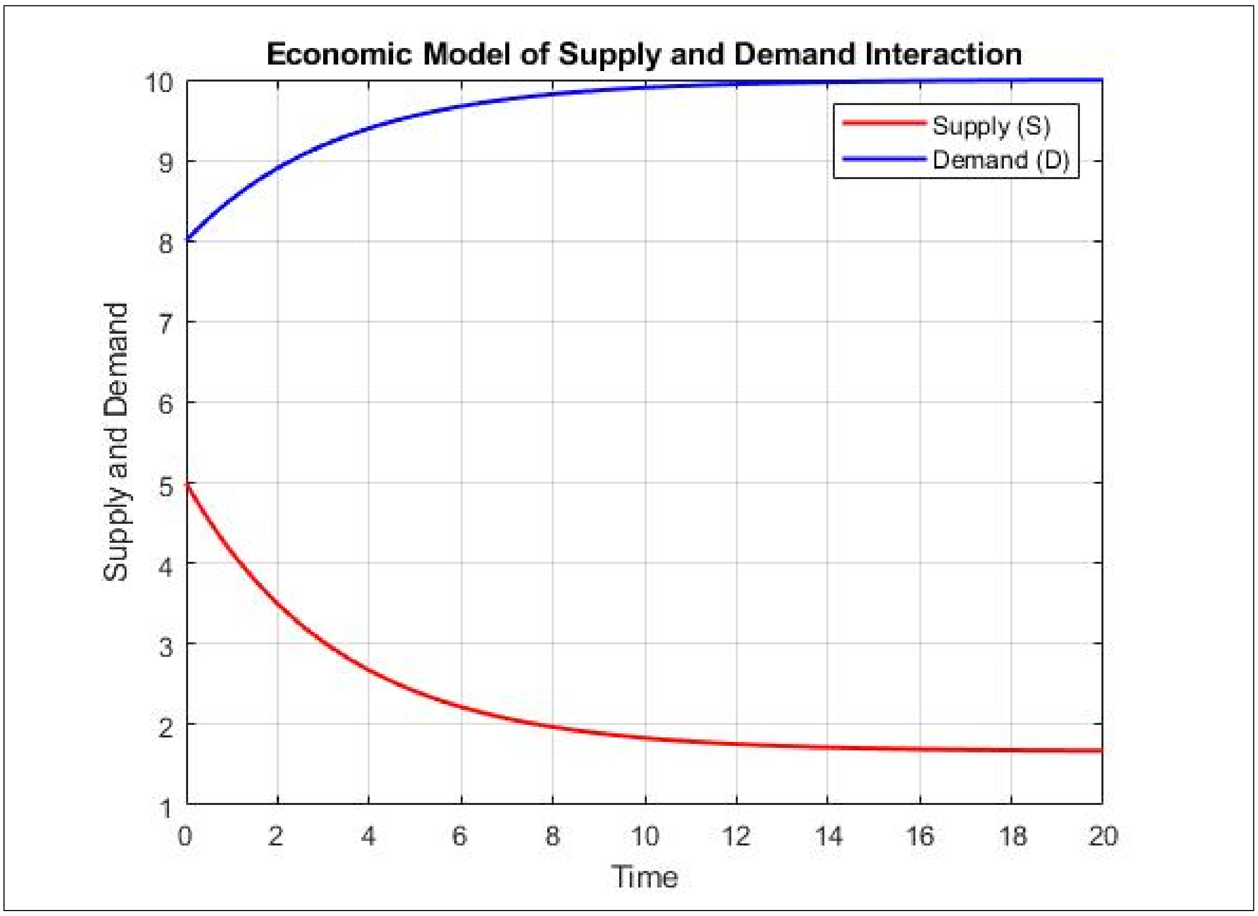
Program

```
% Economic Model of Supply and Demand

% Define constants
a = 0.5;     % Sensitivity of supply to price difference
b = 0.3;     % Sensitivity of demand to price difference
P0 = 10;     % Equilibrium price

% Set initial conditions for supply and demand
S0 = 5;      % Initial supply
D0 = 8;      % Initial demand
initial_conditions = [S0, D0];

% Define time span for the simulation
tspan = [0 20];

% Define the system of ODEs
% dS/dt = a * (P - P0)
% dD/dt = -b * (P - P0)
ode_system = @(t, y) [a * (y(2) - P0); -b * (y(2) - P0)];

% Solve the system of ODEs
[t, sol] = ode45(ode_system, tspan, initial_conditions);

% Extract solutions for S and D
S = sol(:, 1);
D = sol(:, 2);

% Plot the results
```

```
figure;
plot(t, S, 'r-', 'LineWidth', 1.5); hold on;
plot(t, D, 'b-', 'LineWidth', 1.5);
xlabel('Time');
ylabel('Supply and Demand');
legend('Supply (S)', 'Demand (D)');
title('Economic Model of Supply and Demand Interaction');
grid on;
```

Output

The plot generated by the code will show the interaction between supply and demand over time as the system adjusts to reach equilibrium.



Exercise Problem

Write down the system of differential equations representing the supply and demand model with initial values of supply $S_0 = 6$ and demand $D_0 = 10$, and the parameters $a = 0.4$, $b = 0.6$, and $P_0 = 12$.

Practical No. 14

Aim

To model two species competing for the same resources and analyze their interaction over time.

Problem

Consider two competing species x and y, in the same environment. Model how the population of each species changes over time and determine the conditions for coexistence or dominance of one species.

Theory

The competing species model can be described by

$$\frac{dx}{dt} = r_x x \left(1 - \frac{x + \alpha y}{K_x}\right),$$

$$\frac{dy}{dt} = r_y y \left(1 - \frac{y + \beta x}{K_y}\right),$$

with

$$x(0) = x_0, \quad y(0) = y_0,$$

where

- r_x and r_y are the growth rates of prey and predator species respectively,
- K_x and K_y are carrying capacities of x and y respectively,
- α and β are competition coefficients.

Algorithm

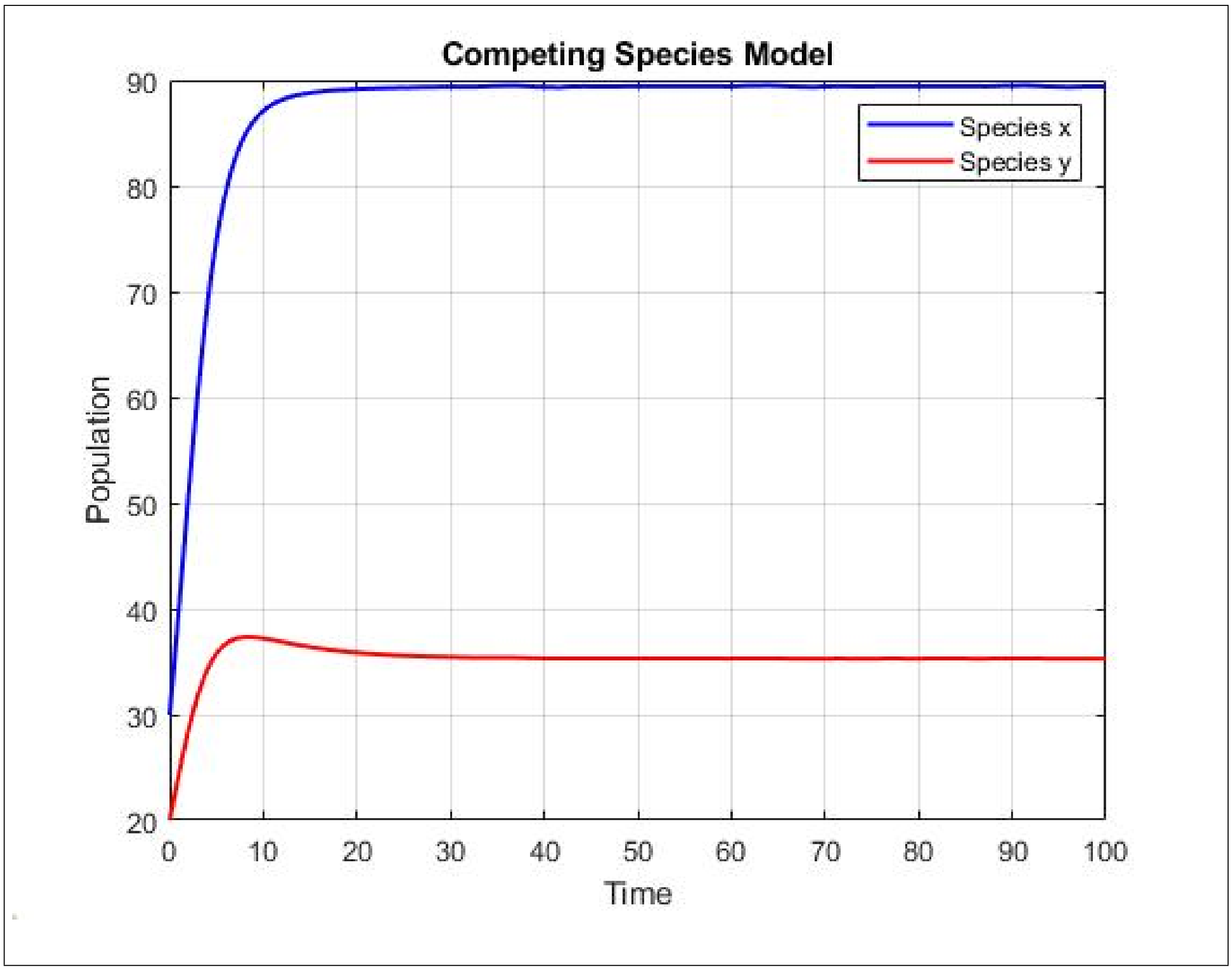
Step 1: Define parameter r_x, r_y, K_x, K_y, α and β,
Step 2: Set initial populations for x and y,
Step 3: Define the system of differential equations,
Step 4: Use ode45 to solve the system,
Step 5: Plot the population dynamics.

Program

```
% Define constants for the competing species model
rx = 0.5;    % Growth rate of species x
ry = 0.4;    % Growth rate of species y
Kx = 100;    % Carrying capacity of species x
Ky = 80;     % Carrying capacity of species y
alpha = 0.3; % Competition coefficient on x by y
beta = 0.5;  % Competition coefficient on y by x

% Initial populations and time span
initial_populations = [30, 20];
T = 100;

% Define the system of differential equations
competing_species_model = @(t, Y) [rx * Y(1) * (1 - (Y(1) + alpha * Y(2)) / Kx);
                                   ry * Y(2) * (1 - (Y(2) + beta * Y(1)) / Ky)];

% Solve the system using ode45
[time, populations] = ode45(competing_species_model, [0 T], initial_populations);

% Plot the results
figure;
plot(time, populations(:,1), 'b', 'LineWidth', 1.5); % Plot for species x
hold on;
plot(time, populations(:,2), 'r', 'LineWidth', 1.5); % Plot for species y
title('Competing Species Model');
xlabel('Time');
ylabel('Population');
legend('Species x', 'Species y');
grid on;
```

Output

A plot showing the population of both species over time, potentially showing coexistence, dominance, or extinction.



Exercise Problem

Consider two species A and B, competing for the same resources, described by the following system of differential equations:

$$\frac{dA}{dt} = r_A A\left(1 - \frac{A+\alpha B}{K_A}\right),$$
$$\frac{dB}{dt} = r_B B\left(1 - \frac{B+\beta A}{K_B}\right),$$

with

$$A(0) = A_0, \quad B(0) \;=\; B_0,$$

where $r_A = 0.5$, $r_B = 0.3$, $K_A = 150$, $K_B = 100$, $\alpha = 0.4$, $\beta = 0.6$ with initial values: $A_0 = 40$, $B_0 = 30$. **Tasks:**
1. Solve the system using MATLAB's 'ode45' and plot the population dynamics of both species over time (0 to 50).
2. Analyze the plot: Do the populations stabilize or oscillate? Does one species dominate, or do both coexist?
3. Modify the competition coefficients α and β to 0.8 and 0.3, respectively, and observe how the dynamics change.

Practical No. 15

Aim

To model the spread of an infectious disease using the SEIR model, accounting for a latent (exposed) stage.

Problem

In a closed region, individuals are susceptible (S), exposed (E), infected (I), recovered (R). Model the progression of the disease over time.

Theory

The SEIR model is described by

$$\frac{dS}{dt} = -\beta S I,$$
$$\frac{dE}{dt} = \beta S I - \sigma E,$$
$$\frac{dI}{dt} = \sigma E - \gamma I,$$

$$\frac{dR}{dt} = \gamma I,$$

with

$$S(0) = S_0, \quad E(0) = E_0 \quad I(0) = I_0, \quad R(0) = R_0,$$

where σ is the rate at which exposed individuals become infectious.

Algorithm

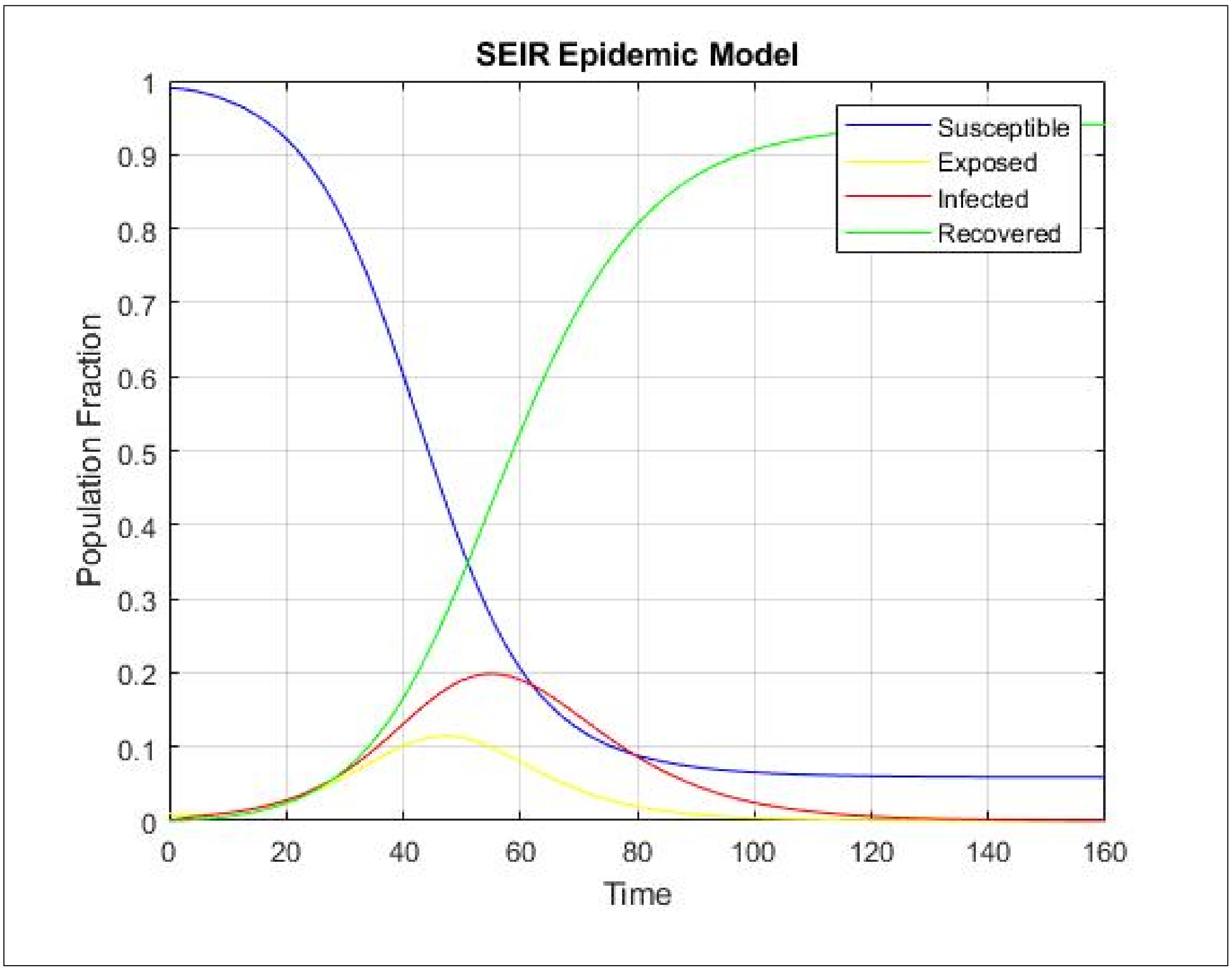
Step 1: Define initial values and parameters β, σ and γ,
Step 2: Set up differential equations for S, E, I and R,
Step 3: Use ode45 to solve the system,
Step 4: Plot S, E, I, and R over time.

Program

```
beta = 0.3;     % Infection rate
sigma = 0.2;    % Rate of exposed to infected
gamma = 0.1;    % Recovery rate
initial_values = [0.99, 0.01, 0, 0];  % Initial S, E, I, R values
T = 160;

seir_model = @(t, Y) [-beta * Y(1) * Y(3); beta * Y(1) * Y(3) - sigma * Y(2);
                      sigma * Y(2) - gamma * Y(3); gamma * Y(3)];

[time, SEIR] = ode45(seir_model, [0 T], initial_values);

plot(time, SEIR(:,1), ‘b’, time, SEIR(:,2), ‘y’, time, SEIR(:,3), ‘r’, time, SEIR
(:,4), ‘g’);
title(‘SEIR Epidemic Model’);
xlabel(‘Time’);
ylabel(‘Population Fraction’);
legend(‘Susceptible’, ‘Exposed’, ‘Infected’, ‘Recovered’);
grid on;
```

Output

A plot showing the dynamics of each population group over time, illustrating the epidemic curve with a latent phase.



Exercise Problem

Use MATLAB's ode45 to solve the system of differential equations and plot the time evolution of the susceptible, exposed, infected, and recovered populations over a period of 160 days.
Analyze the epidemic curve: Describe how the number of exposed individuals changes over time compared to the infected and recovered individuals. Identify the peak of the infected population. What impact does the latent (exposed) phase have on the spread of the disease compared to a simpler SIR model?

Practical No. 16

Aim

To model the dynamics that evolve in discrete time steps using difference equations.

Problem

Analyze a population model where the population changes in discrete steps (e.g., annually), such as modeling the population growth of a species with annual breeding.

Theory

Difference equations describe the relationships where each term is based on previous terms, rather than on continuous rates of change. They are essential for modeling systems where changes occur in distinct intervals.

For example, a simple population model can be described as

$$P_{n+1} = P_n + rP_n\left(1 - \frac{P_n}{K}\right),$$

where

- P_n is the population at time step n,
- r is the growth rate,
- K is the carrying capacity of the environment.

This difference equation is a discrete system analog to the logistic growth model used in continuous systems.

Algorithm

Step 1: Define parameters r (growth rate) and K (carrying capacity),
Step 2: Set the initial population P_0 and number of time steps,
Step 3: Create a loop to apply the difference equation iteratively,
Step 4: Plot the population P over time.

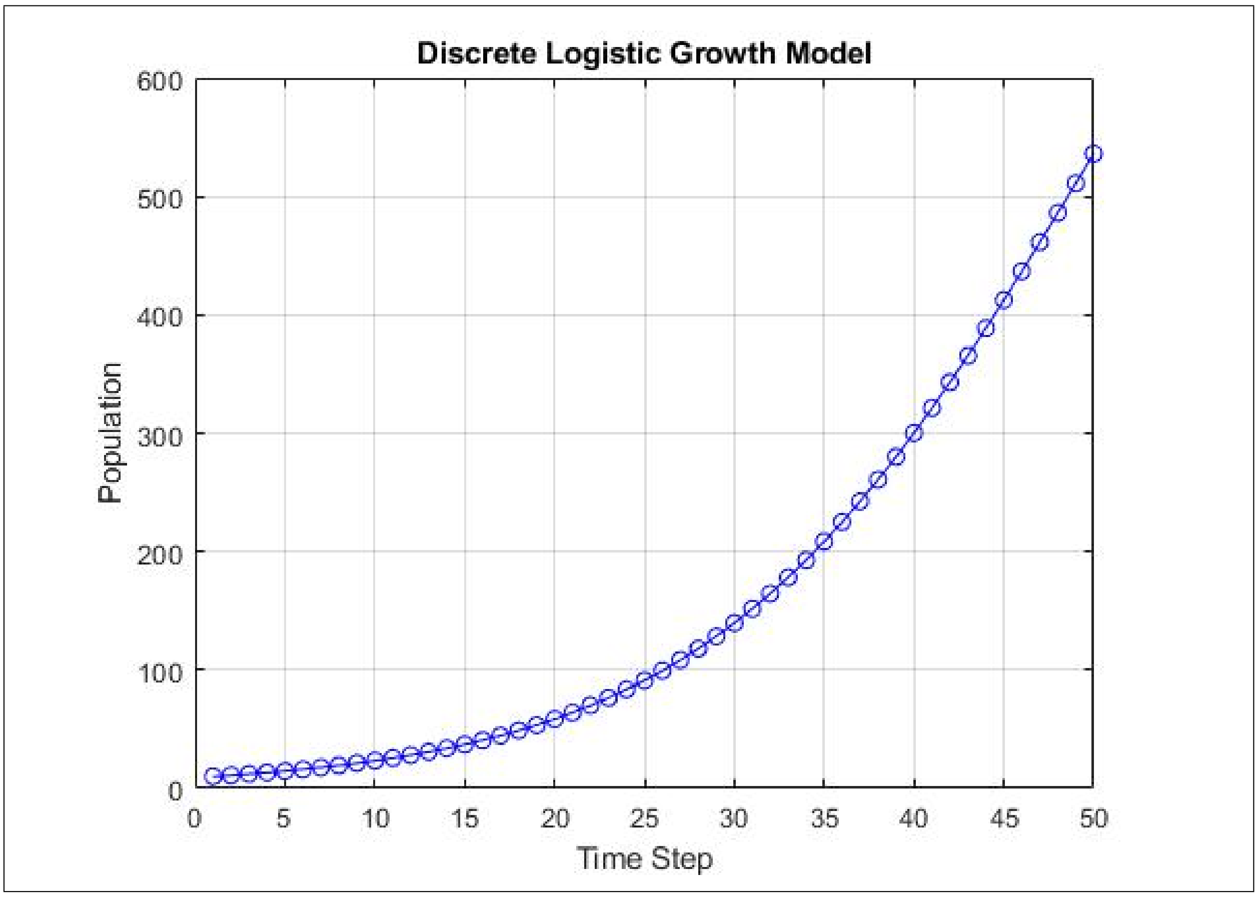
Program

```
r = 0.1;      % Growth rate
K = 1000;     % Carrying capacity
P0 = 10;      % Initial population
N = 50;       % Number of time steps

P = zeros(1, N);  % Initialize population array
P(1) = P0;        % Set initial population

for n = 1:N-1
```

```
    P(n+1) = P(n) + r * P(n) * (1 - P(n) / K);
end

plot(1:N, P, ‘b-o’);
title(‘Discrete Logistic Growth Model’);
xlabel(‘Time Step’);
ylabel(‘Population’);
grid on;
```

Output

A plot that illustrates how the population evolves over discrete time steps, stabilizing as it approaches the carrying capacity K.



Exercise Problem

Implement the difference equation in MATLAB to compute the population P_n over 50 time steps. Then plot the population P over time. Describe the long-term behavior of the population. Does it approach a stable value? If so, what is it?

Practical No. 17

Aim

To model the spread of a disease in a population where individuals are infected, recover, and then become susceptible again (no immunity).

Problem

We consider a population where individuals are susceptible (S) and infected (I). At each time step, new infections occur, and some infected individuals recover. The SIS model captures these dynamics, with individuals transitioning between the susceptible and infected states without gaining immunity.

Theory

The SIS model equations are given by

$$S_{n+1} = S_n - \beta S_n I_n + \gamma I_n,$$

$$I_{n+1} = I_n + \beta S_n I_n - \gamma I_n,$$

where

- S_n is the number of susceptible individuals at time n,
- I_n is the number of infected individuals at time n,
- β is the infection rate,
- γ is the recovery rate.

This model assumes a closed population without any demographic effects.

Algorithm

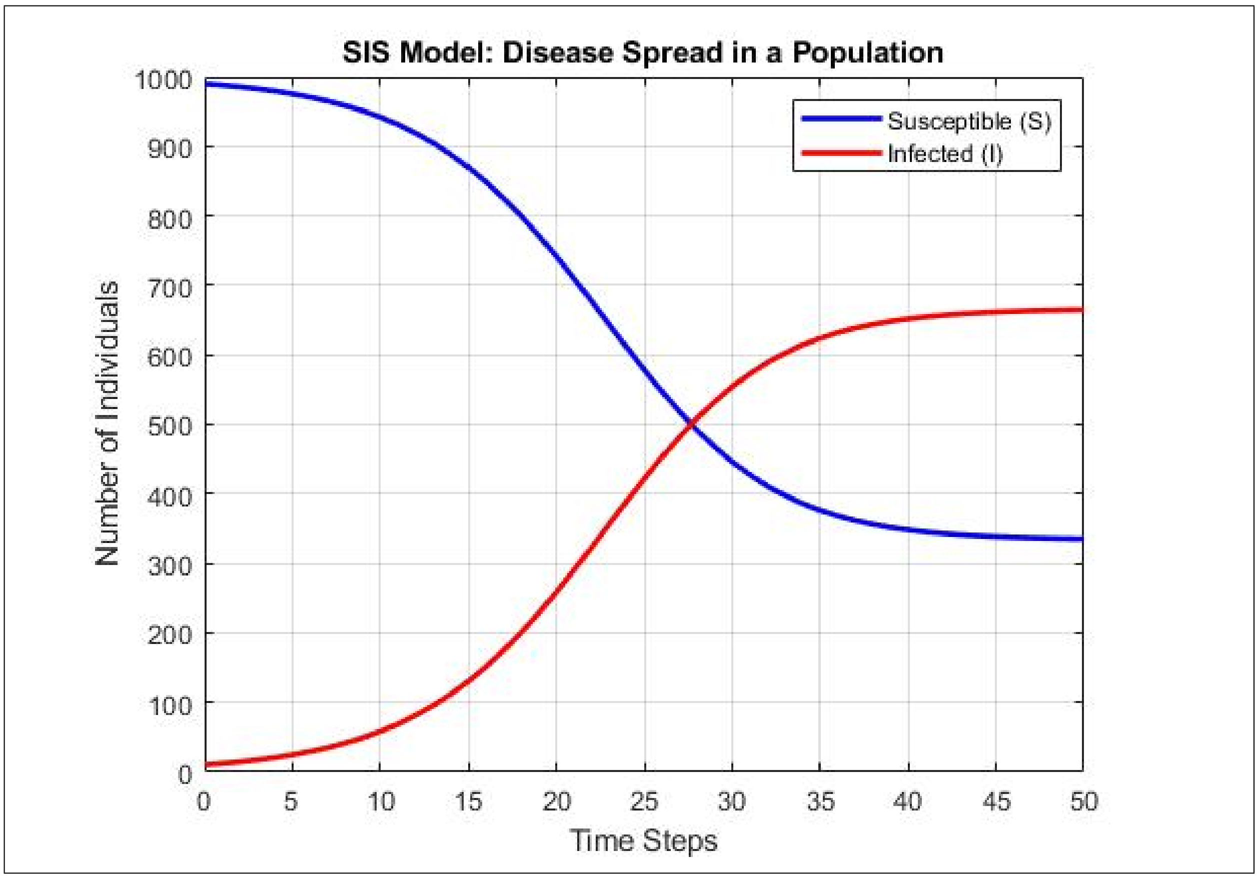
Step 1: Define parameters I_0, S_0, β, γ, N,
Step 2: Set up iterations: define total simulation time steps,
Step 3: Initialize arrays: prepare arrays to store S and I at each time step,
Step 4: Iterate SIS model: compute $S_(n+1)$ and $I_(n+1)$ for each time step using difference equations,
Step 5: Plot results: display susceptible and infected populations over time.

Program

```
% Parameters for the SIS Model
beta = 0.3;     % Infection rate
gamma = 0.1;    % Recovery rate
N = 1000;       % Total population

% Initial conditions
S0 = 990;       % Initial number of susceptible individuals
I0 = 10;        % Initial number of infected individuals
T = 50;         % Total number of time steps

% Initialize arrays to store susceptible and infected populations
S = zeros(1, T+1);
I = zeros(1, T+1);

% Set initial values
S(1) = S0;
I(1) = I0;

% Iterate the SIS model equations over time
for n = 1:T
    S(n+1) = S(n) - beta * S(n) * I(n) / N + gamma * I(n);
    I(n+1) = I(n) + beta * S(n) * I(n) / N - gamma * I(n);
end

% Plotting the SIS model results
figure;
plot(0:T, S, ‘b-’, ‘LineWidth’, 2); hold on;
plot(0:T, I, ‘r-’, ‘LineWidth’, 2);
xlabel(‘Time Steps’);
ylabel(‘Number of Individuals’);
title(‘SIS Model: Disease Spread in a Population’);
legend(‘Susceptible (S)’, ‘Infected (I)’);
grid on;
```

Output

The plot will show the dynamics of susceptible (S) and infected (I) populations over time. The susceptible population decreases as infections rise, then stabilizes as recovered individuals rejoin the susceptible pool.
Infected individuals initially increase, then stabilize or oscillate around an equilibrium as infections and recoveries balance. This reflects a steady state in disease spread, with no immunity, as susceptible and infected groups continuously interact.



Exercise Problem

Implement an SIS model using the difference equations in MATLAB to compute the number of susceptible and infected individuals over 50 time steps. Plot the number of susceptible and infected individuals over time. Analyze the long-term behavior of the disease spread. Does the number of infected individuals stabilize or fluctuate over time?

Practical No. 18

Aim

To model population growth with environmental stress affecting the carrying capacity.

Problem

Consider a population with an initial size of 500 and a carrying capacity of 2,000. Due to an environmental stress factor, the carrying capacity is decreased by 5% every year. The growth rate is 5% per year. Model the population over 20 years.

Theory

The logistic growth model with a time-dependent carrying capacity $K(t)$ is:

$$P_{n+1} = P_n + rP_n\left(1 - \frac{P_n}{K_n}\right),$$

with $P(0) = P_0$, where

- P_n is the population at time n,
- r is the growth rate (5%),
- K_n is the carrying capacity at time n, which decreases by 5% annually.

Algorithm

Step 1: Set the initial population $P_0 = 500$, growth rate $r = 0.05$, initial carrying capacity $K_0 = 2000$, and annual decrease in carrying capacity (5%),
Step 2: Define the number of years,
Step 3: Use a loop to iteratively calculate population with decreasing carrying capacity,
Step 4: Plot the population over time.

Program

```
r = 0.05;           % Growth rate
K0 = 2000;          % Initial carrying capacity
P0 = 500;           % Initial population
N = 20;             % Number of years
K_decrement = 0.05 * K0;  % Annual decrease in carrying capacity

P = zeros(1, N);
K = zeros(1, N);
```

```
P(1) = P0;
K(1) = K0;

for n = 1:N-1
    P(n+1) = P(n) + r * P(n) * (1 - P(n) / K(n));
    K(n+1) = K(n) - K_decrement;
end

plot(1:N, P, ‘b-o’);
title(‘Logistic Growth with Environmental Stress’);
xlabel(‘Year’);
ylabel(‘Population’);
grid on;
```

Output

A plot showing population growth with an initially rapid increase that slows as the carrying capacity decreases due to environmental stress.

Exercise Problem

Implement the logistic growth model using the difference equation in MATLAB, accounting for the 5% annual decrease in carrying capacity. Plot the population over 20 years. Analyze the impact of the decreasing carrying capacity on long-term population growth. How does the environmental stress affect the population dynamics?

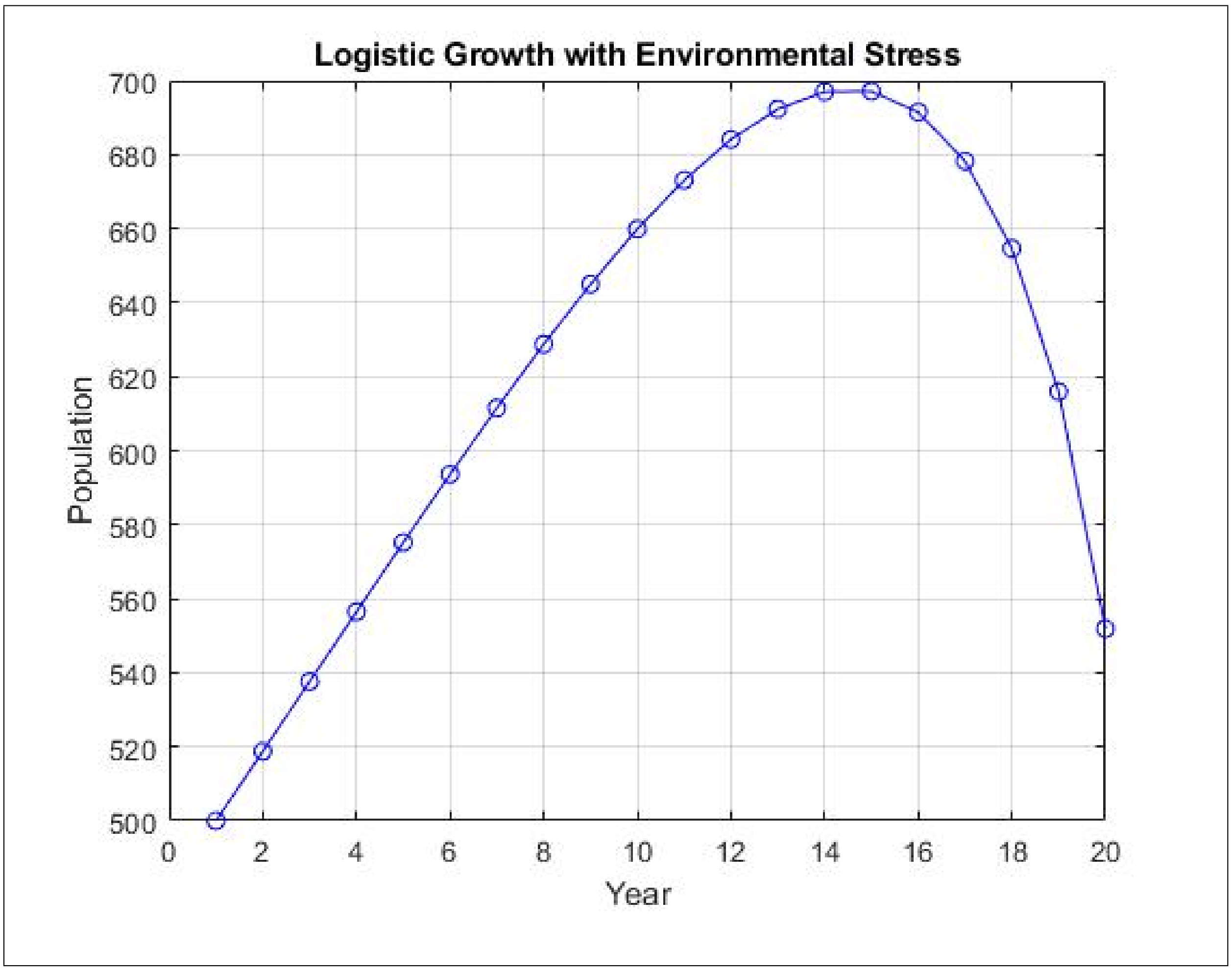
Practical No. 19

Aim

To model the spread of an infectious disease with an SIR model, incorporating vaccination.

Problem

In a population of 10,000, 100 individuals are initially infected. The transmission rate is 0.3, the recovery rate is 0.1, and the vaccination rate is 0.05. Model the disease spread over 100 days.



Theory

The SIR model with vaccination is given by the following system of ordinary differential equations

$$\frac{dS}{dt} = -\beta SI - \mu S,$$
$$\frac{dI}{dt} = \beta SI - \gamma I,$$
$$\frac{dR}{dt} = \gamma I,$$

with $S(0) = S_0, \quad I(0) = I_0, \quad R(0) = R_0$, where

- S is the susceptible population,
- I is the infected population,
- R is the recovered population,
- β is the transmission rate,

- γ is the recovery rate,
- μ is the vaccination rate.

Algorithm

Step 1: Set initial values for the SIR model,
Step 2: Implement a numerical solution using Eulers method,
Step 3: Plot the results for S, I and R.

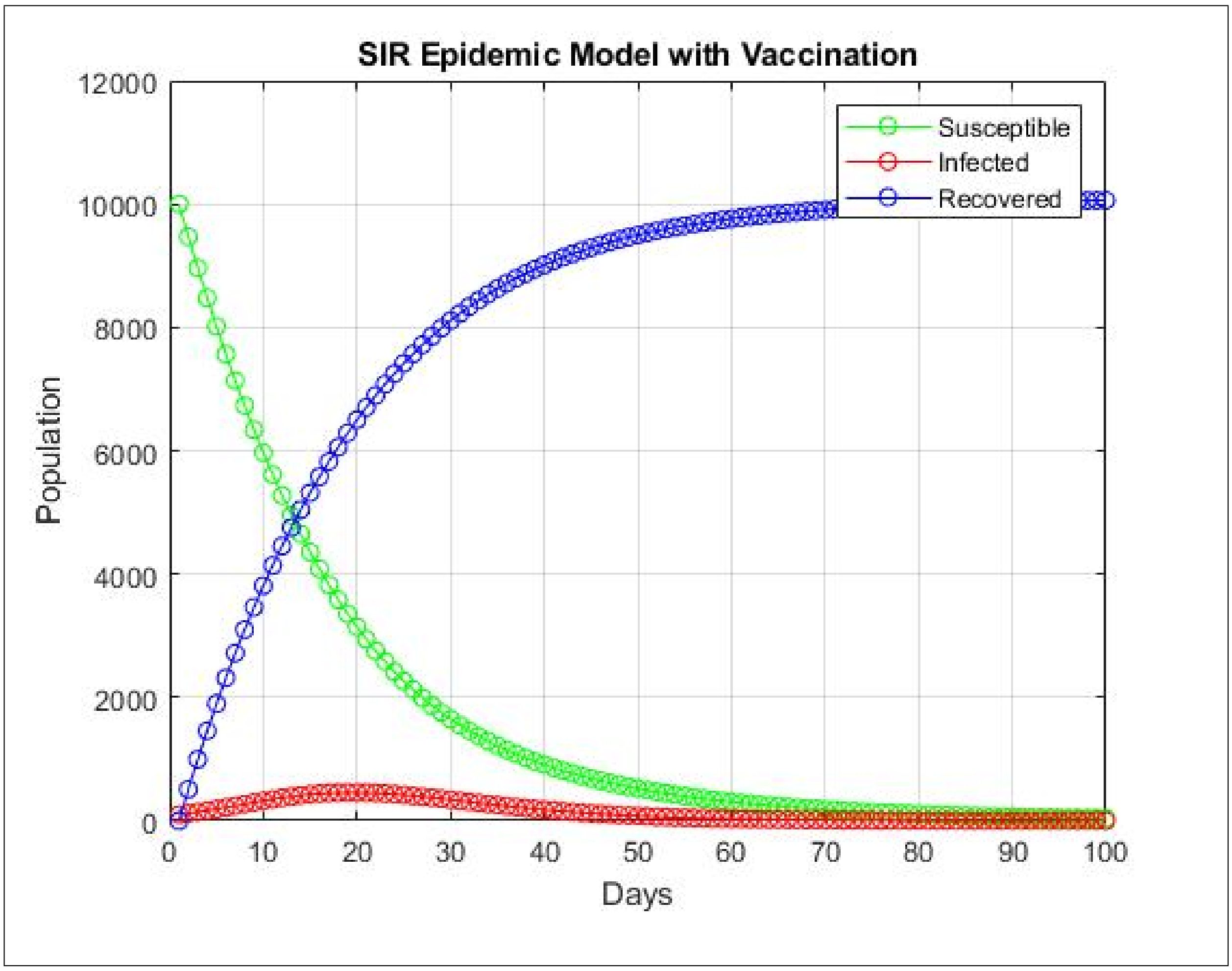
Program

```
% Initial values and parameters for the SIR model with vaccination
S0 = 9999;    % Initial susceptible population
I0 = 100;     % Initial infected population
R0 = 0;       % Initial recovered population
beta = 0.3;   % Transmission rate
gamma = 0.1;  % Recovery rate
mu = 0.05;    % Vaccination rate
N = 100;      % Number of days
dt = 1;       % Time step (1 day)

% Initialize arrays for storing population values over time
S = zeros(1, N);
I = zeros(1, N);
R = zeros(1, N);
S(1) = S0;
I(1) = I0;
R(1) = R0;

% Total population (constant)
TotalPop = S0 + I0 + R0;

% Simulation loop for SIR model
for t = 1:N-1
    dS = (-beta * S(t) * I(t) / TotalPop - mu * S(t)) * dt;
    dI = (beta * S(t) * I(t) / TotalPop - gamma * I(t)) * dt;
    dR = (gamma * I(t) + mu * S(t)) * dt;

    S(t+1) = S(t) + dS;
    I(t+1) = I(t) + dI;
    R(t+1) = R(t) + dR;
end
```

```
% Plot the results
figure;
plot(1:N, S, ‘g-o’, 1:N, I, ‘r-o’, 1:N, R, ‘b-o’);
title(‘SIR Epidemic Model with Vaccination’);
xlabel(‘Days’);
ylabel(‘Population’);
legend(‘Susceptible’, ‘Infected’, ‘Recovered’);
grid on;
```

Output

A plot showing the dynamics of susceptible, infected, and recovered populations over time, with the infection rate declining due to the vaccination effect.



Exercise Problem

Implement the SIR model with vaccination using Eulers method in MATLAB. Simulate the disease spread for 100 days. Plot the time series of susceptible, infected and recovered populations. Analyze the effect of vaccination on the disease spread. How does vaccination influence the number of infected individuals over time?

Practical No. 20

Aim

To model the repayment of a loan using difference equations and analyze the solution using MATLAB.

Problem

Given an initial loan amount, monthly repayment amount, and interest rate, we aim to model the repayment of the loan over time using a difference equation.

Theory

The repayment of a loan can be described by the difference equation as

$$L_{n+1} = L_n(1 + r) - M,$$

with $L(0) = L_0$, where

- L_n is the amount of loan at month n,
- r is the monthly interest rate,
- M is the monthly repayment amount.

Algorithm

Step 1: Define parameters: initial loan amount L_0, monthly repayment M, interest rate r, and number of months N,
Step 2: Initialize array: store loan amounts at each month,
Step 3: Calculate: use a loop to compute the loan amount at each month using the difference equation,
Step 4: Plot: Visualize loan repayment over time.

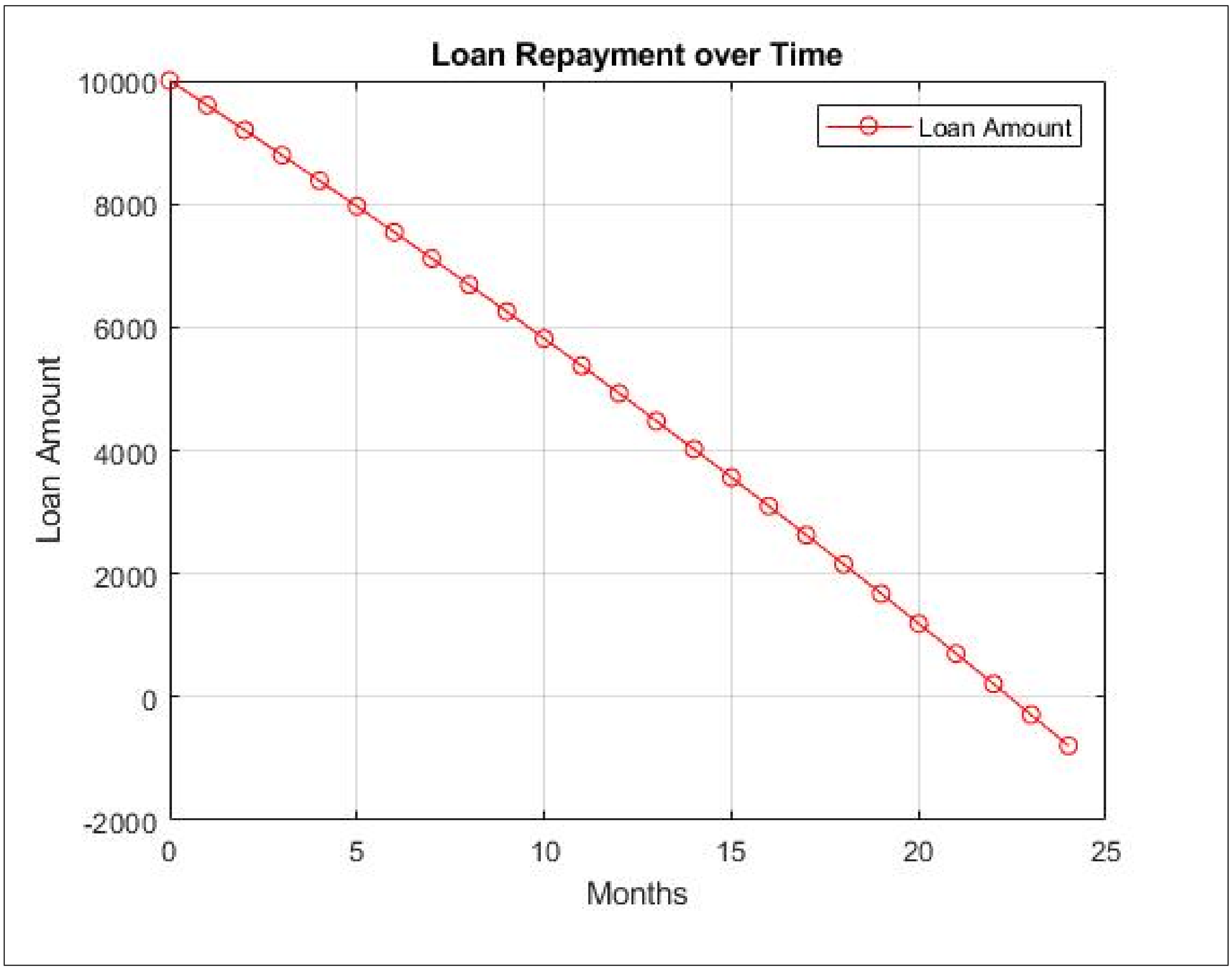
Program

```
% Parameters
L0 = 10000; % Initial loan amount
M = 500; % Monthly repayment
r = 0.01; % Monthly interest rate
N = 24; % Number of months

% Initialize loan amount array
L = zeros(1, N+1);
L(1) = L0;

% Iterate using the difference equation
for n = 1:N
    L(n+1) = L(n) * (1 + r) - M;
end
```

```
% Plotting the results
figure;
plot(0:N, L, 'r-o', 'DisplayName', 'Loan Amount');
xlabel('Months');
ylabel('Loan Amount');
title('Loan Repayment over Time');
legend;
grid on;
```

Output

MATLAB program will generate a plot showing the loan repayment over time. The x-axis represents the number of months, and the y-axis represents the loan amount. The plot will show a decrease in the loan amount due to monthly repayments and interest.



Exercise Problem

Write a MATLAB program to simulate the loan repayment over 24 months using the difference equation. Plot the loan amount over time (number of months). How does the loan amount decrease over time? Does it follow the expected pattern of reducing due to the monthly repayments and accumulating interest?

Chapter 4

Elementary Number Theory

Practical No. 1

Aim

To write a MATLAB program to find the factor of a given positive integer.

Problem

Write a program to compute the factors of $n = 50$.

Theory

The factor of a positive integer is an integer that divides the given integer.

Algorithm

Step 1: Define the integer $n = 50$;

Step 2: Compute the factors of n using the command `factor(n)`;

Step 3: Output the factors using the command `disp`;

Step 4: After writing the code press the green triangle button (▶) to run the code.

Program

```
clc
clear
%Set the integer n==50
n=50;
```

```
%Compute the factor of n=50.
f = factor(50);
%The output
disp(['Factor of n = ' num2str(n) ' are:']);
disp(f);
```

Output

```
Factor of n = 50 are:
     2     5     5
```

Conclusion

The program correctly computes the factors of the given positive integer.

Exercise Problem

Write a program to compute the factors of $n = 120$.

Practical No. 2

Aim

To write a MATLAB program to find the divisors of a given positive integer.

Problem

Write a program to compute all possible divisors of $n = 100$.

Theory

The divisor of an integer is an integer that completely divides the given integer leaving remainder 0.

Algorithm:

Step 1: Take the input of the integer n;

Step 2: Compute the divisors using the command `finddivisor(n)`;

Step 3: Output the divisors using the command `disp`;

Step 4: Define the function `finddivisor(n)`;

Step 5: After writing the code press the green triangle button (▶) to run the code.

Program:

```
% Find divisors of a positive integer
clc;
clear;
n= fix(input("Enter an integer : ")) ;
divisors = finddivisors(n);
fprintf("Divisors of %d :\n", n)
disp(divisors)

function divisors = finddivisors(n) %find divisors of an integer
    divisors = [];
    for i = 1:n
        if mod(n, i) == 0
            divisors = [divisors, i];
        end
    end
end
```

Output

```
Enter an integer : 100
Divisors of 100 :
     1     2     4     5    10    20    25    50   100
```

Conclusion

The program correctly computes the divisor of the given integer.

Exercise Problem

Write a program to compute all possible divisors divisors of $n = 120$.

Practical No. 3

Aim

To write a MATLAB program to find the remainder of a given integer when divided by some other positive integer.

Problem

Write a program to compute the remainder on division of 123339 by 15.

Theory

The remainder r of an integer n when divided by another positive integer m is given by $r = n - q.m$ such that $0 \leq r < m$ where q is the quotient of n when divided by m.

Algorithm:

Step 1: Set an integer $n = 123339$, $m = 15$.

Step 2: Compute the remainder using the command `rem(n,m)`.

Step 3: Output the remainder using the command `disp`.

Step 4: After writing the code press the green triangle button (▶) to run the code.

Program:

```
clc
clear
%set an integer
n=12339;
%set divisor
m=15;
%compute the remainder
r = rem(n,m);
%output
disp(['The remainder of n = ' num2str(n) ' when divided by m = ' num2str(m) ' is:']);
disp(r)
```

Output

```
The remainder of n = 12339 when divided by m = 15 is:
     9
```

Conclusion

The program correctly computes the remainder of the given integer when divided by a positive integer.

Exercise Problem

Write a program which takes input of an integer n and decides if it is odd or even by calculating remainder of division of n by 2. Run the program for $n = 20$ and $n = 97$.

Practical No. 4

Aim

To write a MATLAB program to find the greatest common divisor (GCD) of given positive integers.

Problem

Write a program to compute the greatest common divisor (GCD) of $28, 68$ and 104.

Theory

The greatest common divisor (GCD) of two or more integers is the largest positive integer that divides all those integers without remainder.

Algorithm

Step 1: Set the integers $n1 = 28, n2 = 68, n3 = 104$;

Step 2: Compute the greatest common divisor (GCD) using command `n = gcd(n1,n2)` and `k = gcd(n,n3)`;

Step 3: Output the greatest common divisor (GCD) using command `disp`;

Step 4: After writing the code press the green triangle button (▶) to run the code.

Program

```
clc
clear
%set the integers n1, n2 and n3
n1=28;
n2=68;
n3=104;
%compute the GCD
n=gcd(n1,n2);
k=gcd(n3,n);
%output
disp(['GCD:']);
disp(k)
```

Output

```
GCD:
    4
```

Conclusion

The program correctly computes the greatest common divisor (GCD) of the given integers.

Exercise Problem

Write a program to test if two given positive integers $n1$ and $n2$ are coprimes. Run your program for $n1 = 20$ and $n2 = 32$.

Practical No. 5

Aim

To write a MATLAB program to find the least common multiple (LCM) of integers.

Problem

Write a program to find the least common multiple of $15, 20$ and 50.

Theory

The least common multiple (LCM) of two or more integers is the smallest positive integer that is divisible by all those integers.

Algorithm

Step 1: Set the integers $n1 = 15, n2 = 20, n3 = 50$;

Step 2: Compute the LCM using the command `lcm`;

Step 3: Output the result using the command `disp`;

Step 4: After writing the code press the green triangle button (▶) to run the code.

Program

```
clc
clear
%set the integers n1, n2 and n3
n1=15;
n2=20;
n3=50;
%compute the LCM
n=lcm(n1,n2);
k=lcm(n3,n);
%output
disp(k)
```

Output

```
LCM:
   300
```

Conclusion

The program performs correctly to find the least common multiple (LCM) of integers.

Exercise Problem

Write a program to find the least common multiple of $23, 25$ and 100.

Practical No. 6

Aim

To solve linear congruences using MATLAB.

Problem

Write a program to solve linear congruence given by

$$12x = 9(\text{mod}15);$$

Theory

A linear congruence is an equation of the form:$ax = b$ (mod m where $m \neq a$. Incongruent solutions of this linear congruence form a subset of $\{0, 1, 2, ..., m-1\}$.

Algorithm

Step 1: Calculate the incongruent solutions of the given linear congruence using `solvelinearcongruence(12, 9, 15)`;

Step 2: Output using the command `fprintf`;

Step 3: Define the function `solvelinearcongruence(a, b, m)`.

Program

```
clc ;
clear ;
a = 12;
b = 9;
m = 15;
solutions = solvelinearcongruence(a, b, m);

if isempty(solutions)
    fprintf('The linear congruence %dx = %d(mod %d) has no solutions.\n', a, b, m);
else
    fprintf('The linear congruence %dx = %d(mod %d) has following solutions:\n', a, b, m);
    disp(solutions);
end

function solutions = solvelinearcongruence(a, b, m)
```

```
    if m < 0
        error("Error. \nThe modulus must be positive integer.")
    end

    d = gcd(a, m);
    if mod(b, d) == 0
        solutions = zeros(1, d);
        idx = 1;
        for x = 0:(m-1)
            if mod(a*x - b, m) == 0
                solutions(idx) = x;
                idx = idx + 1;
            end
        end
    else
        solutions = [];
    end
end
```

Output

```
The linear congruence 12x = 9(mod 15) has following solutions:
     2     7    12
```

Conclusion

The program correctly finds the solution of linear congruence equation.

Exercise Problem

Write a program to solve the linear congruence $30x = 8 \pmod 9$.

Practical No. 7

Aim

To write a MATLAB program to find the Mersenne numbers.

Problem

Write a program to find the first 10 Mersenne numbers.

Theory

A Mersenne number is a number of the form $M(n) = 2^n - 1$, where n is an integer.

Algorithm

Step 1: Set the range n using command `linspace(1, 10, 10)`;

Step 2: Calculate the Mersenne number using formula $mersenne\_num = 2^n - 1$;

Step 3: Make a table using command `table`;

Step 4: Check whether system has trivial solution or infinite solutions;

Step 5: Output the solution using the command `disp`;

Step 6: Run the code by pressing the run button.

Program

```
clc;
clear;
n = linspace(1, 10, 10) ;
mersenne_num = 2.^n - 1 ;
t1 = table(n', mersenne_num','VariableNames',{'n','Mersenne number'}) ;
disp(t1)
```

Output

```
    n     Mersenne number
    --    ---------------

     1            1
     2            3
     3            7
     4           15
     5           31
     6           63
     7          127
     8          255
     9            511
    10           1023
```

Conclusion

The program correctly computes the first 10 Mersenne numbers.

Exercise Problem

Write a program to test if a given number is a Mersenne number. Run the program for $n = 8191$.

Practical No. 8

Aim

To write a program to find the first 8 Mersenne primes.

Problem

Write a program to determine the first 8 Mersenne primes.

Theory

A Mersenne prime is a prime number of the form $M(p) = 2^p - 1$, where p is prime number.

Algorithm

Step 1: Find the 8 Mersenne primes using the function `mersenneprimes(8)`;

Step 2: Output the result using the command `fprintf`.

Step 3: Define the function `mersenneprimes(n)` to return n Mersenne primes;

Program

```
clc;
clear;
n = 8;
```

```
[primes, mprimes] = mersenneprimes(n);

t1 = table((1:n)', primes', sym(mprimes'), 'VariableNames', {'n', 'p','Mersenne prime'});
disp(t1);

function [primes, mprimes] = mersenneprimes(n)
    if n < 1
        error('n should be at least 1.')
    end

    primes = zeros(1, n);
    mprimes = primes;
    idx = 1;

    p = 2;
    while idx <= n
        mp = 2^p - 1;
        if isprime(p) && isprime(mp)
            primes(idx) = p;
            mprimes(idx) = mp;
            idx = idx + 1;
        end
        p = p + 1;
    end
end
```

Output

```
    n    p     Mersenne prime
    -    --    --------------

    1     2      3
    2     3      7
    3     5      31
    4     7      127
    5    13      8191
    6    17      131071
    7    19      524287
    8    31      2147483647
```

Conclusion

The program correctly find the first 8 Mersenne primes.

Exercise Problem

Write a program to test if a given number is a Mersenne prime. Run the program for $n = 8191$.

Practical No. 9

Aim

To write a MATLAB program to find the first 8 even perfect numbers.

Problem

Write a program to compute the first 8 perfect numbers.

Theory

A perfect number is a positive integer that is equal to the sum of its proper divisors (all divisors excluding the number itself). The Euclid-Euler theorem states that an even number is perfect if and only if it has the form $2^{p-1}(2^p - 1)$, where $(2^p - 1)$ is a Mersenne prime for any prime p.

Algorithm

Step 1: Find the 8 Mersenne primes using the function `perfectNumbers(8)`;

Step 2: Output the result using the command `fprintf`.

Step 3: Define the function `perfectNumbers(n)` to return n Mersenne primes.

Program

```
clc;
clear;
n = 8;
[primes, perfect] = perfectNumbers(n);

t1 = table((1:n)', primes', sym(perfect'), 'VariableNames', {'n', 'p','Perfect Nu
disp(t1);

function [primes, perfect] = perfectNumbers(n)
```

```
    if n < 1
        error('n should be at least 1.')
    end

    primes = zeros(1, n);
    perfect = primes;
    idx = 1;

    p = 2;
    while idx <= n
        mp = 2^p - 1;
        if isprime(p) && isprime(mp)
            primes(idx) = p;
            perfect(idx) = 2^(p - 1)*mp;
            idx = idx + 1;
        end
        p = p + 1;
    end
end
```

Output

```
    n    p       Perfect Number
    -    --    -------------------

    1     2    6
    2     3    28
    3     5    496
    4     7    8128
    5    13    33550336
    6    17    8589869056
    7    19    137438691328
    8    31    2305843008139952128
```

Conclusion

The program correctly computes first 8 even perfect number.

Exercise Problem

Write a program to test if a given number n is an even perfect number. Run the program for $n = 28$.

Practical No. 10

Aim

To write a MATLAB program to find the order of an integer a modulo m.

Problem

Write a program to compute the order of 2 modulo 385.

Theory

The order of an integer a modulo m is the smallest positive value of x which satisfies the congruence -

$$a^x = 1(\text{mod } m), \quad \gcd(a, m) = 1.$$

Algorithm

Step 1: Set the integers $m = 385$, $a = 2$;

Step 2: If a and m are coprimes, then calculate the order and print;

Step 3: Otherwise, print the error.

Program

```
clc ;
clear;

m = 385 ;
a = 2;
if (gcd(a,m) == 1)
    phi = eulerPhi(m);
    ord = 1;
    for d = divisors(phi)
        if(powermod(a, d, m) == 1)
            ord = d;
            break;
        end
    end
    fprintf("The order of %d modulo %d is %d.\n", a, m, ord);
else
    error('a and m must be coprime. i.e gcd(a,m) = 1')
```

```
end
```

Output

```
The order of 2 modulo 385 is 60.
```

Conclusion

The program correctly computes the order of 2 modulo 385.

Exercise Problem

Write a program to compute the order of 7 modulo 32.

Practical No. 11

Aim

To write a MATLAB program to find the primitive roots modulo m.

Problem

Write a program to compute all primitive roots modulo 11.

Theory

An integer g is a primitive root modulo n if for every integer a coprime to n, there is some integer k for which

$$g^k = a \bmod n.$$

Algorithm

Step 1: Set the value $G = Z_n$.

Step 2: Compute the primitive roots using the command `isPrimitiveRoot(G, 11)`.

Step 3: Output the primitive roots using the command `disp`.

Program

```
clc
clear
n = 11;
G = 1:11;
isPR = isPrimitiveRoot(G, 11);
PR = G(isPR);% all primitive roots
disp(['All the primitive roots modulo ' num2str(n) ' are as following:'])
disp(PR)
```

Output:

```
All the primitive roots modulo 11 are as following:
     2     6     7     8
```

Conclusion:

The program successfully computes all the primitive roots modulo 11.

Exercise Problem

Given two positive integers a and b, write a program to check if a is a primitive root modulo b. Run the program for $a = 2$ and $b = 5$.

Practical No. 12

Aim

To write a program to find the Euler totient function Phi of an integer.

Problem

Write a program to find the value of Euler Phi function for $n = 50$.

Theory

The Eulers Totient function $\phi(n)$ counts the number of positive integers up to n, that are relatively prime to n.

Algorithm

Step 1: Set an integer `n = 50`;

Step 2: Compute the value of Euler Phi function using the command `eulerPhi(n)`;

Step 3: Output the Euler phi function value using the command `fprintf`.

Program

```
clc;
clear;
n = 50;
phi = eulerPhi(n);
fprintf("The Euler's Totient phi(%d) = %d.\n", n, phi)
```

Output

```
The Euler's Totient phi(50) = 20.
```

Conclusion

The program successfully computed the value of Euler phi function at $n = 50$.

Exercise Problem

Write a program to find the value of Euler Phi function for $n = 120$.

Practical No. 13

Aim

To write a MATLAB program to find the first 10 Fibonacci numbers.

Problem

Write a program to find the first 10 Fibonacci numbers.

Theory

The Fibonacci numbers are the sequence $0, 1, 1, 2, \ldots$. The Fibonacci sequence can be defined recursively as $a_0 = 0$, $a_1 = 1$ and

$$a_n = a_{n-1} + a_{n-2}, \qquad n \geq 2.$$

Algorithm

Step 1: Set the number $n = 1 : 10$;

Step 2: Compute the Fibonacci number using the command `fibonacci(n)`.

Program

```
clc;
clear;
n = 1:10;
k = fibonacci(n)
```

Output

```
k =

     1     1     2     3     5     8    13    21    34    55
```

Conclusion

The program correctly finds the first 10 Fibonacci numbers.

Exercise Problem

Write a program to find the n^{th} Fibonacci number. Run the program for $n = 15$.

Practical No. 14

Aim

To write a MATLAB program to find triangle numbers.

Problem

Write a program to determine the first 10 triangle numbers.

Theory

An integer is called a triangle number if it is the sum of consecutive natural numbers.

Algorithm

Step 1: Set the number $n = 10$;

Step 2: Compute the triangle number using the function `tringlenumbers(n)`;

Step 3: Define the function `trianglenumbers(n)`.

Program

```
clc;
clear;
n = 10;
result = trianglenumbers(n);
fprintf('First %d triangle numbers are:\n', n);
disp(result);
function result = trianglenumbers(n)
    result = [];
    sum = 0;
    for i=1:n
        sum = sum+i;
        result = [result sum];
    end
end
```

Output

```
First 10 triangle numbers are:
     1     3     6    10    15    21    28    36    45    55
```

Conclusion

The program correctly finds the first 10 triangle numbers.

Exercise Problem

Write a program which checks if a given number n is a triangle number or not.

Practical No. 15

Aim

To write a MATLAB program to compute the Lucas numbers.

Problem

Write a program to compute the first 10 Lucas numbers.

Theory

The Lucas numbers are the members of the sequence defined by $a_1 = 2$, $a_2 = 1$ and

$$a_n = a_{n-1} + a_{n-2}$$

. The first few members of the sequence are $2, 1, 3, \ldots$.

Algorithm

Step 1: Set the number $n = 10$;

Step 2: Compute the Lucas number using the user define command `lucasnumbers(n)`;

Step 3: Define a user function by `lucasnumbers(n)`.

Program:

```
clc;
clear;
n = 10;
result = lucasnumbers(n);
fprintf('First %d Lucas numbers are:\n', n);
disp(result);
function result = lucasnumbers(n)
    result = [2, 1];
    if(n > 2)
```

```
        for i = 3:n
            result = [result, result(end) + result(end-1)];
        end
    end
end
```

Output:

```
First 10 Lucas numbers are:
     2     1     3     4     7    11    18    29    47    76
```

Conclusion:

The program successfully computes the first 10 Lucas numbers.

Exercise Problem

Write a program to find the n^{th} Fibonacci number. Run the program for $n = 15$.

Practical No. 16

Aim

To write a MATLAB program to find nth Fermat numbers.

Problem

Write a program to compute first 5 Fermat numbers.

Theory

A Fermat number is defined as $k = 2^{2^n} + 1$, where n is 0, 1, 2,. . . .

Algorithm

```
Step 1: Set $n = 1 : 5$;
Step 2: Compute the first five Fermat numbers using the formula $k = 2^{2^n} + 1$;
Step 3: Output the results using the command disp.
```

Program

```
clc;
clear;
n = 5;
result = fermatnumbers(n);
fprintf('First %d Fermat numbers are:\n', n);
for i = 1:n
    fprintf('\t%d', result(i));
end
fprintf('\n');
function result = fermatnumbers(n)
    result = sym(2.^(2.^(1:n)) + 1);
end
```

Output

```
First 5 Fermat numbers are:
 5 17 257 65537 4294967297
```

Conclusion

The program successfully computes the 9th Fermat number.

Exercise Problem

Write a program to find the n^{th} Fermat number. Run the program for $n = 8$.

Practical No. 17

Aim

To write a MATLAB program to check the Pythagorean triplet.

Problem

Write a program to check whether a given triplet $(3,\ 4,\ 5)$ is Pythagorean triplets or not.

Theory

The Pythagorean triplet $(a,\ b,\ c)$ are the positive integers such that

$$a^2 + b^2 = c^2.$$

Algorithm

Step 1: Input a, b and c;

Step 2: Check if a, b and c satisfy $a^2 + b^2 = c^2$;

Step 3: Output the result using the command `fprintf`.

Program

```
clc;
clear;
a = input("Enter the value a: ");
b = input("Enter the value b: ");
c = input("Enter the value c: ");
if a^2 + b^2 == c^2
    fprintf('(%d, %d, %d) is a Pythagorean triplet.\n', a, b, c);
else
    fprintf('(%d, %d, %d) is a NOT Pythagorean triplet.\n', a, b, c);
end
```

Output

```
Enter the value a: 3
Enter the value b: 4
Enter the value c: 5
(3, 4, 5) is a Pythagorean triplet.
```

Conclusion

The Pythagorean triplet can be easily verified through the above program.

Exercise Problem

Write a program to decompose a non-negative integer n in four squares. Run the program for $n = 23$.

Practical No. 18

Aim

To write a MATLAB program to verify a pair of integers (a, b) for amicability.

Problem

Write a program to check whether a pair $(a,\ b)$ of integers is amicable.

Theory

A pair (a, b) is said to be amicable if sum of proper divisors of a (except a) is equal to b and sum of proper divisors of b (except b) is equal to a.

Algorithm

Step 1: Input a and b;

Step 2: Set s_1 = the sum of all divisors of a except a;

Step 3: Set s_2 = the sum of all divisors of a except b;

Step 4: Compute whether $a = s_2$ and $b = s_1$ using `if` and `else` command;

Step 5: Output the result using the command `fprintf`.

Program

```
clc;
clear;
a = input("Enter the value of a: ");
b = input("Enter the value of b: ");
s1 = sum(divisors(a)) - a;
s2 = sum(divisors(b)) - b;
if s2==a && s1==b
    fprintf('(%d, %d) are amicable numbers\n', a, b);
else
    fprintf('(%d, %d) are NOT amicable numbers\n', a, b);
end
```

Output

```
Enter the value of a: 220
Enter the value of b: 284
(220, 284) are amicable numbers
```

Conclusion

The program successfully verifies whether the pair $(a,\ b)$ is amicable or not.

Exercise Problem

Write a program to check whether a pair $(a,\ b)$ of amicable integers is regular or exotic.

Practical No. 19

Aim

To write a program to find Legendre symbol.

Problem

Write a program to find Legendre symbol of quadratic residue 4 modulo 17.

Theory

The Legendre symbol is a multiplicative function with values $1,\ -1,\ 0$ that is a quadratic character modulo an odd prime p. Its value at a quadratic residue is 1 and -1 at non residue quadratic character modulo p. Its value at 0 is 0.

Algorithm:

Step 1: Input a and p;

Step 2: Compute the result using `Legendresymbol(a, p)`;

Step 3: Define the function `Legendresymbol(a, p)`.

Program

```
clc;
clear;
a = input("Enter the value of a: ");
p = input("Enter the value of p: ");
result = Legendresymbol(a, p);
fprintf('LegendreSymbol(%d, %d) = %d.\n', a, p, result);

function result = Legendresymbol(a, p)
    if (p <= 1) || ~isprime(p)
     error('p must be a prime.')
    else
        if a == 0
            result = 0;
            return;
        end

        if p == 2
            if mod(a, 2) == 1
                result = 1;
            else
                result = 0;
            end
        else
            if mod(a, p) == 0
                result = 0;
            else
                if powermod(a,(p-1)/2, p) == 1
                    result = 1;
                else
                    result = -1;
                end
            end
        end
    end
end
```

Output

```
Enter the value of a: 4
Enter the value of p: 17
LegendreSymbol(4, 17) = 1.
```

Conclusion

The program correctly finds the Legendre symbol of quadratic character of a modulo p.

Exercise Problem

Write a program to find Legendre symbol of quadratic residue 6 modulo 7.

Chapter 5

Numerical Analysis

Practical No. 1

Aim

To find the root of an equation using **Mathematica** with various numerical techniques.

Problem

Find the root of the following polynomial in the interval [1,2] by using bisection method correct up to three decimal places.

$$x^3 + 4x^2 - 10 = 0$$

.

Algorithm

Step1: Define the function $f(x) = x^3 - 4x - 9 = 0$ and let $a = 1$ and $b = 2$;
Step2: Check $f(a)f(b) < 0$. If this is the case then there is a root in $[a, b]$;
Step3: Calculate $c = \frac{a+b}{2}$;
Step4: If $f(c) = 0$, the c is a root of polynomial f. Stop and return c;
Step5: $sign(f(a)) \neq sign(f(c))$ the assign $b = c$, else $a = c$;
Step6: Repeat steps 3-5 until the convergence is achieved i.e.$tol < 10^{-4}$;
Absolute error (tol) up to three decimal is obtained as: $tol =\mid b - a \mid< 10^{-4}$.

Program

```
In[18]:= x0 = 1;
x1 = 2.0;
Nmax = 20;
```

```
eps = 0.0001;

f[x_] := x^3 + 4 + x^2 - 10;

If[N[f[x0] * f[x1]] > 0,
   Print["These values do not satisfy the IVP so change the values."],

   For[i = 1, i <= Nmax, i++,
      a = (x0 + x1)/2;

      If[Abs[(x1 - x0)/2] < eps,
         Return[a],
         Print[i, "th iteration value is : ", a];
         Print["Estimated error in ", i, "th iteration is : ", (x1 - x0)/2];
      ];

      If[f[a] + f[x1] > 0, x1 = a, x0 = a]
   ]
];

Print["Root is : ", a];
Print["Estimated error in ", i, "th iteration is : ", (x1 - x0)/2];
```

Output

```
1th iteration value is : 1.5
Estimated error in 1th iteration is : 0.5
2th iteration value is : 1.75
Estimated error in 2th iteration is : 0.25
3th iteration value is : 1.625
Estimated error in 3th iteration is : 0.125
4th iteration value is : 1.5625
Estimated error in 4th iteration is : 0.0625
5th iteration value is : 1.53125
Estimated error in 5th iteration is : 0.03125
6th iteration value is : 1.546875
Estimated error in 6th iteration is : 0.015625
7th iteration value is : 1.5390625
Estimated error in 7th iteration is : 0.0078125
8th iteration value is : 1.53515625
Estimated error in 8th iteration is : 0.00390625
9th iteration value is : 1.537109375
Estimated error in 9th iteration is : 0.001953125
10th iteration value is : 1.5380859375
```

```
Estimated error in 10th iteration is : 0.0009765625
11th iteration value is : 1.53759765625
Estimated error in 11th iteration is : 0.00048828125
12th iteration value is : 1.537841796875
Estimated error in 12th iteration is : 0.000244140625
13th iteration value is : 1.5377197265625
Estimated error in 13th iteration is : 0.0001220703125
14th iteration value is : 1.53765869140625
Estimated error in 14th iteration is : 6.103515625e-05

Root is : 1.53765869140625
Estimated error in 14th iteration is : 6.103515625e-05
```

Practical No. 2

Aim

To find the root of an equation using **Mathematica** with various numerical techniques.

Problem

Find the root of the following polynomial in the interval [1, 2] by using bisection method correct up to three decimal places

$$\sqrt{x} - cos(x) = 0$$

.

Algorithm

Step1: Define the function $f(x) = \sqrt{x} - cos(x) = 0$ and let$a = 1$ and $b = 2$;
Step2: Check $f(a)f(b) < 0$. If this is the case then there is a root in [a,b];
Step3: Calculate $\frac{a+b}{2}$;
Step4: If $f(c) = 0$, the c is a root of polynomial f.Stop and return c;
Step5: If $sign(f(a)) \neq sign(f(c))$ the assign $b = c$,else $a = c$;
Step6: Repeat Steps 3-5 until the convergence is achieved i.e.$tol < 10^{-4}$;
Absolute error (tol) up to three decimal is obtained as: $tol = \mid b - a \mid < 10^{-4}$.

Program

```
In[28]:= (* Bisection Method: By Constructing the Function *)
bisection[f_, x0_, x1_, Nmax_, eps_] :=
 If[N[f[x0] * f[x1], 16] > 0,
  Print["These values do not satisfy the IVP so change the values."],
  y0 = x0;
  y1 = x1;

  For[i = 1, i <= Nmax, i++,
    a = (y0 + y1)/2;

    If[Abs[(y1 - y0)/2] < eps,
      Return[N[a, 8]],
      Print[i, "th iteration value is : ", N[a, 16]];
      Print["Estimated error is : ", N[(y1 - y0)/2, 8]];
    ];

    If[f[a] * f[y1] > 0, y1 = a, y0 = a];
  ];

  Print["Root is : ", N[a, 8]];
  Print["Estimated error is : ", N[(y1 - y0)/2, 8]];
]

f[x_] := Sqrt[x] - Cos[x];

bisection[f, 1, 2, 20, 0.0001]
```

Output

```
1th iteration value is : 1.250000000000000
Estimated error is : 0.75000000
2th iteration value is : 0.8750000000000000
Estimated error is : 0.37500000
3th iteration value is : 0.6875000000000000
Estimated error is : 0.18750000
4th iteration value is : 0.5937500000000000
Estimated error is : 0.09375000
5th iteration value is : 0.6406250000000000
Estimated error is : 0.04687500
6th iteration value is : 0.6640625000000000
Estimated error is : 0.02343750
7th iteration value is : 0.6523437500000000
Estimated error is : 0.01171875
```

```
8th iteration value is : 0.6464843750000000
Estimated error is : 0.00585938
9th iteration value is : 0.6435546875000000
Estimated error is : 0.00292969
10th iteration value is : 0.6420898437500000
Estimated error is : 0.00146484
11th iteration value is : 0.6413574218750000
Estimated error is : 0.00073242
12th iteration value is : 0.6417236328125000
Estimated error is : 0.00036621
13th iteration value is : 0.6415405273437500
Estimated error is : 0.00018311
14th iteration value is : 0.6416320800781250
Estimated error is : 0.00009155

Root is : 0.64163208
Estimated error is : 0.00009155
```

Practical No. 3

Aim

To find the root of an equation using **Mathematica** with various numerical techniques.

Problem

Write a Mathematica program to approximate $\sqrt{3}$ correct upto two decimal places using Bisection Algorithm.

Algorithm

Note that $\sqrt{3}$ is the only root of $f(x) = x^2 - 3$ in $[1, 2]$;
Step1: Let $a = 1$ and $b = 2$;
Step2: Check $f(a)f(b) < 0$.If this is the case then there is a root in $[a, b]$;
Step3: Calculate $\frac{a+b}{2}$;
Step4: If $f(c) = 0$, the c is a root of polynomial f. Stop and return c;
Step5: If $sign(f(a)) \neq sign(f(c))$ the assign $b = c$, else $a = c$;
Step6: Repeat Steps 3-5 until the convergence is achieved i.e., $tol < 10^{-3}$.
Absolute error (tol) up to three decimal is obtained as: $tol =| b - a |< 10^{-3}$.

Program

```
x0 = Input["Enter first guess "];
```

```
x1 = Input["Enter second guess "];
Nmax = Input["Enter maximum number of iterations : "];
eps = Input["Enter a value of convergence parameter : "];

Print["x0 = ", x0];
Print["x1 = ", x1];
Print["Nmax = ", Nmax];
Print["epsilon = ", eps];

f[x_] := x^3 - 3;

Print["f(x) = x^3 - 3"];

If[N[f[x0] * f[x1]] > 0,
   Print["These values do not satisfy the IVP so change the values."],

   For[i = 1, i <= Nmax, i++,
      a = (x0 + x1)/2;

      If[Abs[(x1 - x0)/2] < eps, Return[N[a, 16]]];

      If[f[a] * f[x1] > 0, x1 = a, x0 = a];

      Print[i, "th iteration value is : ", N[a, 16]];
      Print["Estimated error is : ", N[x1 - x0, 16]];
   ];

   Print["Root is : ", N[a, 16]];
   Print["Estimated error is : ", N[x1 - x0, 16]];
];

Plot[f[x], {x, -1, 3}]
```

Output

```
x0 = 1;
x1 = 2;
Nmax = 20;
epsilon = 0.001;

f[x_] := -3 + x^2;

For[i = 1, i <= Nmax, i++,
   a = (x0 + x1)/2;
   If[Abs[(x1 - x0)/2] < epsilon, Return[N[a, 16]]];

   Print[i, "th iteration value is : ", N[a, 16]];
   Print["Estimated error is : ", N[(x1 - x0)/2, 16]];

   If[f[a] * f[x1] > 0, x1 = a, x0 = a];
];

Return[N[a, 16]];
```

Practical No. 4

Aim

To find the root of an equation using **Mathematica** with various numerical techniques.

Problem

Write a *Program* to find the first approximation to the root of the function $f(x) = sin(x) - x$ in the interval $[1, 2]$ using Secant method.

Algorithm

Step1: Choose initial guesses x_0 and x_1 such that $x_0 \neq x_1$.
Step2: Evaluate x_2 as

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

.

Program

```
x0 = 2;
x1 = 3;

f[x_] := x^2 - 6;

Print["f(x) = ", f[x]];

x2 = N[x1 - (f[x1] / ( (f[x] /. x -> x1) - (f[x] /. x -> x0) ) * (x1 - x0) )];

Print["Root is : ", x2];
```

Output

```
f(x) = x^2 - 6
Root is : 2.4000000000000000
```

Practical No. 5

Aim

To find the root of an equation using **Mathematica** with various numerical techniques.

Problem

Write a program to find the root of the function $f(x) = x^4 - x - 10 = 0$ in the interval [1,2] correct upto three decimal places using Secant method.

Algorithm

Step1: Choose initial guesses x_0 and x_1 such that $x_0 \neq x_1$.
Step2: Evaluate nth approximation x_n as,

$$x_n = \frac{x_{n-2}f(x_{n-1}) - x_{n-1}f(x_{n-2})}{f(x_{n-1}) - f(x_{n-2})}$$

Step3: If $f(x_n) = 0$ then x_n is an exact root, else $x_{n-2} = x_{n-1}$ and $x_{n-1} = x_n$.
Step4: Repeat steps 2 & 3 until $f(x_n) = 0$ or $\mid f(x_n) \mid \leq tolerance (= 10^{-4})$.

Program

```
 x0 = 1;
x1 = 2;
Nmax = 20;
eps = 0.0001;

f[x_] := x^4 - x - 10;

Print["f(x) := ", f[x]];

For[i = 1, i <= Nmax, i++,
   x2 = N[x1 - (f[x1] * (x1 - x0)) / ((f[x] /. x -> x1) - (f[x] /. x -> x0))];

   If[Abs[x1 - x2] < eps, Return[x2], x0 = x1; x1 = x2];
   Print["In ", i, "th Number of iterations the root is : ", x2];
   Print["Estimated error is : ", Abs[x1 - x0]];
];

Print["Root is : ", x2];
Print["Estimated error is : ", Abs[x2 - x1]];
```

Output

```
f(x) = x^4 - x - 10

In 1th iteration, the root is : 1.7142857142857100
Estimated error is : 0.2857142857142860

In 2th iteration, the root is : 1.8385312463222300
Estimated error is : 0.1242455320365170

In 3th iteration, the root is : 1.8577757949191400
Estimated error is : 0.0192445485969133

In 4th iteration, the root is : 1.8555528651416900
Estimated error is : 0.0022229297774552

Root is : 1.8555844703303500
Estimated error is : 0.0000316051886582
```

Practical No. 6

Aim

To find the root of an equation using **MATLAB** with various numerical techniques.

Problem

Write a program to find the roots of polynomial $P(x) = x^2 - 2$.

Algorithm

Step1: Store the coefficients of the polynomial as a vector.
Step2: Evaluate roots use roots function.
Step3: fprintf is used to print the values of r.

Program

```
% Calculates the roots of a single-variable polynomial
% Numeric Roots

% Create a vector to represent the polynomial x^2 - 2
p = [1 0 -2];

% Calculate the roots
r = roots(p);

% Display the roots
fprintf('The root of the equation is: %.3f\n', r);
```

Output

```
The root of the equation is: 1.414
The root of the equation is: -1.414
```

Practical No. 7

Aim

To find the root of an equation using **MATLAB** with various numerical techniques.

Problem

Write a program to find a root of the equation by using Newton-Raphson method correct up to five decimal places.

$$x^3 - 4x - 9 = 0$$

Algorithm

Step1: Choose an initial approximation x_0 for the root.
Step2: For each iteration $n = 0, 1, 2, \ldots\ldots\ldots$:

- Evaluate the function $f(x_n)$ and its derivative $f'(x_n)$

- Update the approximation (x_{n+1}) using the formula:

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

Step3: Stop if $\mid x_{n+1} - x_n \mid < 10^{-6}$

Program

```
function x = newton_raphson
   % Ask the user to input the function as a string
   func_str = input('Enter the function in terms of x: ', 's');

   % Convert the input string to a function handle
   f = str2func(['@(x) ' func_str]);

   % Ask the user to input the derivative of the function as a string
   df_str = input('Enter the derivative of the function in terms of x: ', 's');

   % Convert the input string to a function handle for the derivative
   df = str2func(['@(x) ' df_str]);

   % Ask the user to input the initial guess
   x0 = input('Enter the initial guess: ');

   % Define the tolerance for convergence (five decimal places)
   tol = 1e-6; % This ensures that the result is accurate to five decimal places

   % Initialize the current value
   x = x0;

   % Maximum number of iterations to prevent infinite loop
   max_iter = 1000;
   iter = 0;

   % Newton-Raphson method loop
   while true
       % Evaluate function and its derivative at the current value
       fx = f(x);
       dfx = df(x);

       % Check if the derivative is zero to avoid division by zero
       if dfx == 0
           error('Derivative is zero. The method fails.');
       end

       % Calculate the next value using the Newton-Raphson formula
       x_next = x - fx / dfx;
```

```
        % Check for convergence
        if abs(x_next - x) < tol
            break;
        end

        % Update the current value
        x = x_next;

        % Increment the iteration counter
        iter = iter + 1;

        % Check for maximum iterations
        if iter >= max_iter
            error('Maximum number of iterations reached. The method may not be converging.'
        end
    end

    % Display the result
    fprintf('The root is approximately: %.5f\n', x);
end
```

Output

Now by using the function newton_raphson we will take input of function and its derivative from the user. Also, the initial approximation is given by the user.

```
x = newton_raphson
Enter the function in terms of x: x^3 - 4*x - 9
Enter the derivative of the function in terms of x: 3*x^2 - 4
Enter the initial guess: 0.5
The root is approximately: 2.70653

x =
    2.7065
```

Practical No. 8

Aim

To find the root of an equation using **Mathematica** with various numerical techniques.

Problem

Write a program to find the root of the equation by using the fixed point iteration method.

$$e^x - 3x^2 = 0$$

Algorithm

Step1: Rewrite the equation $f(x) = 0$ to $x = g(x)$. (So,$g(x) = x + e^x - 3x^2$).
Step2: Select an initial guess x_0.
Step3: Iterate using the formula:

$$x_{n+1} = g(x_n)$$

Program

```
g[x_] := Exp[x]/3;
x0 = 0.5;
tol = 10^(-6);
maxIter = 100;
iter = 0;

While[iter < maxIter,
  xn = g[x0];

  If[Abs[xn - x0] < tol, Return[xn];];

  x0 = xn;
  iter++;
];

Print["Maximum iterations reached without convergence and the root is ", xn];
```

Output

```
Return[0.5495737569000427]
"Maximum iterations reached without convergence and the root is " 0.5495737569000
```

Practical No. 9

Aim

To find the root of an equation using **MATLAB** with various numerical techniques.

Problem

Solve $\frac{dy}{dx} = y + x$ wih $y(0) = 1$ using Euler's Method.

Algorithm

Step1: Given $y'(x) = f(x, y)$ with initial condition $y(x_0) = y_0$.
Step2: Choose a step size h.
Step3: Iterate using:

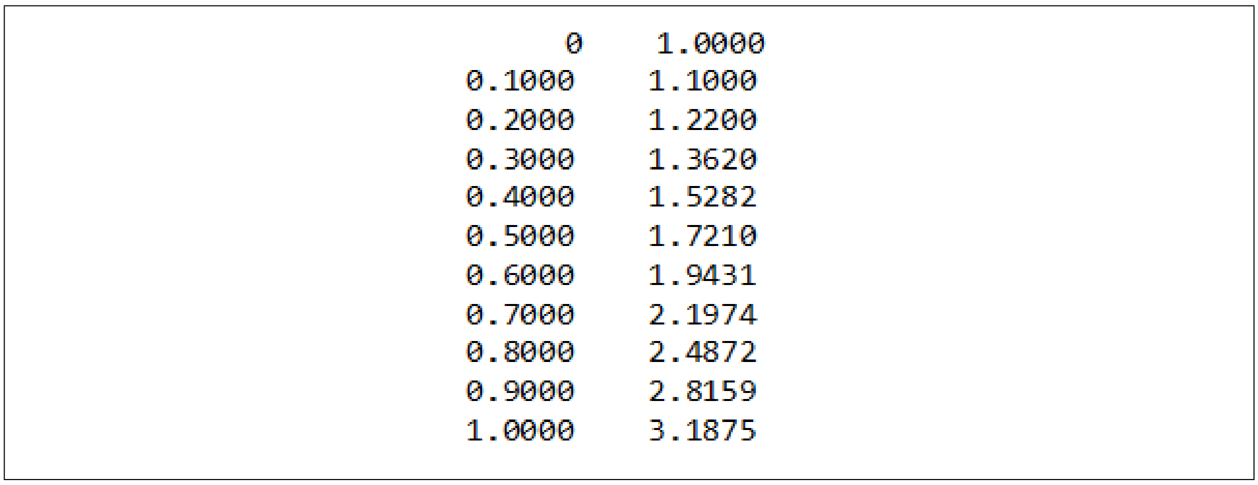
$$y_{n+1} = y_n + hf(x_n, y_n)$$

Program

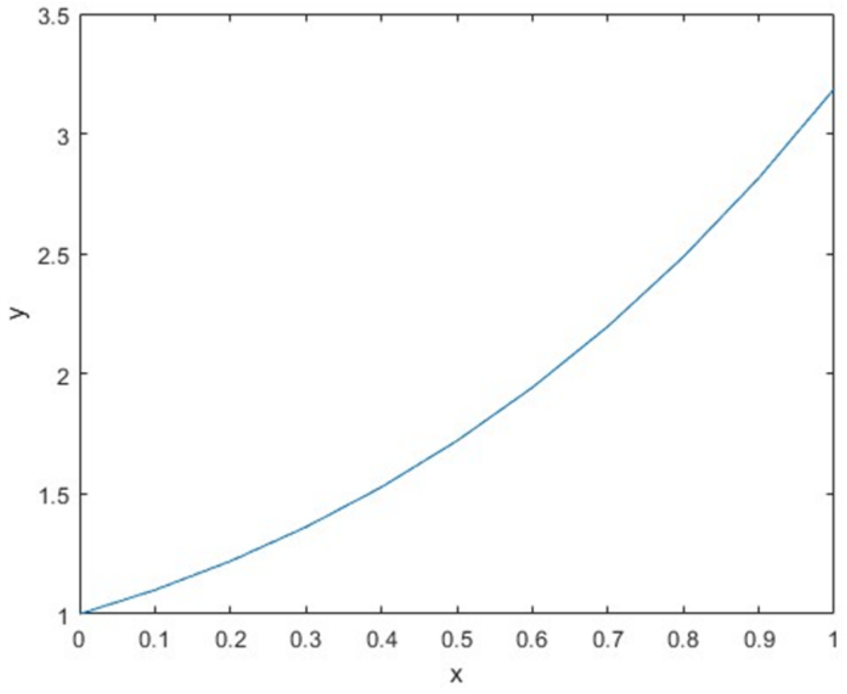
```
 % Euler's Method
f = @(x, y) y + x; % Given function on [0,1]

% Initial conditions
y0 = 1;
x0 = 0;
endX = 1;
h = 0.1; % Step size

% Number of iterations
n = round((endX - x0) / h);

% Initialize variables
x = x0;
y = y0;
sol = zeros(n+1, 2); % Store (x, y) values
sol(1, :) = [x, y];

% Euler's method loop
for i = 1:n
    y = y + h * f(x, y); % Euler formula: y(n+1) = y(n) + h*f(x(n), y(n))
    x = x + h; % Increment x by step size
    sol(i+1, :) = [x, y]; % Store new (x, y)
end

% Display solution table
```

```
disp(sol)

% Plot the numerical solution
plot(sol(:,1), sol(:,2), '-o')
xlabel('x')
ylabel('y')
title('Euler''s Method Solution')
grid on
```

Output



```
        0    1.0000
   0.1000    1.1000
   0.2000    1.2200
   0.3000    1.3620
   0.4000    1.5282
   0.5000    1.7210
   0.6000    1.9431
   0.7000    2.1974
   0.8000    2.4872
   0.9000    2.8159
   1.0000    3.1875
```


3.5
3
2.5
2
1.5
1
y
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
x

Practical No. 10

Aim

To find the root of an equation using **MATLAB** with various numerical techniques.

Problem

Solve $\frac{dy}{dx} = ysin(x)$ with $y(0) = 1$ using Heun's Method.

Algorithm

Step1: Given $y'(x) = f(x, y)$ with initial condition $y(x_0) = y_0$.
Step2: Choose a step size h.
Step3: Iterate using:

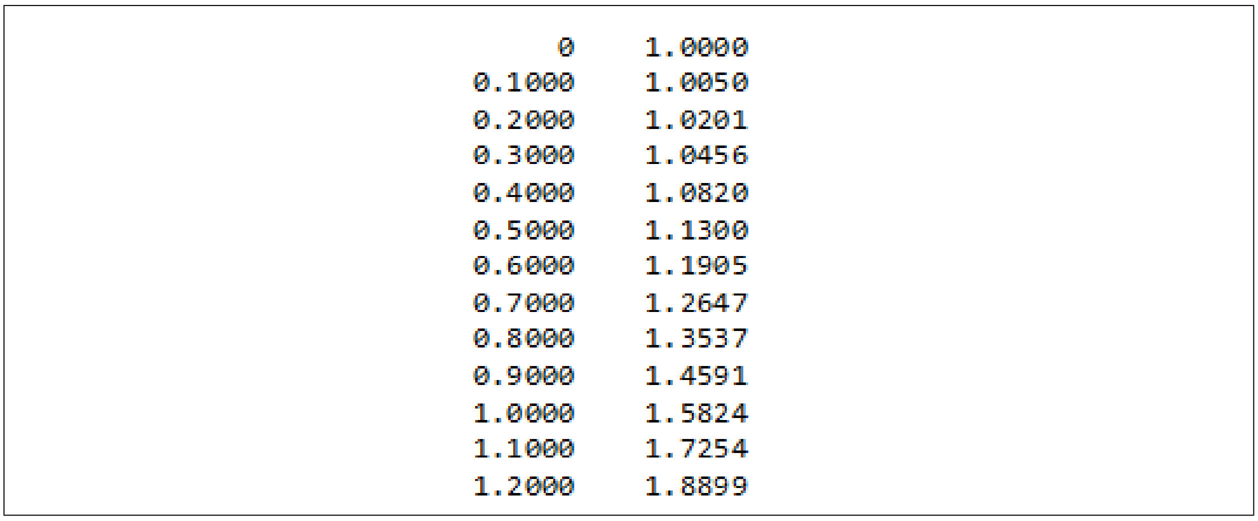
$$y_{pred} = y_n + hf(x_n, y_n)$$

$$y_{n+1} = y_n + \frac{h}{2}((f(x_n, y_n) + f(x_n + h, y_{pred}))$$

Program

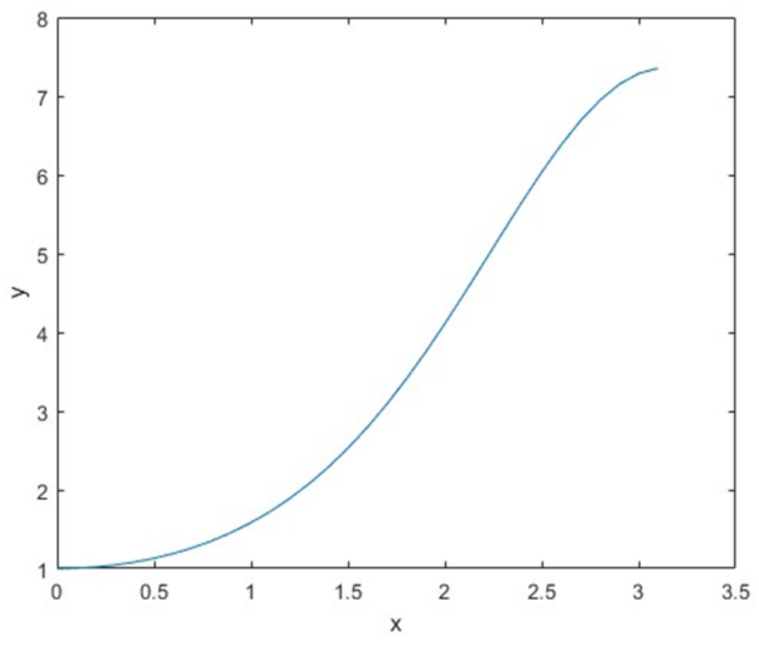
```
% Heun's Method
f = @(x, y) y * sin(x); % Given function on [0, pi]

% Initial conditions
y0 = 1;
x0 = 0;
endX = pi;
h = 0.1; % Step size

% Number of iterations
n = round((endX - x0) / h);

% Initialize variables
x = x0;
y = y0;
sol = zeros(n+1, 2); % Store (x, y) values
sol(1, :) = [x, y];

% Heun's method loop
for i = 1:n
    yPred = y + h * f(x, y); % Predictor step (Euler's method)
    y = y + h/2 * (f(x, y) + f(x + h, yPred)); % Corrector step (Heun's update)
    x = x + h; % Increment x by step size
    sol(i+1, :) = [x, y]; % Store new (x, y)
end

% Display solution table
disp(sol)

% Plot the numerical solution
plot(sol(:,1), sol(:,2), '-o')
xlabel('x')
ylabel('y')
title('Heun's Method Solution')
grid on
```

Output



```
         0    1.0000
    0.1000    1.0050
    0.2000    1.0201
    0.3000    1.0456
    0.4000    1.0820
    0.5000    1.1300
    0.6000    1.1905
    0.7000    1.2647
    0.8000    1.3537
    0.9000    1.4591
    1.0000    1.5824
    1.1000    1.7254
    1.2000    1.8899
```

```
          1.3000    2.0775
          1.4000    2.2898
          1.5000    2.5281
          1.6000    2.7931
          1.7000    3.0851
          1.8000    3.4032
          1.9000    3.7456
          2.0000    4.1092
          2.1000    4.4895
          2.2000    4.8804
          2.3000    5.2744
          2.4000    5.6625
          2.5000    6.0346
          2.6000    6.3800
          2.7000    6.6878
          2.8000    6.9475
          2.9000    7.1498
          3.0000    7.2870
          3.1000    7.3538
```



Practical No. 11

Aim

To find the root of an equation using **MATLAB** with various numerical techniques.

Problem

Solve the system of ODEs in $[0, T]$

$$\frac{dx}{dt} = a(y-x) \quad with \quad x(0) = 1,$$
$$\frac{dy}{dt} = x(b-z) - y \quad with \quad y(0) = 1,$$
$$\frac{dz}{dt} = xy - cz \quad with \quad z(0) = 1,$$

where, $a = 10, b = 28, and \quad c = \frac{8}{3}$.

Algorithm

Step1: Let $\frac{dx}{dt} = f(x,y,z) \quad with \quad x(0) = x_0, \frac{dy}{dt} = g(x,y,z) \quad with \quad y(0) = y_0, \quad and \quad \frac{dz}{dt} = k(x,y,z) \quad with \quad z(0) = z_0$.
Step2: Choose a time step size dt, then time $t_n = t_0 + dt$, where $t_0 = 0$.
Step3: Iterate for $n = 1, 2, ...$ Until the time T is reached using:

$$x_{n+1} = x_n + dtf(x_n, y_n, z_n)$$
$$y_{n+1} = y_n + dtg(x_n, y_n, z_n)$$
$$z_{n+1} = z_n + dtk(x_n, y_n, z_n)$$

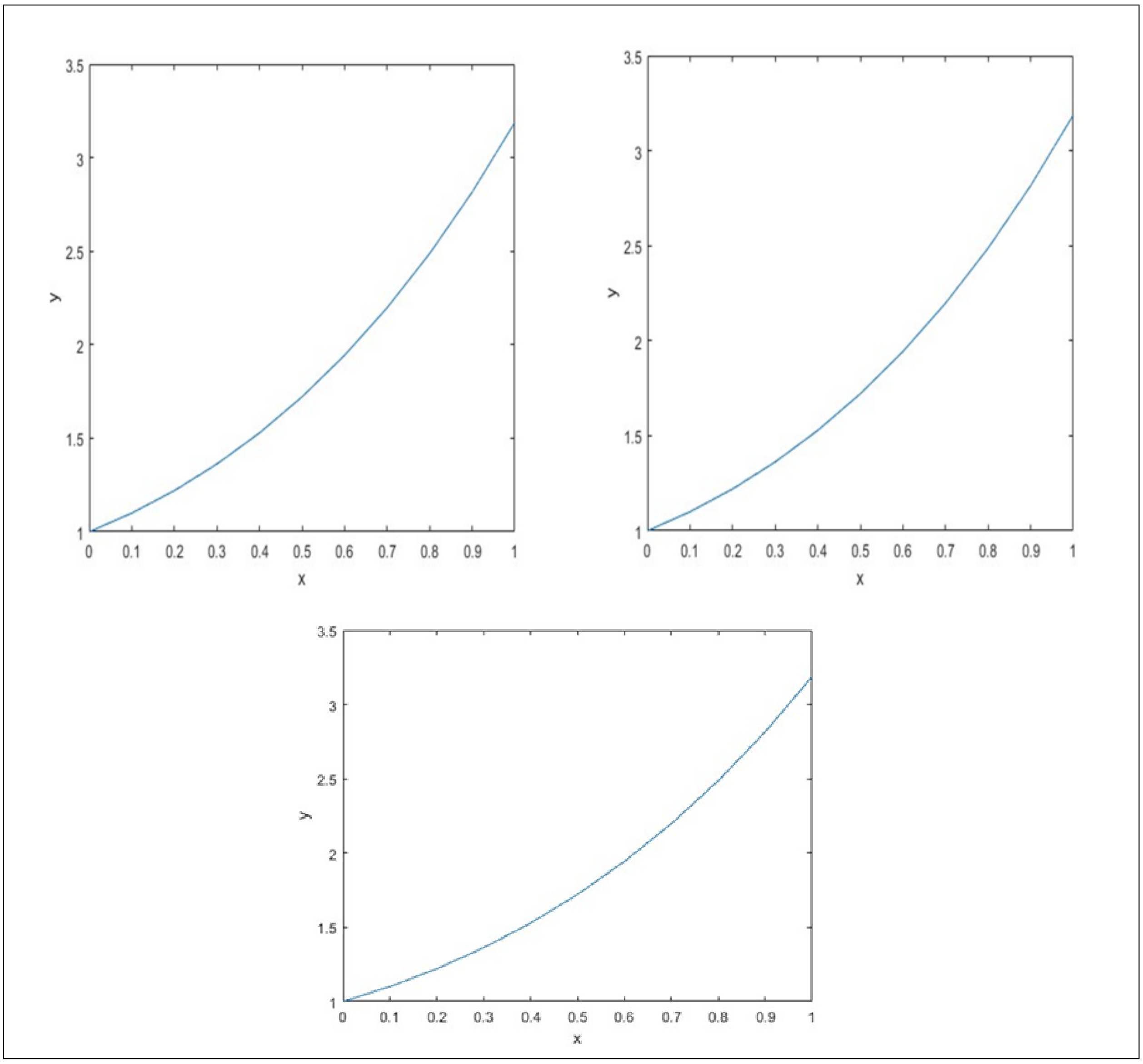
with $x_0 = 1, y_0 = 1, z_0 = 1$.

Program

```
% Parameters
a = 10;
b = 28;
c = 8/3;

% System of equations
f = @(x, y, z) a * (y - x);
g = @(x, y, z) x * (b - z) - y;
h = @(x, y, z) x * y - c * z;

% Initial conditions and time parameters
```

```
initialConditions = [1, 1, 1];
t0 = 0;
T = 10;
dt = 0.01;

% Initialize variables
x = initialConditions(1);
y = initialConditions(2);
z = initialConditions(3);
t = t0;

n = round((T - t0) / dt); % Number of time steps
sol = zeros(n+1, 4); % Store (t, x, y, z) values
sol(1, :) = [t, x, y, z];

% Euler method loop
for i = 1:n
    x = x + dt * f(x, y, z);
    y = y + dt * g(x, y, z);
    z = z + dt * h(x, y, z);
    t = t + dt;
    sol(i+1, :) = [t, x, y, z];
end

% Plotting results
figure(1)
plot(sol(:,1), sol(:,2), ‘b’)
xlabel(‘t’)
ylabel(‘x’)
title(‘x vs t’)
grid on

figure(2)
plot(sol(:,1), sol(:,3), ‘r’)
xlabel(‘t’)
ylabel(‘y’)
title(‘y vs t’)
grid on

figure(3)
plot(sol(:,1), sol(:,4), ‘g’)
xlabel(‘t’)
ylabel(‘z’)
title(‘z vs t’)
grid on
```

Output



Practical No. 12

Aim

To find the root of an equation using **MATLAB** with various numerical techniques.

Problem

Solve the following ordinary differential equation using Runge-Kutta (RK4) Method.

$$\frac{dy}{dt} = y - t^2 + 1 \quad with \quad y(0) = 0.5 \quad in \quad [0, T]$$

Algorithm

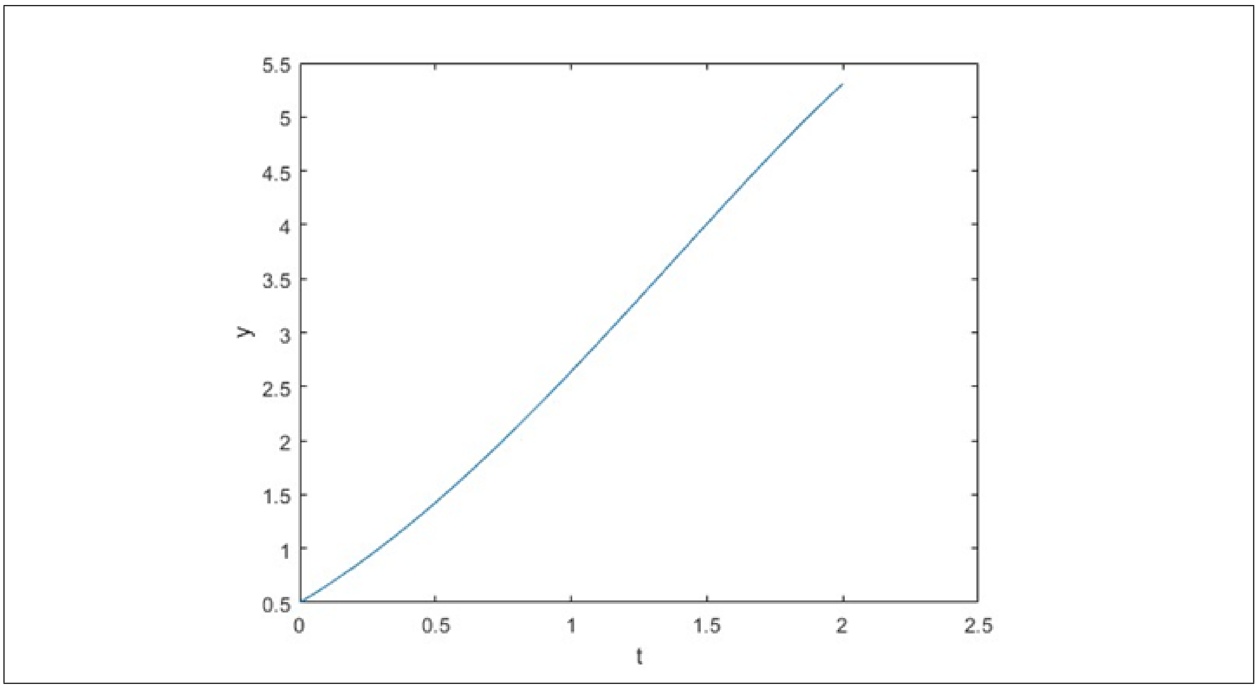
Step1: Let $\frac{dy}{dt} = f(t, y)$ with initial condition $y(t_0) = y_0$.
Step2: Choose a step size dt.
Step3: Iterate using:

$$\begin{aligned} k_1 &= dtf(t_n, y_n) \\ k_2 &= dtf(t_n + \frac{dt}{2}, y_n + \frac{k_1}{2}) \\ k_3 &= dtf(t_n + \frac{dt}{2}, y_n + \frac{k_2}{2}) \\ k_4 &= dtf(t_n + dt, y_n + k_3) \\ y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned}$$

Program

```
% RK4 Method for ODE
f = @(t, y) y - t^2 + 1; % Given function
y0 = 0.5;                % Initial condition
t0 = 0;                  % Start time
T = 2;                   % End time
dt = 0.1;                % Step size

n = round((T - t0) / dt); % Number of iterations
t = t0;                  % Initialize time
y = y0;                  % Initialize solution
sol = zeros(n+1, 2);     % Store (t, y) values
sol(1, :) = [t, y];      % Store initial condition

% RK4 Loop
for i = 1:n
   k1 = dt * f(t, y);
   k2 = dt * f(t + dt/2, y + k1/2);
   k3 = dt * f(t + dt/2, y + k2/2);
   k4 = dt * f(t + dt, y + k3);
   y = y + (k1 + 2*k2 + 2*k3 + k4) / 6; % Update solution
   t = t + dt;                          % Increment time
   sol(i+1, :) = [t, y];                % Store new (t, y)
end

% Plot the numerical solution
plot(sol(:,1), sol(:,2), '-o')
xlabel('t')
ylabel('y')
title('RK4 Method Solution')
grid on
```

Output



Practical No. 13

Aim

To find the root of an equation using **MATLAB** with various numerical techniques.

Problem

Approximate the first derivative of $f(x) = sin(x)$ at $x_0 = \frac{\pi}{4}$ using the forward difference method with two different step sizes.

Algorithm

Step1: Select a step size h.
Step2: Evaluate first derivative using the formula $\frac{f(x_0+h)-f(x_0)}{h}$.
Step3: Error is obtained as the difference between the actual derivative and the approximated derivative given as $\mid Actual f'(x_0) - Approximate f'(x_0) \mid$.

Program

```
% Forward Difference Approximation with h = 0.01
h = 0.01;
f = @(x) sin(x); % Function
x0 = pi/4; % Point of differentiation
ForwardDerivative_1 = (f(x0 + h) - f(x0)) / h; % Forward difference formula
Error_1 = abs(ForwardDerivative_1 - cos(x0)); % Error calculation

% Display results for h = 0.01
fprintf('For h = 0.01:\n');
fprintf('Forward Derivative: %.6f\n', ForwardDerivative_1);
fprintf('Error: %.6f\n\n', Error_1);

% Forward Difference Approximation with h = 0.001
h = 0.001;
ForwardDerivative_2 = (f(x0 + h) - f(x0)) / h; % Forward difference formula
Error_2 = abs(ForwardDerivative_2 - cos(x0)); % Error calculation

% Display results for h = 0.001
fprintf('For h = 0.001:\n');
fprintf('Forward Derivative: %.6f\n', ForwardDerivative_2);
fprintf('Error: %.6f\n', Error_2);
```

Output

```
ForwardDerivative =    0.7036
Error  =    0.0035

ForwardDerivative = 0.7068
Error = 3.5367e-04
```

Practical No. 14

Aim

To find the root of an equation using **MATLAB** with various numerical techniques.

Problem

Approximate the first derivative of $f(x) = sin(x)$ at $x_0 = \frac{\pi}{4}$ using the backward difference method with two different step sizes.

Algorithm

Step1: Select a step size h.
Step2: Evaluate first derivative using the formula $\frac{f(x_0)-f(x_0-h)}{h}$.
Step3: Error is obtained as the difference between the actual derivative and the approximated derivative given as $\mid Actual f'(x_0) - Approximate f'(x_0) \mid$.

Program

```
% Backward Difference Approximation with h = 0.01
h = 0.01;
f = @(x) sin(x); % Function
x0 = pi / 4; % Point of differentiation
BackwardDerivative_1 = (f(x0) - f(x0 - h)) / h; % Backward difference formula
Error_1 = abs(BackwardDerivative_1 - cos(x0)); % Error calculation

% Display results for h = 0.01
fprintf('For h = 0.01:\n');
fprintf('Backward Derivative: %.6f\n', BackwardDerivative_1);
fprintf('Error: %.6f\n\n', Error_1);

% Backward Difference Approximation with h = 0.001
h = 0.001;
BackwardDerivative_2 = (f(x0) - f(x0 - h)) / h; % Backward difference formula
Error_2 = abs(BackwardDerivative_2 - cos(x0)); % Error calculation

% Display results for h = 0.001
fprintf('For h = 0.001:\n');
fprintf('Backward Derivative: %.6f\n', BackwardDerivative_2);
fprintf('Error: %.6f\n', Error_2);
```

Output

```
BackwardDerivative =    0.7106
Error  =    0.0035

BackwardDerivative = 0.7075
Error = 3.5344e-04
```

Practical No. 15

Aim

To find the root of an equation using and **MATLAB** with various numerical techniques.

Problem

Approximate the first derivative of $f(x) = sin(x)$ at $x_0 = \frac{\pi}{4}$ using the central difference method with two different step sizes and make a comparative conclusion as compare to forward and backward difference methods.

Algorithm

Step1: Select a step size h.
Step2: Evaluate first derivative using the formula $\frac{f(x_0+h)-f(x_0-h)}{2h}$.
Step3: Error is obtained as the difference between the actual derivative and the approximated derivative given as $\mid Actual f'(x_0) - Approximate f'(x_0) \mid$.

Program

```
% Central Difference Approximation with h = 0.01
h = 0.01;
f = @(x) sin(x); % Function
x0 = pi / 4; % Point of differentiation
CentralDerivative_1 = (f(x0 + h) - f(x0 - h)) / (2 * h); % Central difference for
Error_1 = abs(CentralDerivative_1 - cos(x0)); % Error calculation

% Display results for h = 0.01
fprintf('For h = 0.01:\n');
fprintf('Central Derivative: %.6f\n', CentralDerivative_1);
fprintf('Error: %.6f\n\n', Error_1);
```

```
% Central Difference Approximation with h = 0.001
h = 0.001;
CentralDerivative_2 = (f(x0 + h) - f(x0 - h)) / (2 * h); % Central difference formula
Error_2 = abs(CentralDerivative_2 - cos(x0)); % Error calculation

% Display results for h = 0.001
fprintf('For h = 0.001:\n');
fprintf('Central Derivative: %.6f\n', CentralDerivative_2);
fprintf('Error: %.6f\n', Error_2);
```

Output

```
CentralDerivative =    0.7071
Error  =    1.1785e-05

CentralDerivative = 0.7071
Error = 1.1785e-07
```

Conclusion

Note that by comparing the error in forward difference, backward difference and central difference, we can see that the error in central difference is least. Therefore, it can be concluded that the central difference is most accurate for this problem.

Practical No. 16

Aim

To find the root of an equation using and **MATLAB** with various numerical techniques.

Problem

Approximate the second derivative of $f(x) = sin(x)$ at $x_0 = \frac{\pi}{4}$ using the finite difference approximation with two different step sizes.

Algorithm

Step1: Choose a step size h.
Step2: Evaluate the second derivative using the formula $\frac{f(x_0+h)-2f(x_0)+f(x_0-h)}{2h}$.

Step3: Error is obtained as the difference between the actual derivative and the approximated derivative given as $| Actual f'(x_0) - Approximate f'(x_0) |$.

Program

```
% Finite Difference for Second Derivative with h = 0.01
h = 0.01;
f = @(x) sin(x); % Function
x0 = pi / 4; % Point of differentiation
SecondDerivative_1 = (f(x0 + h) - 2*f(x0) + f(x0 - h)) / (h * h); % Second deriva
Error_1 = abs(SecondDerivative_1 + sin(x0)); % Error calculation (exact second de

% Display results for h = 0.01
fprintf('For h = 0.01:\n');
fprintf('Second Derivative: %.6f\n', SecondDerivative_1);
fprintf('Error: %.6f\n\n', Error_1);

% Finite Difference for Second Derivative with h = 0.001
h = 0.001;
SecondDerivative_2 = (f(x0 + h) - 2*f(x0) + f(x0 - h)) / (h * h); % Second deriva
Error_2 = abs(SecondDerivative_2 + sin(x0)); % Error calculation

% Display results for h = 0.001
fprintf('For h = 0.001:\n');
fprintf('Second Derivative: %.6f\n', SecondDerivative_2);
fprintf('Error: %.6f\n', Error_2);
```

Output

```
SecondDerivative =  - 0.7071
Error  =    5.8925e-06

SecondDerivative = -0.7071
Error = 5.9011e-08
```

Conclusion

Note that by comparing the error it can be seen that the error will small step size is least.

Practical No. 17

Aim

To find the root of an equation using and **MATLAB** with various numerical techniques.

Problem

Interpolate the function $f(x) = sin(x)$ at $x = 1.5$ given points $(1, 0.8415)$ and $(2, 0.9093)$ using MATLAB in-build function *interp1*.

Working rule of interp1

The function interp1$(x, y, x\theta)$ returns interpolated values of a 1-D function at specific point x0 using linear interpolation. The vector x contains the sample points, and y contains the corresponding values, y(x).

Program

```
x = [1, 2];
y = [0.8415, 0.9093];
x0 = 1.5;
y_interp = interp1(x, y, x0);
fprintf('The interpolated value at x = %.2f is y = %.4f\n', x0, y_interp);
```

Output

```
SecondDerivative =  - 0.7071
Error  =    5.8925e-06

SecondDerivative = -0.7071
Error = 5.9011e-08
```

Conclusion

Note that by comparing the error it can be seen that the error will small step size is least.

Practical No. 18

Aim

To find the root of an equation using and **MATLAB** with various numerical techniques.

Problem

Interpolate the function $f(x) = sin(x)$ at $x = \frac{\pi}{4}$ using cubic spline interpolating polynomial for given points $(0, 0), (\frac{\pi}{2}, 1)$ and $(\pi, 0)$.

Algorithm

Step1: The function 'spline(x,y)' returns coefficients of the cubic spline polynomial.
Step2: 'ppval' takes the input polynomial coefficients as first argument and the query point as the second argument and returns the interpolated value as the query point.

Program

```
x = [0 pi/2 pi];
y = [0 1 0];
spline_coeffs = spline(x, y);
x_interp = pi/4;
y_interp = ppval(spline_coeffs, x_interp);
y_interp
error = abs(sin(x_interp) - y_interp);
```

Output

```
y_interp = 0.7500
error = 0.0429
```

Practical No. 19

Aim

To find the root of an equation using and **MATLAB** with various numerical techniques.

Problem

Compare the interpolating results obtained using linear and cubic spline interpolation for the function $f(x) = sin(x)$ at $x = \frac{\pi}{4}$ for given points $(0, 0), (\frac{\pi}{2}, 1)$ and $(\pi, 0)$.

Algorithm

Step1: Evaluate the linear interpoling value of $f(x)$ at the query point using 'interp1'. The vector x contains the sample points , and y contains the corresponding values, $y(x)$.
Step2:

- Obtain the cubic interpolation using the function 'spline(x,y)' that returns the coefficients of the cubic spline polynomial.
- 'ppval' takes the input polynomial coefficients as first argument and the query points as the second argument and returns the interpolated value as the query point.

Step3: Compare the results by obtaining the errors.

Program

```
x = [0 pi/2 pi]; % Sample points
y = [0 1 0]; % Function values at sample points
x_interp = pi/4; % Query point

% Linear interpolation
y_linear = interp1(x, y, x_interp);

% Spline interpolation
spline_coeffs = spline(x, y);
y_spline = ppval(spline_coeffs, x_interp);

% Errors
error_linear = abs(sin(x_interp) - y_linear);
error_spline = abs(sin(x_interp) - y_spline);

% Display results
fprintf('Linear Interpolation Value: %.6f\n', y_linear);
fprintf('Spline Interpolation Value: %.6f\n', y_spline);
fprintf('Error in Linear Interpolation: %.6f\n', error_linear);
fprintf('Error in Spline Interpolation: %.6f\n', error_spline);
```

Output

```
y_linear = 0.9549
y_spline = 0.7500
error_linear = 0.2478
error_spline = 0.0429
```

Conclusion

Note that the error with cubic spline interpolating polynomial is less.

Practical No. 20

Aim

To find the root of an equation using and **MATLAB** with various numerical techniques.

Problem

Write a program for solving integration by using trapezoidal rule taking h = 0.5, 0.25, 0.125, and also find error.

$$\int_0^1 \frac{dx}{1+x^2}$$

Algorithm

To solve this problem we will create a function file that will take all the inputs from the user.

- **Input the function, interval and step size:** The user is prompted to input the function as a string, which is then converted to a function handle.
  The user is asked to input the lower (a) and upper (b) limits of integration.
  The step sizes h is defined in an array.

- **Exact integral value:** The user is asked to input the exact integral value if known. If left empty, $exact\_value$ is set to NaN.
- **Trapezoidal rule calculation:** For each step size, the number of steps n is calculated.
  The sum for the trapezoidal rule is initialized and calculated in a loop.
  The boundary values are added, and the final integral value is computed.
  If the exact integral value is known, the error is calculated and printed along with the integral value. If not, only the integral value is printed.

Program

```
function trapezoidal_integration()
% Ask the user to input the function to integrate
func_str = input('Enter the function to integrate in terms of x: ', 's');
f = str2func(['@(x) ', func_str]);

% Ask the user to input the interval [a, b]
a = input('Enter the lower limit of integration (a): ');
```

```
    b = input('Enter the upper limit of integration (b): ');

    % Step size
    h = input('Enter the step size: ');

    % Exact integral value (if known, for error calculation)
    exact_value_str = input('Enter the exact integral value (if known), otherwise leave emp
    if isempty(exact_value_str)
        exact_value = NaN;
    else
        exact_value = str2double(exact_value_str);
    end

    % Number of steps
    n = (b - a) / h;

    % Initialize sum
    sum = 0;

    % Trapezoidal rule calculation
    for i = 1:n-1
        x_i = a + i * h;
        sum = sum + f(x_i);
    end

    % Add the boundary values
    integral_value = h * (0.5 * f(a) + sum + 0.5 * f(b));

    % Calculate error if exact value is known
    if ~isnan(exact_value)
        error = abs(exact_value - integral_value);
        fprintf('Integral with h = %.3f is approximately %.5f with an error of %.5e\n', h,
    else
        fprintf('Integral with h = %.3f is approximately %.5f\n', h, integral_value);
    end
end
```

Output

```
To call this function from the MATLAB command window, simply type:
 trapezoidal_integration()
Enter the function to integrate in terms of x: (1+x^2)^(-1)
Enter the lower limit of integration (a): 0
Enter the upper limit of integration (b): 1
Enter the step size: [0.5, 0.25, 0.125]
Enter the exact integral value (if known), otherwise leave empty: 0.78539

Integral with h = 0.500 is approximately 0.77500 with an error of 1.03900e-02
Integral with h = 0.250 is approximately 0.78279 with an error of 2.59588e-03
Integral with h = 0.125 is approximately 0.78475 with an error of 6.42876e-04
```

Chapter 6

Laplace and Fourier Transformation

Practical No. 1

Aim

Compute the Laplace transform of basic function with the help of MATLAB.

Problem

Compute the Laplace transform of function

$$f(t) = t^2 e^{3t}. \tag{6.1}$$

Theory

The Laplace transform of the function $f(t)$ is given by

$$F(s) = \int_0^\infty e^{-st} f(t)dt. \tag{6.2}$$

Algorithm

Step1: Create a symbolic variable t and s;
Step2: Input the function $f(t)$;
Step3: Use Laplace function in MATLAB to compute the Laplace transform $F(s)$;
Step4: Display the result.

Program

The following MATLAB script implements the algorithm.

```
syms t s
f = t^2 * exp(3*t);
F = laplace(f, t, s);
fprintf('The Laplace transform is given by, F(s)=%s\n', F)
```

Output

The output of the program is as follows:

```
The Laplace transform is given by, F(s)=2/(s - 3)^3
```

Exercise Problem

Find the Laplace transform of $f(t) = t^3 e^{-2t}$.

Practical No. 2

Aim

Verify the linearity property of the Laplace transform with the help of MATLAB.

Problem

Verify the linearity property of the Laplace transform using $f(t) = t$, $g(t) = \exp(-t)$, $a = 2$ and $b = 3$.

Theory

Let $F(s)$ and $G(s)$ be the Laplace transform of $f(t)$ and $g(t)$, respectively. The linearity property is given by $L(af(t) + bg(t)) = aF(s) + bG(s)$.

Algorithm

Step1: Create a function file $prob\_2\_verify\_laplace\_linearity$;
Step2: Prompt the user to enter functions f(t) and g(t) and constants a and b;
Step3: Convert input functions to symbolic expressions;
Step4: Define symbolic variables;
Step5: Compute Laplace transforms of $f(t)$ and $g(t)$;
Step6: Compute the linear combination of individual Laplace transforms;
Step7: Compute the Laplace transform of the combined function (a*f(t) + b*g(t));

Step8: Display the results;
Step9: In the command window enter *prob_2_verify_laplace_linearity*;
Step10: Now enter the functions and constants.

Program

The following MATLAB script implements the algorithm.

```
function prob_2_verify_laplace_linearity()
    f_str = input('Enter the function f(t) in terms of t: ', 's');
    g_str = input('Enter the function g(t) in terms of t: ', 's');
    a = input('Enter the constant a: ');
    b = input('Enter the constant b: ');

    f_sym = str2sym(f_str);
    g_sym = str2sym(g_str);

    syms t s

    F_s = laplace(f_sym, t, s);
    G_s = laplace(g_sym, t, s);

    LHS = a * F_s + b * G_s;

    combined_function = a * f_sym + b * g_sym;
    RHS = laplace(combined_function, t, s);

    disp('Laplace Transform of a*f(t) + b*g(t):')
    disp(RHS)
    disp('a * Laplace Transform of f(t) + b * Laplace Transform of g(t):')
    disp(LHS)

    % Verify if LHS equals RHS
    if simplify(LHS - RHS) == 0
        disp('The linearity property of the Laplace transform is verified.')
    else
        disp('The linearity property of the Laplace transform does not hold.')
    end
end
```

Output

The output of the program is as follows:

```
>> prob_2_verify_laplace_linearity()
>> Enter the function f(t) in terms of t: t
>> Enter the function g(t) in terms of t: exp(-t)
>> Enter the constant a: 2
>> Enter the constant b: 3
>> Laplace Transform of a*f(t) + b*g(t):
3/(s + 1) + 2/s^2

>> a * Laplace Transform of f(t) + b * Laplace Transform of g(t):
3/(s + 1) + 2/s^2

>> The linearity property of the Laplace transform is verified.
```

Exercise Problem

Verify the linearity property of the Laplace transform using $f(t) = cos(t)$ and $g(t) = e^{2t}$ with $a = 4, b = -1$.

Practical No. 3

Aim

Implement the first translation theorem in MATLAB.

Problem

Find the Laplace transform of $f(t) = e^{2t} sin(3t)$ using the first translation theorem.

Theory

The first translation theorem is used when the function is a product of two functions with one of the factors being an exponential function. Then the Laplace transform is given as

$$L(e^{at}f(t)) = F(s-a). \tag{6.3}$$

Algorithm

Step1: Create a symbolic variable t and s;
Step2: Input the function $e^{at}f(t)$;
Step3: Use Laplace function in MATLAB to compute the Laplace transform $F(s-a)$;
Step4: Display the result.

Program

The following MATLAB script implements the algorithm.

```
syms t s
f = exp(2 * t) * sin(3 * t);
F = laplace(f, t, s);
fprintf('The Laplace transform is given by, F(s-a)=%s\n', F)
```

Output

The output of the program is as follows:

```
The Laplace transform is given by, F(s-a)=3/((s - 2)^2 + 9)
```

Exercise Problem

Find the Laplace transform of $f(t) = e^{3t}cos(4t)$ using the first translation theorem.

Practical No. 4

Aim

Compute Laplace transform using partial fractions with the help of MATLAB.

Problem

Compute the Laplace transform of $f(t) = te^{4t}$ and simplify the result using partial fractions.

Theory

The Laplace transform by partial fraction technique uses partial fraction expansion to split a complicated fraction into forms that are in the Laplace transform.

Algorithm

Step1: Create a symbolic variable t and s;
Step2: Input the function $f(t)$;
Step3: Compute the Laplace transform of f(t);
Step4: Use partfrac function for partial fraction decomposition of the result;
Step4: Display the decomposed result.

Program

The following MATLAB script implements the algorithm.

```
syms t s
f = t * exp(4 * t);
F = laplace(f, t, s);
F = partfrac(F); % Using partial fraction decomposition
fprintf('The Laplace transform using partial practions is given by:%s\n', F)
```

Output

The output of the program is as follows:

```
The Laplace transform using partial practions is given by:1/(s - 4)^2
```

Exercise Problem

Compute the Laplace transform of $f(t) = te^{-3t}$ and simplify the result using partial fractions.

Practical No. 5

Aim

Apply the change of scale property with the help of MATLAB.

Problem

Find the Laplace transform of $f(t) = \cos(5t)$ using the change of scale property.

Theory

If the Laplace transform of $f(t)$ is $F(s)$, then the Laplace transform of $f(at)$ is $(1/a)F(s/a)$.

Algorithm

Step1: Creates a symbolic variable t and s;
Step2: Input the variable a;
Step3: Input the function $f(t)$;

Step4: Compute the Laplace transform of $f(t)$;
Step5: Apply the change of scale property by substituting s with $\frac{s}{a}$ and multiplying by $\frac{1}{a}$;
Step5: Display the result to verify the change of scale.

Program

The following MATLAB script implements the algorithm.

```
 syms t s
 a = 5;  % Scaling factor
 f = cos(t);
 F = laplace(f, t, s);
 G = (1/a) * subs(F, s, s/a);
 fprintf('The Laplace transform using the change of scale property is given by:
%s\n', G)
```

Output

The output of the program is as follows:

```
The Laplace transform using the change of scale property is given by:
s/(25*(s^2/25 + 1))
```

Exercise Problem

Find the Laplace transform of $f(t) = \sin(6t)$ using the change of scale property.

Practical No. 6

Aim

Calculate the Laplace transform of a derivative with the help of MATLAB.

Problem

For $f(t) = t^3$, find the Laplace transform of its first derivative.

Theory

The Laplace transform of the first derivative is a fundamental property in transforming and solving differential equations. The Laplace transform of the function $f'(t)$ is given by:

$$F(s) = \int_0^\infty e^{-st} f'(t)dt \tag{6.4}$$

Algorithm

Step1: Create a symbolic variable t and s;
Step2: Input the function $f(t)$;
Step3: Differentiate $f(t)$ with respect to t;
Step4: Apply the Laplace transform to the derivative;
Step5: Display the transformed result.

Program

The following MATLAB script implements the algorithm.

```
syms t s
f = t^3;
F_prime = laplace(diff(f, t), t, s);
fprintf('The Laplace transform of first derivative is given by:%s\n', F_prime)
```

Output

The output of the program is as follows:

```
The Laplace transform of first derivative is given by: 6/s^3
```

Exercise Problem

For $f(t) = t^2$, find the Laplace transform of its first and second derivatives.

Practical No. 7

Aim

Implementation of initial and final value theorem with the help of MATLAB.

Problem

Use the initial and final value theorems to find the initial and final values of $F(s) = \frac{10}{s^2+5s+6}$.

Theory

The initial value theorem and the final value theorem are Laplace transform theorems that are used to determine the behavior of a function as time approaches zero or infinity.

Algorithm

Step1: Create a symbolic variable s;
Step2: Input the function $F(s)$;
Step3: Apply the initial value theorem on $F(s)$;
Step4: Apply the final value theorem on $F(s)$;
Step5: Display the initial and final values.

Program

The following MATLAB script implements the algorithm.

```
syms s
F = 10 / (s^2 + 5 * s + 6);
init_val = limit(s * F, s, inf);
final_val = limit(s * F, s, 0);
```

Output

The output of the program is as follows:

```
>> fprintf('The initial value of F(s) is given by:%s\n', init_val)
The initial value of F(s) is given by:0
>> fprintf('The final value of F(s) is given by:%s\n', final_val)
The final value of F(s) is given by:0
```

Exercise Problem

Use the initial and final value theorems to find the initial and final values of $F(s) = \frac{5s+3}{s^2+4s+5}$.

Practical No. 8

Aim

Laplace transform of an integral with the help of MATLAB.

Problem

Find the Laplace transform of $f(t) = \int_0^t e^{2\tau}\, d\tau$.

Theory

The Laplace transform of the integral of a function $f(t)$ is defined by the integral $F(s) = \int_0^\infty e^{-st} f(t) dt,\ s > 0$.

Algorithm

Step1: Create a symbolic variable t and s.
Step2: Define the integrand $g(\tau) = e^{2\tau}$;
Step3: Compute the integral of $g(\tau)$ from 0 to t, which results in $f(t)$;
Step4: Apply the Laplace transform to $f(t)$ with respect to$t$, giving $F(s)$;
Step5: Display the final Laplace transform result.

Program

The following MATLAB script implements the algorithm.

```
 syms t tau s
 % Define the integrand function g() = e^(2)
g = exp(2 * tau);
 % Compute the integral from 0 to t to find f(t)
f = int(g, tau, 0, t);
 % Find the Laplace transform of f(t)
F = laplace(f, t, s);
 % Display the result
disp('Laplace Transform of f(t):')
disp(F)
```

Output

The output of the program is as follows:

```
Laplace Transform of f(t):
1/(2*(s - 2)) - 1/(2*s)
```

Exercise Problem

Find the Laplace transform of $f(t) = \int_0^t \cos(3\tau)\, d\tau$.

Practical No. 9

Aim

Laplace transform of periodic functions with the help of MATLAB.

Problem

Determine the Laplace transform of a periodic function $f(t) = sin(t)$ with period $T = 2\pi$.

Theory

The Laplace transform of a periodic function provides an efficient way to analyze periodic signals in the frequency domain. For a function $f(t)$ with period T, the Laplace transform can be computed using a specific formula that takes the periodicity into account.

$$L(f(t)) = \frac{\int_0^T f(t)e^{-st}dt}{1 - e^{-sT}}. \tag{6.5}$$

Algorithm

Step1: Create a symbolic variable t and s;
Step2: Define the function $f(t) = \sin(t)$;
Step3: Define the period $T = 2 * pi$;
Step4: Compute the integral of $f(t)$ over one period $[0, T]$;
Step5: Apply the Laplace transform formula for periodic functions.
Step6: Display the result.

Program

The following MATLAB script implements the algorithm.

```
syms t s
% Define the function f(t) = sin(t)
f = sin(t);
% Define the period T = 2 * pi
T = 2 * pi;
```

```
 % Compute the integral of f(t) * exp(-s * t) over one period [0, T]
 integral_result = int(f * exp(-s * t), t, 0, T);
 % Apply the Laplace transform formula for periodic functions
 F = integral_result / (1 - exp(-s * T));
 % Display the result
 disp('Laplace Transform of the periodic function f(t):')
disp(F)
```

Output

The output of the program is as follows:

```
Laplace Transform of the periodic function f(t):
1/(s^2 + 1)
```

Exercise Problem

Determine the Laplace transform of a periodic function $f(t) = cos(t)$ with period $T = \pi$.

Practical No. 10

Aim

Inverse Laplace transform with convolution theorem with the help of MATLAB.

Problem

Find the inverse Laplace transform of $F(s) = \frac{1}{s(s+2)}$ using convolution theorem.

Theory

The inverse Laplace transform of $F(s)$ is given by $f(t) = L^{-1}(F(s))$.

Algorithm

Step1: Create a symbolic variable t and s;
Step2: Define the function $F(s)$;
Step3: Use MATLABs ilaplace function to compute the inverse Laplace transform;
Step4: Display the result.

Program

The following MATLAB script implements the algorithm.

```
syms t s
F = 1 / (s * (s + 2));
f = ilaplace(F, s, t);
fprintf('The inverse Laplace transform is given by:%s\n', f)
```

Output

The output of the program is as follows:

```
The inverse Laplace transform is given by:1/2 - exp(-2*t)/2
```

Exercise Problem

Find the inverse Laplace transform of $F(s) = \frac{1}{s^2(s+1)}$ using convolution theorem.

Practical No. 11

Aim

Compute inverse Laplace transform with Heaviside expansion with the help of MATLAB.

Problem

Compute the inverse Laplace transform of $F(s) = \frac{s+1}{s^2+4s+5}$ with Heaviside expansion.

Theory

The inverse Laplace transform is used to convert functions from the s-domain (frequency domain) back to the t-domain (time domain). When computing the inverse Laplace transform of a rational function we often use the Heaviside expansion method (also known as partial fraction decomposition). This approach decomposes the function into simpler fractions, which can be individually transformed back to the time domain using standard inverse Laplace transforms.

Algorithm

Step1: Create a symbolic variable t and s;
Step2: Define the function $F(s)$;
Step3: Perform partial fraction decomposition;
Step4: Compute the inverse Laplace transform;
Step5: Display the result.

Program

The following MATLAB script implements the algorithm.

```
 syms s t
 F = (s + 1) / (s^2 + 4*s + 5);
 F_decomposed = partfrac(F);
 f_t = ilaplace(F_decomposed, s, t);
 disp('Inverse Laplace Transform of F(s):')
disp(f_t)
```

Output

The output of the program is as follows:

```
Inverse Laplace Transform of F(s):
exp(-2*t)*(cos(t) - sin(t))
```

Exercise Problem

Compute the inverse Laplace transform of $F(s) = \frac{s+2}{s^2+2s+5}$ with Heaviside expansion.

Practical No. 12

Aim

Solve a differential equation using Laplace transform in MATLAB.

Problem

Solve the following differential equation using Laplace transform:

$$y'' + 3y' + 2y = 0 \tag{6.6}$$

with initial conditions $y(0) = 1, y'(0) = 0$.

Theory

Using the Laplace transform to solve differential equations is an effective method to convert differential equations in the time domain into algebraic equations in the frequency domain (s-domain). Once transformed, the differential equation becomes an algebraic equation in terms of s and can be solved for $Y(s)$, the Laplace transform of $y(t)$. The solution $y(t)$ is then found by applying the inverse Laplace transform.

Algorithm

Step1: Create a symbolic variable s, $y(t)$ and Y;
Step2: Define the differential equation;
Step3: Apply the Laplace transform to both sides of the equation;
Step4: Solve for $Y(s)$;
Step5: Find the inverse Laplace transform to get $y(t)$;
Step6: Display the solution.

Program

The following MATLAB script implements the algorithm.

```
syms y(t) Y s
Dy = diff(y, t);
D2y = diff(y, t, 2);
eqn = D2y + 3*Dy + 2*y == 0;
LEqn = laplace(eqn, t, s);
LEqn = subs(LEqn, [laplace(y(t), t, s), y(0), subs(Dy, t, 0)], [Y, 1, 0]);
Y_s = solve(LEqn, Y);
y_t = ilaplace(Y_s, s, t);
disp('Solution y(t) of the differential equation:')
disp(y_t)
```

Output

The output of the program is as follows:

```
Solution y(t) of the differential equation:
2*exp(-t) - exp(-2*t)
```

Exercise Problem

Solve the following differential equation using Laplace transform:

$$y^{''} + 2y = e^2 t \tag{6.7}$$

with initial conditions $y(0) = 0, y^{'}(0) = 1$.

Practical No. 13

Aim

Solve simultaneous differential equations with Laplace transform in MATLAB.

Problem

Solve the following simultaneous differential equations using Laplace transform:

$$y^{'} + z = 0, \quad z^{'} + 2z + y = 0 \tag{6.8}$$

with initial conditions $y(0) = 0, z(0) = 1$.

Theory

The Laplace transform is particularly useful to solve systems of linear differential equations by transforming each equation from the time domain to the s-domain, where differentiation becomes algebraic. For a system of differential equations, each equation can be transformed individually, then solved simultaneously for the Laplace transforms of the unknown functions. Once the transformed equations are solved, we apply the inverse Laplace transform to retrieve the original functions.

Algorithm

Step1: Create a symbolic variable $y(t)$, $z(t)$, Y and Z;
Step2: Define the differential equations;
Step3: Apply Laplace transform to both equations;
Step4: Solve for $Y(s)$ and $Z(s)$;
Step5: Find the inverse Laplace transform to get $y(t)$ and $z(t)$;
Step6: Display the result.

Program

The following MATLAB script implements the algorithm.

```
syms y(t) z(t) Y Z s
Dy = diff(y, t);
```

```
 Dz = diff(z, t);
 eqn1 = Dy + z == 0;
 eqn2 = Dz + 2*z + y == 0;
 LEqn1 = laplace(eqn1, t, s);
 LEqn2 = laplace(eqn2, t, s);
 LEqn1 = subs(LEqn1, [laplace(y(t), t, s), laplace(z(t), t, s), y(0), z(0)],
[Y, Z, 0, 1]);
 LEqn2 = subs(LEqn2, [laplace(y(t), t, s), laplace(z(t), t, s), y(0), z(0)],
[Y, Z, 0, 1]);
 solutions = solve([LEqn1, LEqn2], [Y, Z]);
 Y_s = solutions.Y;
 Z_s = solutions.Z;
 y_t = ilaplace(Y_s, s, t);
 z_t = ilaplace(Z_s, s, t);>>
 disp('Solution y(t):')
 disp(y_t)
 disp('Solution z(t):')
 disp(z_t)
```

Output

The output of the program is as follows:

```
Solution y(t):
-(2^(1/2)*exp(-t)*sinh(2^(1/2)*t))/2
Solution z(t):
exp(-t)*(cosh(2^(1/2)*t) - (2^(1/2)*sinh(2^(1/2)*t))/2)
```

Exercise Problem

Solve the following simultaneous differential equations using Laplace transform:

$$y^{'} = y + z, \quad z^{'} = 2y - z \tag{6.9}$$

with initial conditions $y(0) = 1, z(0) = -1$.

Practical No. 14

Aim

Solve partial differential equations with Laplace transform in MATLAB.

Problem

Solve the partial differential equations using Laplace transform:

$$u_t = ku_{xx}, \quad u(x, 0) = \sin x. \tag{6.10}$$

Theory

The Laplace transform in the time domain is often applied to PDEs with respect to t to simplify time-dependent terms. This process converts derivatives in t into algebraic terms in s, allowing the PDE to be reduced to an ODE in the spatial variable x.

Algorithm

Step1: Set up the symbolic variables x, t, s, k, and $U(x)$;
Step2: Define the initial condition;
Step3: Laplace transform of u_t with respect to t;
Step4: Laplace transform of u_{xx} with respect to x;
Step5: Solve the resulting ODE in x for $U(x, s)$;
Step6: Simplify $U(x, s)$ if possible;
Step7: Take the inverse Laplace transform to get $u(x, t)$;
Step8: Display the final solution.

Program

The following MATLAB script implements the algorithm.

```
syms x t s k U(x)
u0 = sin(x);
PDE = s*U - u0 == k*diff(U, x, 2);
U_sol = dsolve(PDE);
U_sol = simplify(U_sol);
u_sol = ilaplace(U_sol, s, t);
u_sol = simplify(u_sol);
disp('Simplified Solution u(x, t):')
disp(u_sol)
```

Output

The output of the program is as follows:

```
Simplified Solution u(x, t):
C3*ilaplace(exp((x*(k*s)^(1/2))/k), s, t)
+ C4*ilaplace(exp(-(x*(k*s)^(1/2))/k), s, t)
+ (sin(x)*ilaplace(exp((x*((k*s)^(1/2)
- k^(1/2)*s^(1/2)))/k)/(k + s), s, t))/2
+ (sin(x)*ilaplace(exp(-(x*((k*s)^(1/2)
- k^(1/2)*s^(1/2)))/k)/(k + s), s, t))/2
+ (k^(1/2)*cos(x)*ilaplace(exp((x*((k*s)^(1/2)
- k^(1/2)*s^(1/2)))/k)/(s^(1/2)*(k + s)), s, t))/2
- (k^(1/2)*cos(x)*ilaplace(exp(-(x*((k*s)^(1/2)
- k^(1/2)*s^(1/2)))/k)/(s^(1/2)*(k + s)), s, t))/2
```

Exercise Problem

Solve the partial differential equations using Laplace transform:

$$u_t = 4u_{xx}, \quad u(x,0) = x\ sinx \tag{6.11}$$

Practical No. 15

Aim

Visualize the differential equation of the circuit with the help of MATLAB.

Problem

Visualize the following circuit's differential equation using Laplace transform:

$$V(t) = RI(t) + L\frac{dI(t)}{dt} + \frac{1}{C}\int I(t)dt. \tag{6.12}$$

Theory

The Laplace transform is extremely useful in circuit analysis because it allows you to convert the differential equations governing the circuit (Kirchhoff's laws) into algebraic equations. Once the equations are in the $s-$domain, theyre easier to solve, and the current $I(s)$ can be found as an algebraic function. Applying the inverse Laplace transform to $I(s)$ then gives the current in the time domain $I(t)$.

Algorithm

Step1: Set up the symbolic variables t and s;
Step2: Define numerical values for R, L, and C;
Step3: Define the input voltage as a step function in the Laplace domain $V(s)$;
Step4: Define the equation of the RLC circuit in the s-domain and solve for I(s) by rearranging the equation;

Step5: Take the inverse Laplace transform to find $I(t)$;
Step6: Display the time-domain current $I(t)$;
Step7: Plot the current response over time.

Program

The following MATLAB script implements the algorithm.

```
>> syms t s
>> R = 1;     % Resistance in ohms
>> L = 1;     % Inductance in henries
>> C = 0.5;   % Capacitance in farads
>> V_s = 1 / s;
>> I_s = V_s / (R + L * s + 1 / (C * s));
>> I_t = ilaplace(I_s, s, t);
>> disp('Current I(t) in the time domain:')
>> disp(I_t)
>> fplot(I_t, [0, 10])
>> title('Current Response I(t) in an RLC Circuit with Step Input Voltage')
>> xlabel('Time (t)')
>> ylabel('Current I(t)')
>> grid on
```

Output

The output of the program is as follows:

```
Current I(t) in the time domain:
(2*7^(1/2)*exp(-t/2)*sin((7^(1/2)*t)/2))/7
```

The output is also given by the figure 6.1.

Exercise Problem

Solve the partial differential equations using Laplace transform:

$$u_t = 4u_{xx}, \quad u(x,0) = x\sin x. \tag{6.13}$$

Practical No. 16

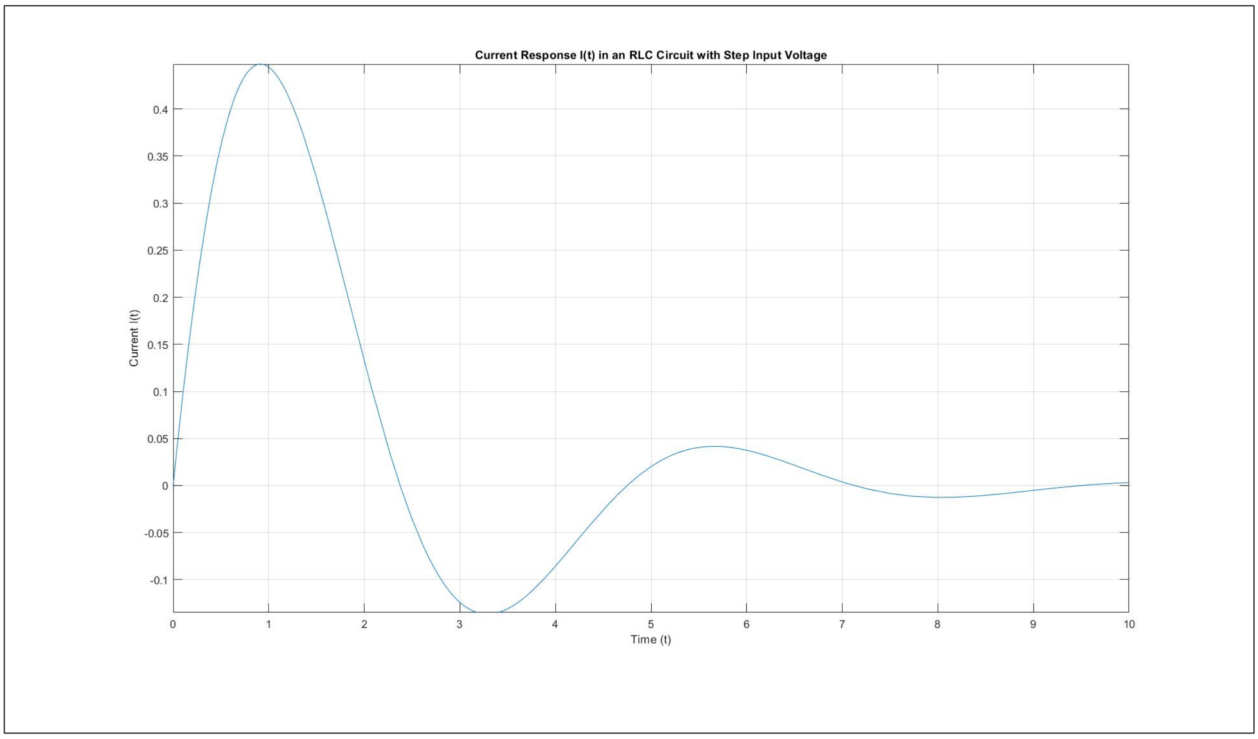


Figure 6.1: Current Response I(t) in an RLC Circuit with Step Input Voltage

Aim

Compute the Fourier transform of basic function with the help of MATLAB.

Problem

Find the Fourier transform of $f(t) = e^{-2|t|}$.

Theory

The Fourier transform of the function $f(t)$ is given by:

$$F(\omega) = \int_{-\infty}^{\infty} e^{-i\omega t} f(t) dt \tag{6.14}$$

Algorithm

Step1: Create a symbolic variable t and w;
Step2: Input the function $f(t)$;
Step3: Use MATLAB's fourier function to compute the Fourier transform $F(\omega)$;
Step4: Display the result.

Program

The following MATLAB script implements the algorithm.

```
>> syms t w
>> f = exp(-2 * abs(t));
>> F = fourier(f, t, w);
>> fprintf('The Fourier transform of f(t) is given by:%s\n', F)
```

Output

The output of the program is as follows:

```
The Fourier transform of f(t) is given by:4/(w^2 + 4)
```

Exercise Problem

Find the Fourier transform of $f(t) = e^{-3|t|}$.

Practical No. 17

Aim

Verify the linearity property of the Fourier transform with the help of MATLAB.

Problem

Verify the linearity property for $f(t) = \sin(t)$ and $g(t) = \cos(t)$, with constants a and b.

Theory

Let $F(\omega)$ and $G(\omega)$ be the Fourier transform of $f(t)$ and $g(t)$, respectively. The linearity property is given by $F(af(t) + bg(t)) = aF(\omega) + bG(\omega)$.

Algorithm

Step1: Set up the symbolic variables t and w;
Step2: Input the function $f(t)$;
Step3: Compute Fourier transform of $f(t)$;
Step4: Input the function $g(t)$;

Step5: Compute Fourier transform of $g(t)$;
Step6: Compute $aF(w) + bG(w)$;
Step7: Compute Fourier transform of $af(t) + bg(t)$;
Step8: Display all the Fourier transforms;
Step9: Verify the linearity property.

Program

The following MATLAB script implements the algorithm.

```
>> syms t w a b
>> f = sin(t);
>> F = fourier(f, t, w);
>> g = cos(t);
>> G = fourier(g, t, w);
>> S = a*F + b*G;
>> T = fourier(a * f + b * g, t, w);
>> fprintf('The Fourier transform of f(t) is given by F(w)=%s\n', F)
>> fprintf('The Fourier transform of g(t) is given by G(w)=%s\n', G)
>> fprintf('aF(s)+bG(s)=%s\n', S)
>> fprintf('The Fourier transform of af(t)+bg(t) is given by:%s\n', T)
>> if T == S
>> disp('Linearity property is satisfied')
>> else
>> disp('Linearity property is not satisfied')
>> end
```

Output

The output of the program is as follows:

```
The Fourier transform of f(t) is given by,
F(w)=-pi*(dirac(w - 1) - dirac(w + 1))*1i
The Fourier transform of g(t) is given by,
G(w)=pi*(dirac(w - 1) + dirac(w + 1))
aF(s)+bG(s)=b*pi*(dirac(w - 1) + dirac(w + 1))
- a*pi*(dirac(w - 1) - dirac(w + 1))*1i
The Fourier transform of af(t)+bg(t) is given by:
b*pi*(dirac(w - 1) + dirac(w + 1)) - a*pi*(dirac(w - 1) - dirac(w + 1))*1i
Linearity property is satisfied
```

Exercise Problem

Verify the linearity property for $f(t) = t$ and $g(t) = e^{-t}$, with constant $a = 4$ and $b = -1$.

Practical No. 18

Aim

Compute the Fourier cosine transform problem with the help of MATLAB.

Problem

Find the Fourier cosine transform of $f(t) = e^{-t}$.

Theory

The Fourier cosine transform of the function $f(t)$ is given by:

$$F(\omega) = \int_0^\infty \cos(\omega t) f(t) dt. \tag{6.15}$$

Algorithm

Step1: Create a symbolic variable t and w;
Step2: Input the function $f(t)$;
Step3: Compute the Fourier cosine transform of $f(t)$ manually;
Step4: Display the Fourier cosine transform;
Step5: Convert the Fourier cosine transform to a MATLAB function for plotting;
Step6: Plot the original function and the Fourier cosine transformed function.

Program

The following MATLAB script implements the algorithm.

```
>> syms t w
>> f = exp(-t);
>> F_cosine = int(f * cos(w * t), t, 0, inf);
>> disp('Fourier Cosine Transform of f(t):')
>> disp(F_cosine)
>> F_cosine_func = matlabFunction(F_cosine, 'Vars', w);
%  Plot the original function
>> figure;
>> fplot(@(t) exp(-t), [0, 5])
>> title('Original Function f(t) = e^{-t}')
>> xlabel('Time (t)')
>> ylabel('f(t)')
>> grid on
```

```
% Plot the Fourier cosine transform F_c(w)
>> figure;
>> fplot(F_cosine_func, [0, 10])
>> title('Fourier Cosine Transform of f(t) = e^{-t}')
>> xlabel('Frequency (\omega)')
>> ylabel('F_c(\omega)')
>> grid on
```

Output

The output of the program is as follows:

```
Fourier Cosine Transform of f(t):
real((1 + w*1i)/(w^2 + 1))
```

The output is also given by the figures 6.2 and 6.3.

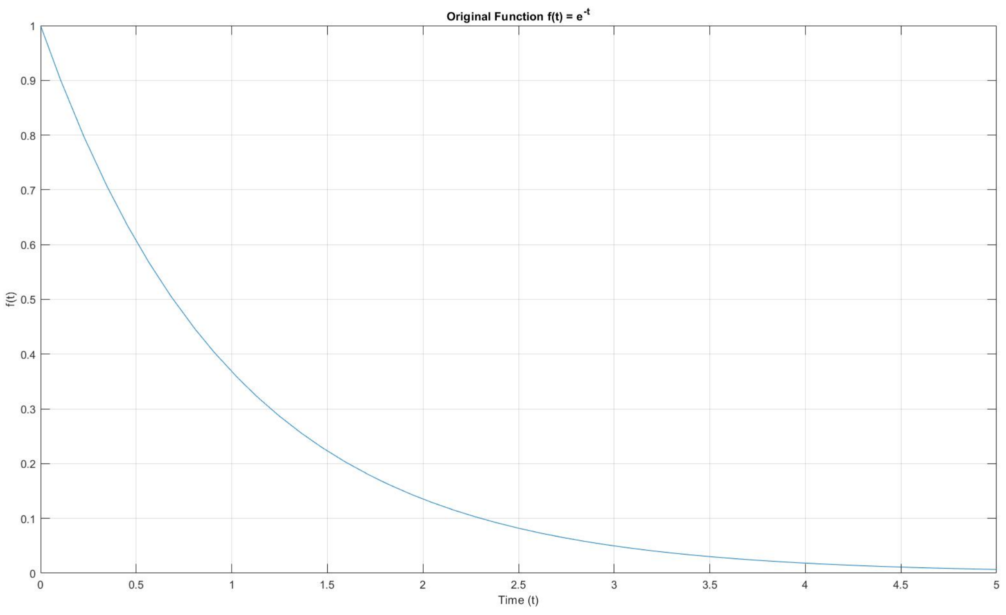


Figure 6.2: Original Function

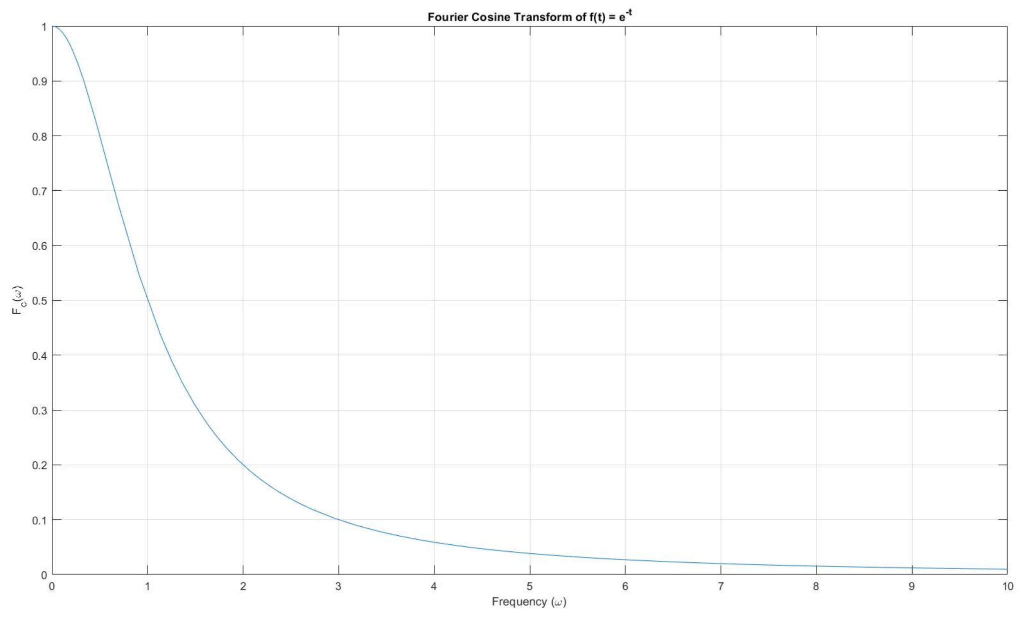


Figure 6.3: Fourier Cosine Transform

Exercise Problem

Find the Fourier cosine transform of $f(t) = e^{-2t}$.

Practical No. 19

Aim

Compute the Fourier sine transform problem with the help of function file in the MATLAB.

Problem

Find the Fourier sine transform of $f(t) = te^{-t}$.

Theory

The Fourier cosine transform of the function $f(t)$ is given by:

$$F(\omega) = \int_0^\infty sin(\omega t)f(t)dt. \tag{6.16}$$

Algorithm

Step1: Create a function file $fourier\_sine\_transform$;
Step2: Input the function $f(t)$ and convert input string to a symbolic expression;
Step3: Create a symbolic variable t and w;
Step4: Compute the Fourier sine transform of $f(t)$ manually;
Step5: Display the Fourier sine transform;
Step6: Convert the Fourier sine transform to a MATLAB function for plotting;
Step7: Plot the original function and Fourier sine transformed function;
Step8: In the command window enter $fourier\_sine\_transform$;
Step9: Now enter the function.

Program

The following MATLAB script implements the algorithm.

```
function fourier_sine_transform()
    % Prompt the user to enter a function f(t)
    f_str = input('Enter the function f(t) in terms of t: ', 's');
    f_sym = str2sym(f_str);  % Convert the input string to a symbolic expression

    % Define symbolic variables
    syms t w

    % Define the function f(t)
    f = subs(f_sym, 't', t);  % Substitute 't' in case the user input uses 't'
    % Compute the Fourier sine transform of f(t) manually
    F_sine = int(f * sin(w * t), t, 0, inf);

    % Display the Fourier sine transform
    disp('Fourier Sine Transform of f(t):')
    disp(F_sine)

    % Convert the Fourier sine transform to a MATLAB function for plotting
    F_sine_func = matlabFunction(F_sine, 'Vars', w);

    % Plot the original function f(t)
    figure;
    fplot(matlabFunction(f), [0, 5])
    title('Original Function')
    xlabel('Time (t)')
    ylabel('f(t)')
    grid on

    % Plot the Fourier sine transform F_s(w)
    figure;
    fplot(F_sine_func, [0, 10])
    title('Fourier Sine Transform')
    xlabel('Frequency (\omega)')
    ylabel('F_s(\omega)')
    grid on
end
```

Output

The output of the program is as follows:

```
>> fourier_sine_transform
Enter the function f(t) in terms of t: t*exp(-t)
Fourier Sine Transform of f(t):
imag(1/(- 1 + w*1i)^2)
```

The output is also given by the figures 6.4 and 6.5.

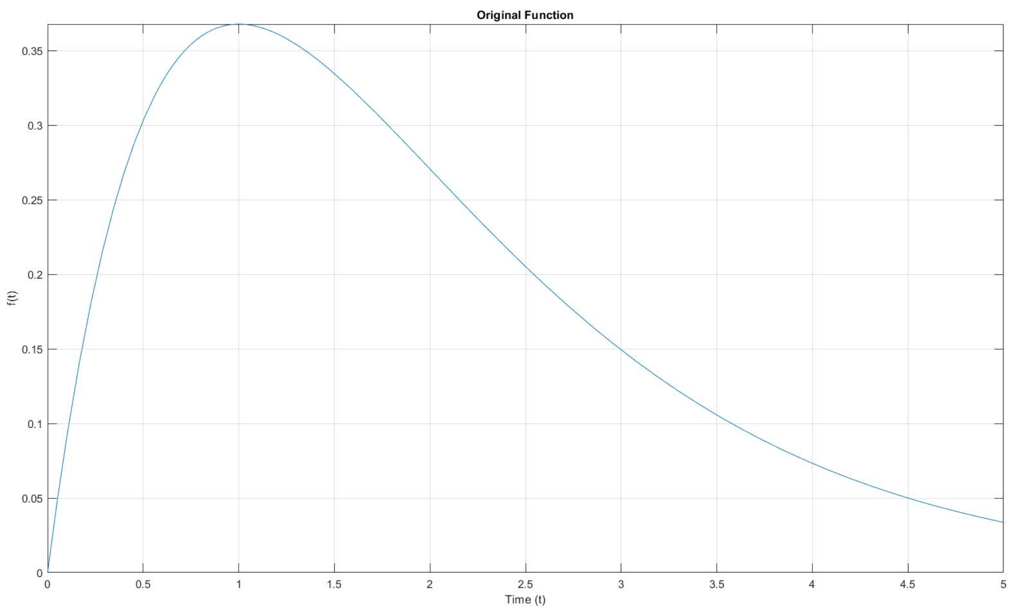


Figure 6.4: Original Function

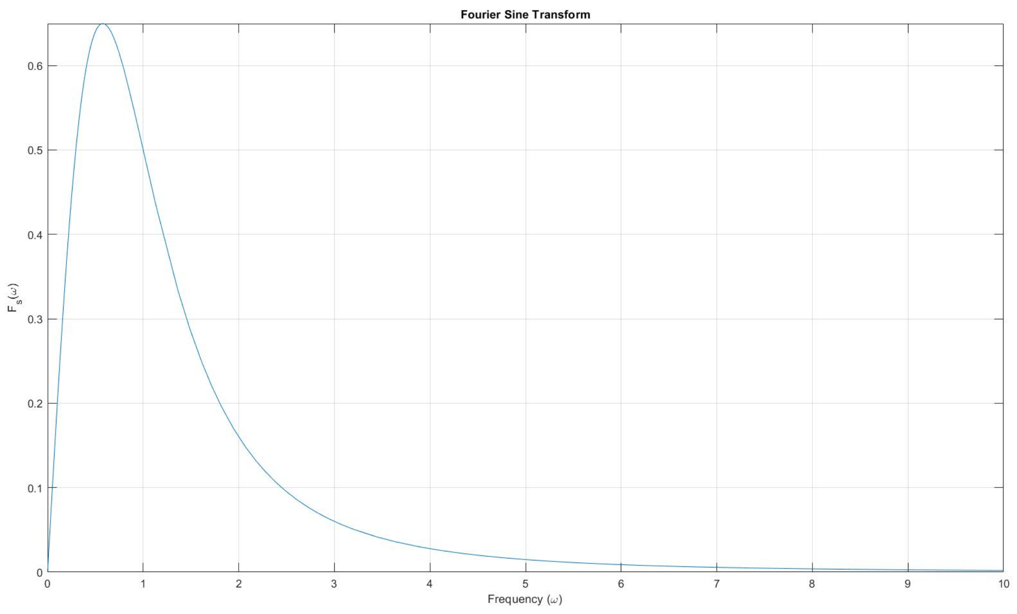


Figure 6.5: Fourier Sine Transform

Exercise Problem

Find the Fourier sine transform of $f(t) = te^{-2t}$.

Practical No. 20

Aim

Compute the Inverse Fourier transform of the function with the help of function file in the MATLAB.

Problem

Find the inverse Fourier transform of $F(w) = \frac{1}{w^2+4}$.

Theory

The inverse Fourier transform of the function $F(\omega)$ is given by:

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)e^{j\omega}d\omega \tag{6.17}$$

Algorithm

Step1: Create a function file *inverse_fourier_transform*;
Step2: Input the function $F(w)$ and convert the input string to a symbolic expression;
Step3: Creates a symbolic variable t and w;
Step4: Compute the inverse Fourier transform of $F(w)$ using function ifourier;
Step5: Display the inverse Fourier transform;
Step6: Convert the inverse Fourier transform to a MATLAB function for plotting;
Step7: Plot the original function and inverse Fourier transformed function;
Step8: In the command window, enter *inverse_fourier_transform*;
Step9: Now enter the function.

Program

The following MATLAB script implements the algorithm.

```
function inverse_fourier_transform()
    % Prompt the user to enter a function F(w)
    F_str = input('Enter the function F(w) in terms of w: ', 's');
    F_sym = str2sym(F_str);  % Convert the input string to a symbolic expression

    % Define symbolic variables
    syms t w

    % Define the function F(w)
    F = subs(F_sym, ‘w’, w);  % Substitute ‘w’ in case the user input uses ‘w’

    % Compute the inverse Fourier transform of F(w)
    f_t = ifourier(F, w, t);

    % Display the inverse Fourier transform
    disp(‘Inverse Fourier Transform of F(w):’)
    disp(f_t)

    % Convert the original function F(w) and the inverse Fourier transform f(t) t
    MATLAB functions
    F_func = matlabFunction(F, ’Vars’, w);
    f_t_func = matlabFunction(f_t, ’Vars’, t);

    % Plot the original function F(w)
    figure;
    fplot(F_func, [-10, 10])
    title(‘Original Function F(\omega)’)
    xlabel(‘Frequency (\omega)’)
    ylabel(‘F(\omega)’)
    grid on

    % Plot the inverse Fourier transform f(t)
    figure;
    fplot(f_t_func, [-10, 10])
    title(‘Inverse Fourier Transform f(t)’)
    xlabel(‘Time (t)’)
    ylabel(‘f(t)’)
    grid on
end
```

Output

The output of the program is as follows:

```
>> inverse_fourier_transform()
Enter the function F(w) in terms of w: 1 / (w^2 + 4)
Inverse Fourier Transform of F(w):
exp(-2*abs(t))/4
```

The output is also given by the figures 6.6 and 6.7.

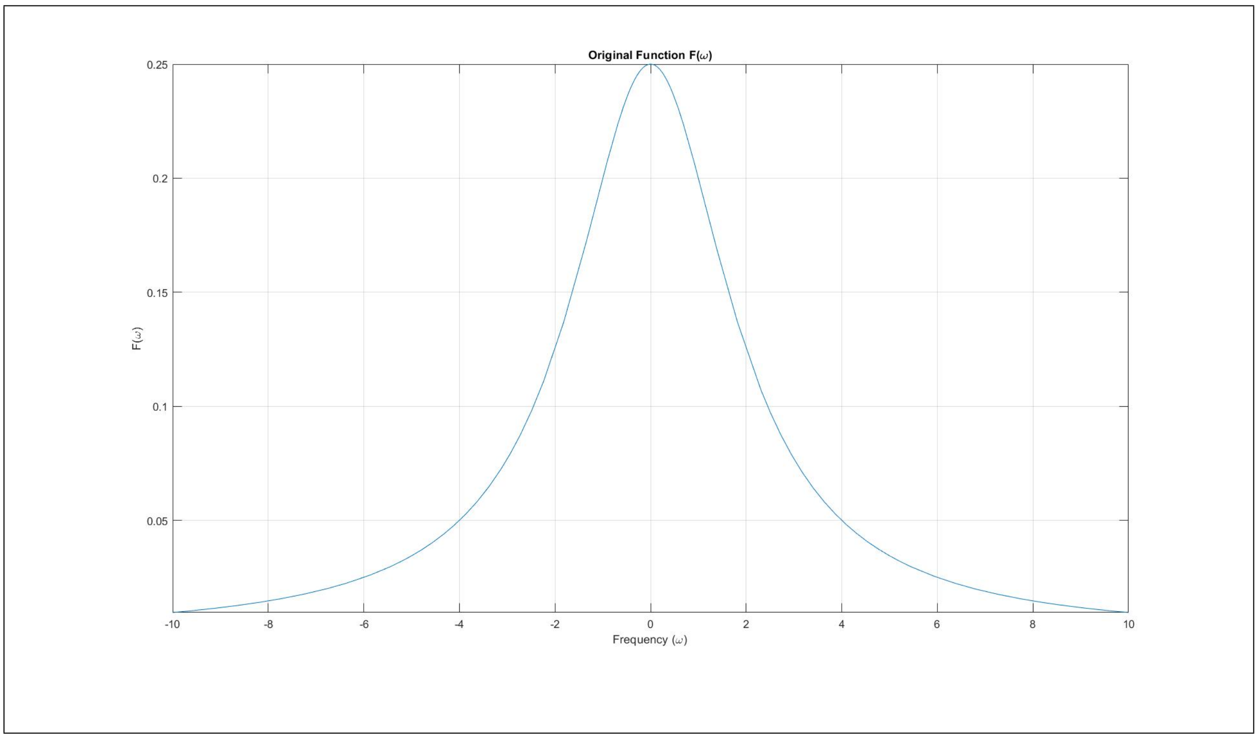


Figure 6.6: Original Function $F(\omega)$

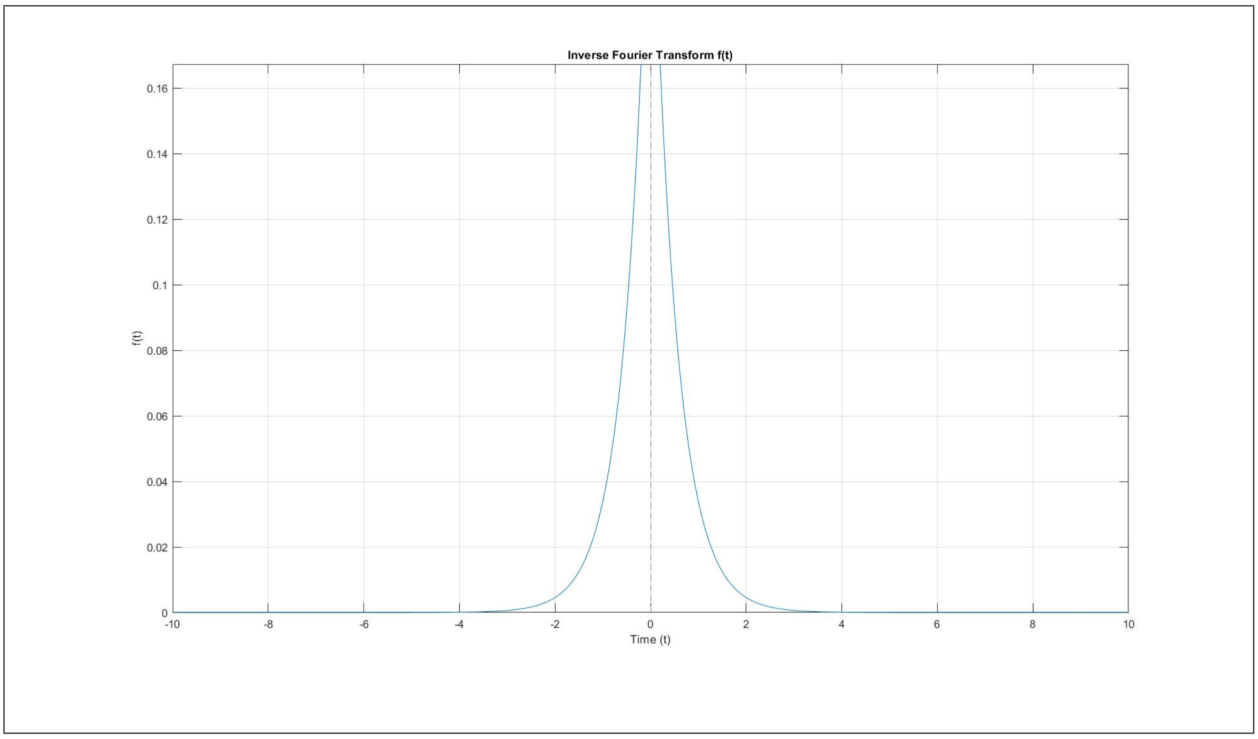


Figure 6.7: Inverse Fourier Transform f(t)

Exercise Problem

Find the inverse Fourier transform of $F(w) = \frac{w}{w^2+9}$.

Chapter 7

Advance Discrete Mathematics

Practical No. 1

Aim

To write a MATLAB program that verifies whether a non-empty set with a given operation is reflexive. Here user will give input of 'set' and 'operation' on that.

Problem

Define $(S, |) = (\{1, 2, 5, 7, 10, 14, 35, 70\}, |)$, here $|$ is the operation of divisibility. Using MATLAB, show that S is reflexive under $|$.

Theory

$a \mid b$ if there exist a number $c \in \mathbf{N}$ such that $b = ac$.
A relation R is reflexive on S if $aRa, \forall a \in S$.

Algorithm

Step1: Using the 'input' command of MATLAB, define a set and relation given by the user;
Step2: Define a MATLAB function that takes set and relation as input;
Step3: Check whether each element is related to itself using the concept of looping;
Step4: If it fails to satisfy for any $x \in S$ show the message 'Failed reflexivity for a = x;
Step5: If given set satisfies Step3, show the message 'The given relation is reflexive.'

Program

```
S=input('Define a set S= ');
```

```
relation=input('define a relation r= ')
A=Q13_Reflexive(S,relation);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function isR = Q13_Reflexive(S,r)
% S: A set of elements (e.g., a vector, matrix, or list of numbers)
% r: A function handle representing the binary relation (e.g., @(a,b) a <= b)

isR = true; % Assume it's a Reflexive until proven otherwise

% 1. Check Reflexivity: a <= a for all a in S
  for i = 1:length(S)
  a = S(i);
  if ~relation(a, a)  % Check if a <= a
  disp(['Failed reflexivity for a = ', num2str(a)]);
  isR = false;
  return
  end
  end
  if isR
  disp('The given relation is reflexive');
  end
  end
```

Input

```
S=[1 2 5 7 10 14 35 70];
r=@(a,b) (a|b);
```

Output

```
The given relation is reflexive.
```

Exercise Problem

Q1. Write a MATLAB code that verifies whether a given non-empty set with a given operation is anti symmetric.
Q2. Verify whether $S = \{1, 2, 4, 6, 8\}$ under operation $r = \{(a, b)|a = b + 1\}$ is reflexive.
Q3. Experiment with this code by a set and relation of your choice.

Practical No. 2

Aim

To write a MATLAB program that verifies whether a non-empty set with a given operation is transitive. Here user will give input of set and operation on that.

Problem

Define a set $S = \{1, 3, 5, 6, 8, 9, 10\}$, and a relation $r = \{(a, b)|a = b + 1\}$. Check by a user-defined MATLAB function, whether it is transitive.

Theory

A relation R is transitive on S, if for $a, b, c \in S$, aRb, bRc then aRc.

Algorithm:

Step1: Define a nonempty set;
Step2: Define a binary relation of your choice. *e.g.*, $@(a, b)a == b^2$;
Step3: Call the user defined MATLAB function ($Q14\_Transitive(S, r)$).

Program:

```
S=input('Define a set S= ');
r=input('Define a relation r= ')
A=Q14_Transitive(S,r);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function isT = Q14_Transitive(S,r)
% S: A set of elements (e.g., a vector, matrix, or list of numbers)
% r: A function handle representing the binary relation (e.g., @(a,b) a <= b)

isT = true; % Assume it's a Transitive until proven otherwise

  % 3. Check Transitivity: If a <= b and b <= c, then a <= c
  for i = 1:length(S)
  for j = 1:length(S)
  for k = 1:length(S)
  a = S(i);
  b = S(j);
  c = S(k);
  if r(a, b) && r(b, c) && ~r(a, c)
  disp(['Failed transitivity for a = ', num2str(a),',
   b = ', num2str(b), ', c = ', num2str(c).]);
  isT = false;
  return;
  end
  end
  end
  end

  if isT
  disp('The given relation is transitive);
  end
  end
```

Input

```
S=[1 3 5 6 8 9 10];
r=@(a,b)(a=b+1);
```

Output

Failed transitivity for a = 10, b = 9, c = 8.

Exercise Problem

Q1. Define a set of numbers S and a binary relation r of your choice. Check the transitivity by giving these input to the defined function.
Q2. Write a similar MATLAB program to check whether a given relation the symmetric.

Practical No. 3

Aim

To write a MATLAB program that verifies whether a non-empty set with two binary operations 'meet', and 'join' is closed. Here non empty set and binary operations meet and join will be given by user input.

Problem

Define a set $S = \{1, 3, 5, 7, 6, 8, 11\}$, operation meet is least common multiple (lcm), and join is the operation of multiplication. Verify by a user defined MATLAB function whether this set is closed under these operations.

Theory

$lcm(a, b)$ = Smallest positive integer that is divisible by both a and b.

Algorithm

Step1: Define a set S;
Step2: Define an operation 'meet';
Step3: Define an operation 'join';
Step4: Recall the used defined function $Q15\_CloseT(S, meet, join)$.

Program

```
S=input('Define a set S= ');
meetOp=input('Define an operation meet = ');
joinOp=input('Define an operation join = ');
A=Q15_CloseT(S, meetOp, joinOp);

function isClose = Q15_CloseT(S, meetOp, joinOp)
% S: Set of elements
% meetOp: Function handle for the meet operation
% joinOp: Function handle for the join operation

% Initialize to true
isClose = true;
% Ckeck the operation meet and  join are closed.
for i = 1:length(S)
for j = 1:length(S)
% meet
if ~ismember(meetOp(S(i), S(j)),S)
disp(['Operation meet is not closed for a=',
num2str(S(i)), ' and b= ', num2str(S(j)),.]);
isClose = false;
return;
end
% join
if ~ismember(joinOp(S(i),S(j)),S)
disp(['Operation join is not closed for a=',
num2str(S(i)), ' and b= ' ,num2str(S(j)),.]);
isClose = false;
```

```
return;
end
end
end
if isClose
disp('The given set is closed under meet and join');
end
end
```

Input

```
S=[1 3 5 6 7 8 11];
meet=@(a,b) lcm(a,b);
join=@(a,b) gcd(a,b);
```

Output

Operation meet is not closed for a=3 and b= 5.

Exercise Problem

Q1. Write a similar MATLAB program that takes set of characters as input and verify the closeness of two given operations on that set.
Q2. Define a finite set of numbers and two operations of your choice. Check the closeness of defined operations by the given user defined MATLAB function.

Practical No. 4

Aim

To write a MATLAB program that verifies the law of absorption on a given non-empty set with two binary operations 'meet' and 'join'.

Problem

Define a set $S = \{1, 4, 5, 23, 56, 99\}$, meet$(a, b) = a + b^2 + 3b + 1$, join$(a, b) = a \cdot b$. Check the law of absorption on S with these operations.

Theory

Laws of absorption: $a \wedge (a \vee b) = a$; $a \vee (a \wedge b) = a$. Here $\vee$, and $\wedge$ denotes operation meet and join respectively.

Algorithm

Step1: Define a set S;
Step2: Define an operation 'meetOp';
Step3: Define an operation 'joinOp';
Step4: recall the user defined function $Q16\_AbsorptionT(S, meetOp, joinOp)$.

Program

```
S=input('Define a set S= ');
meetOp=input('Define an operation meet = ');
joinOp=input('Define an operation join = ');
A=Q16_AbsorptionT(S, meetOp, joinOp);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function isClose = Q16_AbsorptionT(S, meetOp, joinOp)
% S: Set of elements
% meetOp: Function handle for the meet operation
% joinOp: Function handle for the join operation

% Initialize to true
isClose = true;
% Ckeck the operation meet and  join are closed.
for i = 1:length(S)
for j = 1:length(S)
% meet
if ~ismember(meetOp(S(i), S(j)),S)
disp(['Operation meet is not closed for a=',
num2str(S(i)), ' and b= ', num2str(S(j)),]);
isClose = false;
return;
end
% join
if ~ismember(joinOp(S(i),S(j)),S)
disp(['Operation join is not closed for a=',
num2str(S(i)), ' and b= ' ,num2str(S(j)),]);
isClose = false;
return;
end
end
```

```
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Check absorption for meet and join
for i = 1:length(S)
for j = 1:length(S)
% Absorption for meet and join
if meetOp(S(i), joinOp(S(i), S(j))) ~= S(i)
disp(['Failed absorption for meet and join for a=
',num2str(S(i)), ', and b= ' ,num2str(S(j)),]);
isClose = false;
return;
end
if joinOp(S(i), meetOp(S(i), S(j))) ~= S(i)
disp(['Failed absorption for join and meet for
a=',num2str(S(i)),', and b= ' ,num2str(S(j)),]);
isClose = false;
return;
end
end
end

if isClose
disp('The given set satisfies the law of absorption under meet and join');
end
end
```

Input

```
S=[1 4 5 23 56 99];
meet=@(a,b)(a+b^2+3*b+1);
join=@(a,b)(a*b);
```

Output

Operation meet is not closed for a=1 and b= 1

Exercise Problem

Q1. Define a non-empty set and operations 'meet' and 'join' such that it will satisfy the laws of absorption. Verify it using given user defined MATLAB function.
Q2. Define a non empty set with operations 'meet' and 'join' such that it will satisfy first

law of absorption while fails the second law. Verify it by given user defined MATLAB function.
Q3. Write a similar MATLAB program that works for set of characters.

Practical No. 5

Aim

To write a MATLAB program that verifies whether a user defined set with two binary operations ’meet’ and ’join’ satisfies the associative law.

Problem

Define $S = \{1, 2, 4, 10, 20, 25, 50, 100\}$, meet$(a, b) = lcm(a, b)$, join$(a, b) = gcd(a, b)$. Write a MATLAB code to Verify whether the given set with these operations satisfy the associate law.

Theory

Associative law: $a \vee (b \vee c) = (a \vee b) \vee c$; $a \wedge (b \wedge c) = (a \wedge b) \wedge c$. Here $\vee$, and $\wedge$ denotes operation meet and join respectively.

Algorithm

Step1: Define a set S;
Step2: Define an operation ’meet’;
Step3: Define an operation ’join’;
Step4: Recall the user defined function $Q17\_Associative(S, meet, join)$.

Program

```
function isClose = Q17_Associative(S, meetOp, joinOp)
% S: Set of elements
% meetOp: Function handle for the meet operation
% joinOp: Function handle for the join operation

% Initialize to true
isClose = true;
% Ckeck the operation meet and  join are closed.
for i = 1:length(S)
for j = 1:length(S)
% meet
if ~ismember(meetOp(S(i), S(j)),S)
disp(['Operation meet is not closed for a=', num2str(S(i)), ' and b= ',
 num2str(S(j)),]);
isClose = false;
return;
end
% join
if ~ismember(joinOp(S(i),S(j)),S)
disp(['Operation join is not closed for a=', num2str(S(i)), ' and b= '
 ,num2str(S(j)),]);
isClose = false;
return;
end
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check associativity for meet and join
for i = 1:length(S)
for j = 1:length(S)
for k = 1:length(S)
% Associativity for meet
if meetOp(meetOp(S(i), S(j)), S(k)) ~= meetOp(S(i), meetOp(S(j), S(k)))
disp(['Operation meet failed associativity for a=' num2str(S(i)) ', and b= '
 num2str(S(j)) 'and c= ' num2str(S(k))]);
isClose = false;
return;
end
% Associativity for join
if joinOp(joinOp(S(i), S(j)), S(k)) ~= joinOp(S(i), joinOp(S(j), S(k)))
disp(['Operation join failed associativity for a=' num2str(S(i)) ', b= '
 num2str(S(j)) ', and c= ' num2str(S(k))]);
isClose = false;
return;
end
end
end
end

if isClose
disp('The given set satisfies the associative law under meet and join');
end
end
```

Input

```
>> S=[1 2 4 10 20 25 50 100];
>> m=@(a,b)(lcm(a,b));
>> j=@(a,b)(gcd(a,b));
>> Q17_Associative(S,m,j)
```

Output

```
Operation join is not closed for a=10 and b= 25

ans =

logical

0
```

Exercise Problem

Q1. Check the associative law under same operations meet and join by taking $S = \{1, 2, 4, 5, 10, 20, 25, 50, 100\}$.
Q2. Is there any impact on the output if we interchange the definition of meet and join? Verify it by given user defined function.
Q3. Use the given user defined function to verify the associative law on a set and operations defined by yourself.

Practical No. 6

Aim

To write a MATLAB program that verifies the distributive laws of Boolean algebra under two user defined operations 'meet' and 'join'.

Problem

Define $S = \{1, 2, 4, 10, 20, 25\}$, OR$(a, b) = lcm(a, b)$, AND$(a, b) = gcd(a, b)$. Write a MATLAB code to Verify whether the given set with these operations satisfy the distributive law.

Theory

Distributive law: $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$; $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$.

Algorithm

Step1: Define a set S;
Step2: Define an operation 'OR';
Step3: Define an operation 'AND';
Step4: Recall the user defined function $Q18\_Bool\_distri(S, AND, OR)$.

Program

```
function isClose=Q18_Bool_distri(S, AND_Op, OR_Op)
% This function checks whether a given set with defined operations
 is a Boolean algebra.
% S: A vector of elements.
% AND_Op: A function handle for the AND operation.
% OR_Op: A function handle for the OR operation.
isClose = true;
% Check if the set is closed under both AND and OR
for i = 1:length(S)
for j = 1:length(S)
if ~ismember(AND_Op(S(i), S(j)), S)
disp(['Closure property violated under AND operation for a= ',num2str(S(i)),
 ' and b= ',num2str(S(j))]);
isClose=false;
return;
end
if ~ismember(OR_Op(S(i), S(j)), S)
disp(['Closure property violated under OR operation for a= ',num2str(S(i)),
' and b= ',num2str(S(j))]);
isClose=false;
return;
end
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check distributivity of AND over OR and vice versa
for i = 1:length(S)
for j = 1:length(S)
for k = 1:length(S)
if AND_Op(S(i), OR_Op(S(j), S(k))) ~=
OR_Op(AND_Op(S(i), S(j)), AND_Op(S(i), S(k)))
```

```
disp(['Distributivity violated: AND over OR for a= ',num2str(S(i)), '
 and b= ',num2str(S(j)), ' and c= ',num2str(S(k)),]);
isClose=false;
return;
end
if OR_Op(S(i), AND_Op(S(j), S(k))) ~=
AND_Op(OR_Op(S(i), S(j)), OR_Op(S(i), S(k)))
disp(['Distributivity violated: OR over AND for a= ',num2str(S(i)),
' and b= ',num2str(S(j)), ' and c= ',num2str(S(k)),]);
isClose=false;
return;
end
end
end
end

if isClose
disp('The given set satisfies the distributive law under OR and AND');
end
end
```

Input

```
 >> S=[1, 2, 4, 10, 20, 25];
 >> A=@(a,b)(gcd(a,b));
 >> O=@(a,b)(lcm(a,b));
 >> Q18_Bool_distri(S, A, O);
```

Output

```
Closure property violated under OR operation for a= 2 and b= 25

 ans =

 logical

 0
```

Exercise Problem

Q1. Write a similar MATLAB program for set of characters.
Q2. Design a finite set of natural numbers with two binary operations that fails to satisfy the distributive law. Verify it by given user defined MATLAB function.

Practical No. 7

Aim

To write a MATLAB program that verifies the compliment property of Boolean algebra on a user defined set with two binary operations.

Problem

Take $S = \{0, 1\}$, AND and OR are the logical AND and logical OR operation, $i_0 = 0, i_1 = 1$. Write a MATLAB code to verify the compliment property of Boolean algebra on S with these operations.

Theory

Complement property: $a \vee a^c = 0$; $a \wedge a^c = 0$.

Algorithm

Step1: Define a set S;
Step2: Definite binary operations 'and', 'or', unary operations i_0, and i_1;
Step3: Recall the user defined function $Q19\_Bool\_Compli(S, and, or, i_0, i_1)$.

Program

```
function isClose=Q19_Bool_Compli(S, AND_Op, OR_Op,i_0,i_1)
% This function checks whether a given set with defined operations
 is a Boolean algebra.
% S: A vector of elements.
% AND_Op: A function handle for the AND operation.
% OR_Op: A function handle for the OR operation.
isClose = true;
% Check if the set is closed under both AND and OR
for i = 1:length(S)
for j = 1:length(S)
if ~ismember(AND_Op(S(i), S(j)), S)
disp(['Closure property violated under AND operation for a= ',num2str(S(i)),
 ' and b= ',num2str(S(j))]);
isClose=false;
return;
end
if ~ismember(OR_Op(S(i), S(j)), S)
```

```
disp(['Closure property violated under OR operation for a= ',num2str(S(i)),
 ' and b= ',num2str(S(j))]);
isClose=false;
return;
end
end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Check complement property
for i = 1:length(S)
complement_found = false;
for j = 1:length(S)
if AND_Op(S(i), S(j)) == i_0 && OR_Op(S(i), S(j)) == i_1
complement_found = true;
break;
end
end
if ~complement_found
disp(['Complement property violated for element ' num2str(S(i)) '.']);
isClose=false;
return;
end
end
if isClose
disp('The given set satisfies the law of compliment under OR and AND');
end
end
```

Input

```
>> S=[0,1];
>> And=@(a,b) and(a,b);
>> Or=@(a,b) or(a,b);
>> i_0=0;
>> i_1=1;
>> Q19_Bool_Compli(S,And,Or,i_0,i_1)
```

Output

```
The given set satisfies the law of compliment under OR and AND

ans =
logical

1
```

Exercise Problem

Q1. Take the power set $\mathbb{P}(S)$ of a finite non-empty set S of characters. Operation OR as union and operation AND as intersection of two sets, $i_o = \phi, i_1 = S$. Write a MATLAB code that verifies the compliment properties of Boolean algebra on $\mathbb{P}(S)$ with these operations.

Practical No. 8

Aim

To write a MATLAB program that verifies whether a given non-empty set with a user defined operation is a POSET.

Problem

Take $S = \{1, 2, 3, 5, 6, 7, 10, 14, 15, 21, 30, 35, 42, 70, 105, 210\}$, and R, the relation of divisibility. Write a MATLAB code to verify whether (S, R) is a POSET.

Theory

Divisibility: $a|b$ if $\exists$ a number $c \in \mathbb{N}$, such that $b = ac$.

Algorithm

Step1: Define a set S;
Step2: Define a relation R;
Step3: Recall the function $Q1checkPOSET(S, R)$.

Program

```
function isPOSET = Q1checkPOSET(S, relation)
% S: A set of elements (e.g., a vector, matrix, or list of numbers)
% relation: A function handle representing
the binary relation (e.g., @(a,b) a <= b)
```

```
isPOSET = true; % Assume it's a POSET until proven otherwise

% 1. Check Reflexivity: a <= a for all a in S
for i = 1:length(S)
a = S(i);
if ~relation(a, a)  % Check if a <= a
disp(['Failed reflexivity for a = ', num2str(a)]);
isPOSET = false;
return;
end
end

% 2. Check Antisymmetry: If a <= b and b <= a, then a = b
for i = 1:length(S)
for j = 1:length(S)
a = S(i);
b = S(j);
if relation(a, b) && relation(b, a) && a ~= b
disp(['Failed antisymmetry for a = ', num2str(a), ',
 b = ', num2str(b)]);
isPOSET = false;
return;
end
end
end

% 3. Check Transitivity: If a <= b and b <= c, then a <= c
for i = 1:length(S)
for j = 1:length(S)
for k = 1:length(S)
a = S(i);
b = S(j);
c = S(k);
if relation(a, b) && relation(b, c) && ~relation(a, c)
disp(['Failed transitivity for a = ', num2str(a), ', b = ', num2str(b),
 ', c = ', num2str(c)]);
isPOSET = false;
return;
end
end
end
end

% If all checks pass
```

```
disp('The set with the given relation is a POSET.');
end
```

Input

```
>> S=[1,2,3,5,6,7,10,14,15,21,30,35,42,70,105,210];
>> r=@(a,b) rem(b,a)==0;
>> Q1checkPOSET(S,r);
```

Output

```
The set with the given relation is a POSET.
```

Exercise Problem

Q1. Take $S = \{1, 23, 4, 5, 6, 7, 8, 9, 10\}$, R the relation of $\leq$. Verify by the given user defined MATLAB function whether (S, R) is a POSET.
Q2. Define a non empty set S and a relation R which is symmetric but not anti-symmetric on S. verify it by given MATLAB function.

Practical No. 9

Aim

To write a MATLAB program that verifies whether a given non-empty set with two given operations (meet & join) is a Lattice.

Problem

Take $S = \{1, 2, 3, 5, 6, 7, 10, 14, 15, 21, 30, 35, 42, 70, 105, 210\}$, meet$(a, b) = lcm(a, b)$, join$(a, b) = gcd(a, b)$. Write a MATLAB program to verify whether $(S, meet, join)$ is a Lattice.

Theory

Let S be a nonempty set closed under two binary operations called *meet* and *join*, denoted respectively by $\vee$ and $\wedge$. Then $(S, \vee, \wedge)$ is called a lattice if following holds.

Commutative laws: $a \vee b = b \vee a$, $a \wedge b = b \wedge a$.
Associative law: $(a \wedge b) \wedge c = a \wedge (b \wedge c)$, $(a \vee b) \vee c = a \vee (b \vee c)$.
Absorption law: $a \wedge (a \vee b) = a$, $\quad a \vee (a \wedge b) = a$.

Algorithm

Step1: Define a set S;
Step2: Define an operation 'meet';
Step3: Define an operation 'join';
Step4: Recall the user defined function $Q2\_Lattice(S, meet, join)$.

Program

```
function isLattice = Q2_Lattice(S, meetOp, joinOp)
% S: Set of elements
% meetOp: Function handle for the meet operation
% joinOp: Function handle for the join operation

% Initialize isLattice to true
isLattice = true;
% Ckeck the operation meet and  join are closed.
for i = 1:length(S)
for j = 1:length(S)
% meet
if ~ismember(meetOp(S(i), S(j)),S)
disp(['Operation meet is not closed for a=',
 num2str(S(i)), ' and b= ', num2str(S(j)),]);
isLattice = false;
return;
end
% join
if ~ismember(joinOp(S(i),S(j)),S)
disp(['Operation join is not closed for a=',
 num2str(S(i)), ' and b= ' ,num2str(S(j)),]);
isLattice = false;
return;
end
end
end
% Check commutativity for meet and join
for i = 1:length(S)
for j = 1:length(S)
% Commutativity for meet
if meetOp(S(i), S(j)) ~= meetOp(S(j), S(i))
```

```
disp(['Operation meet failed commutativity for
 a=' num2str(S(i)) ', and b= ' num2str(S(j))]);
isLattice = false;
return;
end
% Commutativity for join
if joinOp(S(i), S(j)) ~= joinOp(S(j), S(i))
disp(['Operation join failed commutativity for
 a=' num2str(S(i)) ', and b= ' num2str(S(j))]);
isLattice = false;
return;
end
end
end

% Check associativity for meet and join
for i = 1:length(S)
for j = 1:length(S)
for k = 1:length(S)
% Associativity for meet
if meetOp(meetOp(S(i), S(j)), S(k)) ~=
meetOp(S(i), meetOp(S(j), S(k)))
disp(['Operation meet failed associativity for a=' num2str(S(i)) ',
 and b= ' num2str(S(j)) 'and c= ' num2str(S(k))]);
isLattice = false;
return;
end
% Associativity for join
if joinOp(joinOp(S(i), S(j)), S(k)) ~=
joinOp(S(i), joinOp(S(j), S(k)))
disp(['Operation join failed associativity for a=' num2str(S(i)) ',
 b= ' num2str(S(j)) ', and c= ' num2str(S(k))]);
isLattice = false;
return;
end
end
end
end

% Check idempotence for meet and join
for i = 1:length(S)
% Idempotence for meet
if meetOp(S(i), S(i)) ~= S(i)
disp(['Operation meet failed idempotence for a=' num2str(S(i))]);
isLattice = false;
```

```
return;
end
% Idempotence for join
if joinOp(S(i), S(i)) ~= S(i)
disp(['Operation join failed idempotence for a=' num2str(S(i))]);
isLattice = false;
return;
end
end

% Check absorption for meet and join
for i = 1:length(S)
for j = 1:length(S)
% Absorption for meet and join
if meetOp(S(i), joinOp(S(i), S(j))) ~= S(i)
disp(['Failed absorption for meet and join for a= ',num2str(S(i)),
 ', and b= ' ,num2str(S(j)),]);
isLattice = false;
return;
end
if joinOp(S(i), meetOp(S(i), S(j))) ~= S(i)
disp(['Failed absorption for join and meet for a= ',num2str(S(i)),
 ', and b= ' ,num2str(S(j)),]);
isLattice = false;
return;
end
end
end

if isLattice
disp('The given set with the given operations is a Lattice.');
else
disp('The given set with the given operations is not a Lattice.');
end
end
```

Input

```
>> S=[1,2,3,5,6,7,10,14,15,21,30,35,42,70,105,210];
>> m=@(a,b)(lcm(a,b));
>> J=@(a,b)(gcd(a,b));
>> Q2_Lattice(S,m,J);
```

Output

The given set with the given operations is a Lattice.

Exercise Problem

Q1. Take the set $S = \{1, 2, 3, 5, 6, 7, 10, 14, 15, 21, 30, 35, 42, 70, 105, 210\}$, and define $meet(a, b) = a + b$, and $join(a, b) = a \cdot b$. Verify by the user defined MATLAB function whether $(S, meet, join)$ is a Lattice.
Q2. Define a finite non empty set of numbers, and two binary operation of your choice. Verify whether your set with defined operations is a Lattice. Use the defined MATLAB function for your verification.

Practical No. 10

Aim

To write a MATLAB program that verifies whether a given non-empty set with some given operations is a Boolean algebra.

Problem

Take $S = \{1, 2, 3, 5, 6, 7, 10, 14, 15, 21, 30, 35, 42, 70, 105, 210\}$, meet$(a, b) = lcm(a, b)$, join$(a, b) = gcd(a, b)$, $i_o = 1$, and $i_1 = 210$. Write a MATLAB program to verify whether $(S, meet, join, i_0, i_1)$ is a Boolean algebra.

Theory

A non empty set S with two binary operations $\vee$ and $\wedge$, element 0 and 1, an unary operation $'$ is a Boolean algebra if following holds.
Identity Laws: $x \vee 0 = x, x \wedge 1 = x, \forall x \in S$.
Complement laws: $x \vee x^{'} = 1; x \wedge x^{'} = 0$.
Associative laws: $(x \vee y) \vee z = x \vee (y \vee z); (x \wedge y) \wedge z = x \wedge (y \wedge z)$.
Commutative laws: $x \vee y = y \vee x, \quad x \wedge y = y \wedge x$.
Distributive laws: $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z), \quad x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$.

Algorithm

Step1: Define a set S;
Step2: Define an operation 'AND_Op';

Step3: Define an operation '*OR_Op*';
Step4: Define the identity element i_0;
Step5: Define the compliment i_1;
Step4: Recall the user defined function $Q3BooleanAlg(S, AND\_Op, OR\_Op, i_0, i_1)$.

Program

```
function isBooleanAlgebra=Q3BooleanAlg(S, AND_Op, OR_Op,i_0,i_1)
% This function checks whether a given set with defined
operations is a Boolean algebra.
% S: A vector of elements.
% AND_Op: A function handle for the AND operation.
% OR_Op: A function handle for the OR operation.
isBooleanAlgebra = true;
% Check if the set is closed under both AND and OR
for i = 1:length(S)
for j = 1:length(S)
if ~ismember(AND_Op(S(i), S(j)), S)
disp(['Closure property violated under AND operation
for a= ',num2str(S(i)), ' and b= ',num2str(S(j))]);
isBooleanAlgebra=false;
return;
end
if ~ismember(OR_Op(S(i), S(j)), S)
disp(['Closure property violated under OR operation
for a= ',num2str(S(i)), ' and b= ',num2str(S(j))]);
isBooleanAlgebra=false;
return;
end
end
end

% Check associativity of AND and OR
for i = 1:length(S)
for j = 1:length(S)
for k = 1:length(S)
if AND_Op(AND_Op(S(i), S(j)), S(k)) ~= AND_Op(S(i), AND_Op(S(j), S(k)))
disp(['Associativity violated under AND operation for a= ',num2str(S(i)),
 ' and b= ',num2str(S(j)) ' and c= ',num2str(S(k))]);
isBooleanAlgebra=false;
return;
end
if OR_Op(OR_Op(S(i), S(j)), S(k)) ~= OR_Op(S(i), OR_Op(S(j), S(k)))
disp(['Associativity violated under OR operation for a= ',num2str(S(i)),
 ' and b= ',num2str(S(j)) ' and c= ',num2str(S(k))]);
isBooleanAlgebra=false;
return;
end
end
end
end

% Check commutativity of AND and OR
for i = 1:length(S)
for j = 1:length(S)
if AND_Op(S(i), S(j)) ~= AND_Op(S(j), S(i))
disp(['Commutativity violated under AND operation for a= ',num2str(S(i)),
 ' and b= ',num2str(S(j)),]);
isBooleanAlgebra=false;
return;
end
if OR_Op(S(i), S(j)) ~= OR_Op(S(j), S(i))
disp(['Commutativity violated under OR operation for a= ',num2str(S(i)),
 ' and b= ',num2str(S(j)),]');
isBooleanAlgebra=false;
return;
end
end
end

% Check distributivity of AND over OR and vice versa
for i = 1:length(S)
for j = 1:length(S)
for k = 1:length(S)
if AND_Op(S(i), OR_Op(S(j), S(k))) ~=
OR_Op(AND_Op(S(i), S(j)), AND_Op(S(i), S(k)))
disp(['Distributivity violated: AND over OR for a= ',num2str(S(i)),
 ' and b= ',num2str(S(j)), ' and c= ',num2str(S(k)),]);
isBooleanAlgebra=false;
return;
end
if OR_Op(S(i), AND_Op(S(j), S(k))) ~=
AND_Op(OR_Op(S(i), S(j)), OR_Op(S(i), S(k)))
disp(['Distributivity violated: OR over AND for a= ',num2str(S(i)),
 ' and b= ',num2str(S(j)), ' and c= ',num2str(S(k)),]);
isBooleanAlgebra=false;
return;
end
end
```

```
end
end

% Check for identity elements 0 and 1
%i_0 = find(S == 0);
%i_1 = find(S == 1);
if isempty(i_0) || isempty(i_1)
disp('Missing identity elements 0 or 1.');
isBooleanAlgebra=false;
return;
end
for i = 1:length(S)
if AND_Op(S(i), i_1) ~= S(i) || OR_Op(S(i), i_0) ~= S(i)
disp('Identity element property violated.');
isBooleanAlgebra=false;
return;
end
end

% Check complement property
for i = 1:length(S)
complement_found = false;
for j = 1:length(S)
if AND_Op(S(i), S(j)) == i_0 && OR_Op(S(i), S(j)) == i_1
complement_found = true;
break;
end
end
if ~complement_found
disp(['Complement property violated for element ' num2str(S(i)) '.']);
isBooleanAlgebra=false;
return;
end
end

disp('The given set with the given operations is a Boolean Algebra.');
end
```

Input

```
>> S=[1,2,3,5,6,7,10,14,15,21,30,35,42,70,105,210];
>> m=@(a,b)(lcm(a,b));
>> J=@(a,b)(gcd(a,b));
>> i0=1;
>> i1=210;
>> Q3BooleanAlg(S,m,J,i0,i1);
```

Output

```
Identity element property violated.
```

Exercise Problem

Q1. Take the same set S, operation meet and join as it is. Define $i_0 = 210, i_1 = 1$ and verify by the defined MATLAB function whether $(S, meet, join, i_0, i_1)$ is a Boolean algebra.
Q2. write a similar MATLAB program that works for finite set of characters.

Practical No. 11

Aim

To write a MATLAB program that display all the max terms of a given Boolean expression. Here user will supply the truth table in array as input.

Problem

Write a MATLAB code that finds the max terms of the Boolean expression of three variables represented by following truth table.

p	q	r	Output
1	1	1	1
1	1	0	1
1	0	1	1
0	1	1	0
1	0	0	1
0	0	1	0
0	1	0	1
0	0	0	0

Theory

Maxterm: A maxterm is a sum of variables in Boolean function in which every variable is present either in normal or in complement form.

Algorithm

Step1: Run the given MATLAB code;
Step2: Give the input of number of variables (n) as 3;
Step3: Give the input of last column of truth table in array form;
Step 4 See the output in command window.

Program

```
% Function to find and display maxterms

% Input: Define the number of variables
n = input('Enter the number of variables: ');

% Input: Define the truth table (1s and 0s corresponding
 to f(A1, A2, ..., An))
truth_table = input('Enter the truth table as a binary
array (e.g., [1 0 1 0 1 0 1 1]): ');

% Check if the truth table length matches 2^n
if length(truth_table) ~= 2^n
error('Truth table size must be 2^n.');
end

% Initialize variables (A, B, C, etc.)
variables = cell(1, n);
for i = 1:n
variables{i} = char('A' + i - 1); % Generate variable names A, B, C, etc.
end

% Maxterms Initialization
maxterms = {};

% Loop through the truth table and find maxterms
for i = 0:(2^n - 1)
bin_rep = dec2bin(i, n) - '0'; % Convert number to binary form

% If the function output is 0, it's a maxterm
if truth_table(i+1) == 0
```

```
maxterm = '';
for j = 1:n
if bin_rep(j) == 1
maxterm = [maxterm, variables{j}, ' + '];  % Normal variable (OR form)
else
maxterm = [maxterm, variables{j}, '''', ' + '];  % Complement (OR form)
end
end
maxterms{end+1} = maxterm(1:end-3); % Remove the last ' + '
end
end

% Display the Maxterms
disp('Maxterms:');
for i = 1:length(maxterms)
disp(maxterms{i});
end
```

Input

```
>> Q5_Maxterms
Enter the number of variables: 3;
Enter the truth table as a binary array (e.g., [1 0 1 0 1 0 1 1]):
[1 1 1 0 1 0 1 0];
```

Output

```
The maxterms are:
{'A + B' + C''}    {'A' + B + C''}    {'A' + B' + C''}
```

Exercise Problem

Q1. Write a similar MATLAB code that takes every entry of truth tables as input and find the max terms of the given Boolean expression.
Q2. Write a MATLAB program that takes the Boolean expression as input and display the truth table of that Boolean expression.

Practical No. 12

Aim

To write a MATLAB program that display all the min terms of a given Boolean expression. Here user will supply the truth table in array as input.

Problem

Write a MATLAB program that display all the min terms of the Boolean expression given by following truth table.

p	q	r	Output
1	1	1	0
1	1	0	1
1	0	1	0
0	1	1	0
1	0	0	1
0	0	1	0
0	1	0	1
0	0	0	1

Theory

Min term: A min term is a product of variables in Boolean function in which every variable is present either in normal or in complement form.

Algorithm

Step1: Run the code;
Step2: Give the input of complete truth table as a matrix;
Step3: See the output in command window.

Program

```
% Input: Define the number of variables
[truth_table] = input('Enter the truth table as a matrix: ');

Q4_MintermsT(truth_table);
function Q4_MintermsT(truth_table)
% truth_table is a matrix where the last column represents the output
% and the other columns represent the input variables
% Example input for 3 variables: [A B C Output]
% A B C Output
% 0 0 0    0
% 0 0 1    1
% 0 1 0    1
% 0 1 1    0
% 1 0 0    1
% 1 0 1    0
% 1 1 0    0
% 1 1 1    1

% Get the number of variables (columns excluding the last column for the output)
[num_rows, num_cols] = size(truth_table);
num_vars = num_cols - 1;

% Prepare a list of minterms
minterms = {};

% Iterate over the truth table to identify the minterms (rows with output = 1)
for i = 1:num_rows
if truth_table(i, end) == 1  % Output column is 1
% Create the minterm for this row
minterm = '';
for j = 1:num_vars
if truth_table(i, j) == 1
minterm = [minterm, char('A' + j - 1)];  % Variable A, B, C, ...
else
minterm = [minterm, char('A' + j - 1),''''];  % Inverse A', B', C', ...
end
if j < num_vars
minterm = [minterm, ' * '];
end
end
minterms{end+1} = minterm;
end
end

% Display the minterms
disp('The minterms corresponding to the output 1 are:');
for i = 1:length(minterms)
disp(minterms{i});
end
end
```

Input

```
>> Q4_Minterms
Enter the truth table as a matrix: [1 1 1 0;1 1 0 1;1 0 1 0;0 1 1 0;
1 0 0 1;0 0 1 0;0 1 0 1;0 0 0 1];
```

Output

```
The minterms corresponding to the output 1 are:
A * B * C'
A * B' * C'
A' * B * C'
A' * B' * C'
```

Exercise Problem

Q1. Write a similar MATLAB program that takes input of Boolean expression directly in place of truth table.
Q2. Write a MATLAB program that takes all min terms of Boolean expression as input and display the truth table of that Boolean expression.

Practical No. 13

Aim

To write a MATLAB code that minimize and display a user given Boolean expression.

Problem

Write a MATLAB code that simplifies the expression of a Boolean function of three variables that gives output 1 in the row number 1, 3, 5, 6, and 7 in its truth table represented in standard order.

Theory

Min term: A min term is a product of variables in Boolean function in which every variable is present either in normal or in complement form.

Algorithm

Step1: Call the function $Q20\_Bool\_fminimizer()$;
Step2: Inter the number of variables (n);
Step3: Inter the rows numbers (in vector form) of truth table that gives output 1;
Step4: See the output in command window.

Program

```
function Q20_Bool_fminimizer()
% Prompt the user for input: number of variables and minterms
num_vars = input('Enter the number of
variables (e.g., 2 for A,B, 3 for A,B,C): ');
minterms = input('Enter the minterms as a
vector (e.g., [1, 3, 5, 7]): ');

% Create symbolic variables
vars = sym('A', [1 num_vars]);

% Initialize the Boolean function as 0
f = 0;

% Create the Boolean expression by summing the minterms
for i = 1:length(minterms)
term = 1;  % Initialize each minterm as 1 (ANDed later)
minterm = minterms(i);  % Get the current minterm

% Generate the corresponding product for the minterm
for j = 1:num_vars
if bitget(minterm, j) == 1
term = term & vars(j);  % If bit is 1, include the variable
else
term = term & ~vars(j);  % If bit is 0, include the negation
end
end

% Add the current term (AND of variables) to the function
f = f | term;  % OR all the terms together
end

% Simplify the Boolean expression
simplified_f = simplify(f);

% Display the simplified Boolean function
disp('Simplified Boolean function:');
disp(simplified_f);
end
```

Input

```
>> Q20_Bool_fminimizer
Enter the number of variables (e.g., 2 for A,B, 3 for A,B,C): 3;
Enter the minterms as a vector (e.g., [1, 3, 5, 7]): [1 3 5 6 7];
```

Output

```
Simplified Boolean function:
A1 & ~A2 | A2 & A3 | A1 & A2 & ~A3
```

Exercise Problem

Q1. Using above MATLAB code, check the output of a Boolean expression of four variables and a truth table of your choice.
Q2. Write a similar MATLAB code that takes input of truth table outputs in binary form and display the simplified expression of given Boolean function.

Practical No. 14

Aim

To write a MATLAB code that generates the canonical form (sum of products (SOP)) of given Boolean expression.

Problem

Write a MATLAB program that display the canonical form (sum of products) of the user given truth table of a Boolean expression.

Theory

Canonical form (SOP): The sum of minterms that represents the Boolean function is called the sum-of-products expansion or the disjunctive normal form of the Boolean function.

Algorithm

Step1: Recall the user defined MATLAB function *Q7_Canonicalform_SOP*();
Step2: Define the number of variables (n);
Step3: Give the input of last column (output) of truth table of given Boolean expression;
Step4: See the output in command window of MATLAB.

Program

```
% Function to generate Canonical Sum of Products (SOP) form
function Q7_Canonicalform_SOP()
% Step 1: Get the number of variables from the user
numVars = input('Enter the number of variables: ');

% Step 2: Get the truth table output from the user
% The output is a binary vector (0s and 1s) representing the truth table
numRows = 2^numVars;  % Total number of rows in the truth table
output = zeros(1, numRows);

fprintf('Enter the truth table output values (0s and 1s):\n');
for i = 1:numRows
output(i) = input(['Output for row ', num2str(i), ': ']);
end

% Step 3: Generate the canonical Sum of Products (SOP) form
SOP = '';
for i = 1:numRows
if output(i) == 1
% Generate the product term corresponding to the current row
term = '';
for j = 1:numVars
% Check if the j-th variable is 1 or 0 for this row
if bitget(i-1, numVars-j+1) == 1
term = strcat(term, char(65 + j - 1));  % 'A', 'B', 'C', ...
else
term = strcat(term, char(65 + j - 1), '''');
end
end
% Append the term to SOP, ensuring the correct format
if isempty(SOP)
SOP = term;
else
SOP = strcat(SOP, ' + ', term);
end
end
```

```
end

% Step 4: Display the canonical SOP form
disp('Canonical Sum of Products (SOP) form:');
disp(SOP);
end
```

Input

```
>> Q7_Canonicalform_SOP
Enter the number of variables: 3
Enter the truth table output values (0s and 1s):
Output for row 1: 1
Output for row 2: 0
Output for row 3: 1
Output for row 4: 1
Output for row 5: 0
Output for row 6: 0
Output for row 7: 1
Output for row 8: 1
```

Output

```
Canonical Sum of Products (SOP) form:
A'B'C' +A'BC' +A'BC +ABC' +ABC
```

Exercise Problem

Q1. Write a MATLAB code that takes the general expression without using truth table of a Boolean function as input and display the canonical form (SOP).
Q2. Verify the output of defined MATLAB function for five known boolean expression of your choice.

Practical No. 15

Aim

To write a MATLAB code that generates the canonical form (product of sums (POS)) of given Boolean expression.

Problem

Write a MATLAB program that display the canonical form (product of sums) of the user given truth table of a Boolean expression.

Theory

Canonical form (POS): The products of max terms that represents the Boolean function is called the products of sum expansion or the conjunctive normal form of the Boolean function.

Algorithm

Step1: Recall the user defined MATLAB function *Q11_Canonical_POS*();
Step2: Define the number of variables (n);
Step3: Give the input of last column (output) of truth table of given Boolean expression;
Step4: See the output in command window of MATLAB.

Program

```
function pos_form = Q11_Canonical_POS()
% Step 1: Get number of variables
num_vars = input('Enter the number of variables: ');

% Step 2: Get the truth table from the user
% Number of rows in the truth table is 2^num_vars
num_rows = 2^num_vars;

fprintf('Enter the truth table (first %d columns for variable values,
 last column for output):\n', num_vars);

% Initialize truth table
truth_table = zeros(num_rows, num_vars + 1);

% Get input for the truth table
for i = 1:num_rows
row_input = input(['Row ', num2str(i), ' (enter ', num2str(num_vars), '
 inputs followed by output value): ']);
```

```
truth_table(i, :) = row_input;
end

% Step 3: Initialize the POS form
pos_form = '';

% Step 4: Loop through each row of the truth table
for i = 1:num_rows
% If output is 0, generate the sum term for this row
if truth_table(i, end) == 0
% Initialize the sum term
sum_term = '';

% Generate the sum term for this row
for j = 1:num_vars
if truth_table(i, j) == 1
% If the variable is 1, do not negate it
sum_term = [sum_term, char('A' + j - 1)];
else
% If the variable is 0, negate it
sum_term = [sum_term, char('A' + j - 1), ''''];
end
% Add '+' between variables in the sum term
if j < num_vars
sum_term = [sum_term, ' + ']
end
end

% Add the sum term to the POS form
if isempty(pos_form)
pos_form = sum_term;
pos_form=['(',pos_form,')'];
else
pos_form = [pos_form, ' * ', '(',sum_term,')']; % Add '*' between sum terms
end
end
end

% Step 5: Output the final POS expression
disp('The Product of Sums (POS) Canonical Form is:');
disp(pos_form);
end
```

Input

```
>> Q11_Canonical_POS
Enter the number of variables: 3
Enter the truth table (first 3 columns for variable
values, last column for output):
Row 1 (enter 3 inputs followed by output value): [1 1 1 0]
Row 2 (enter 3 inputs followed by output value): [1 1 0 1]
Row 3 (enter 3 inputs followed by output value): [1 0 1 1]
Row 4 (enter 3 inputs followed by output value): [0 1 1 0]
Row 5 (enter 3 inputs followed by output value): [1 0 0 1]
Row 6 (enter 3 inputs followed by output value): [0 0 1 0]
Row 7 (enter 3 inputs followed by output value): [1 0 1 1]
Row 8 (enter 3 inputs followed by output value): [0 0 0 1]
```

Output

```
The Product of Sums (POS) Canonical Form is:
(A + B + C) * (A' + B + C) * (A' + B' + C)

ans =

'(A + B + C) * (A' + B + C) * (A' + B' + C)'
```

Exercise Problem

Q1. Verify the result by taking a Boolean expression of four variables.
Q2. Write a similar MATLAB program that takes SOP as input and returns the POS as the output.

Practical No. 16

Aim

To write a MATLAB program that verifies whether a given POSET is a Lattice. Here concept of greatest lower bound and least upper bound should be used in the program.

Problem

Write a MATLAB program that takes relation matrix as input and verifies whether a given POSET is Lattice.

Theory

A non empty set S with relation R is a POSET (partially ordered set), if it is reflexive, anti-symmetric, and transitive.

Algorithm

Step1: Take a non empty set S, such that $|S| = n$;
Step2: Define a binary matrix $M = (m_{ij})_{n\times n}$ where $m_{ij} = 1$, if i^{th} element of S is related to j^{th} element of S, otherwise $m_{ij} = 0$;
Step3: Recall the function $Q12\_Checklattice\_byPOSET(M)$;
Step4: See the output in command window.

Program

```
function is_lattice = Q12_CheckLattice_byPOSET(POSET)
%This function takes POSET matrix as input variable.
% Check if a given POSET is a lattice.
% Input: POSET - A matrix representing the partial order
% Output: is_lattice - Boolean (1 for lattice, 0 for non-lattice)

% Get the number of elements in the POSET
n = size(POSET, 1);

% Initially assume that the POSET is a lattice
is_lattice = true;

% Loop through all pairs of elements in the POSET
for i = 1:n
for j = 1:n
if i ~= j
% Check for the join (LUB) and meet (GLB) of elements i and j
% Find the join (LUB) of elements i and j
join_exists = false;
meet_exists = false;

for k = 1:n
if POSET(i, k) == 1 && POSET(j, k) == 1
```

```
join_exists = true;
end
if POSET(k, i) == 1 && POSET(k, j) == 1
meet_exists = true;
end
end

% If join or meet does not exist for any pair, it's not a lattice
if ~join_exists || ~meet_exists
is_lattice = false;
disp('The given POSET is not a Lattice.')
return;
end
end
end
end

% If all pairs have meet and join, the POSET is a lattice
disp('The POSET is a lattice.');
end
```

Input

```
>> M=[1 1 0;0 1 0;1 0 1];
>> Q12_CheckLattice_byPOSET(M)
```

Output

```
The given POSET is not a Lattice.

ans =

logical

0
```

Exercise Problem

Q1. Write a similar MATLAB code that display the least upper bound (LUB), and greatest lower bound (GLB) for every subset of a given POSET. It also display the comment if it

does not exist for some subsets.
Q2. Write a MATLAB program that display the directed graph of a given POSET.

Practical No. 17

Aim

To write a MATLAB program that finds homogeneous solution, particular solution, and total solution of a given system of linear algebraic equations.

Problem

Write a MATLAB program that display the homogeneous solutions, particular solution, and total solutions of a given system of linear algebraic equations.

Theory

A system of linear algebraic equations $AX = b$ is consistent if rank(A)=rank(A:b). A consistent system has unique solution if rank(A)=rank(A:b)=number of variables/unknowns and has infinitely many solutions if rank(A)=rank(A:b)< number of unknowns.

Algorithm

To find the solution of a system of linear algebraic equations $AX = b$, take the following steps.
Step1: Define a matrix A;
Step2: Define a vector b;
Step3: recall the function $Q10\_solve\_system(A, b)$;
Step4: See the output in command window.

Program

```
function [x_homogeneous, x_particular, x_total] = Q10_solve_system(A, b)
% Input:
% A: Coefficient matrix (2x2 for example)
% b: Right-hand side vector (2x1 for example)

% Step 1: Find the Homogeneous Solution A*x = 0
% The homogeneous system is A * x_h = 0
% Solve for x_h using null(A), which gives the null space of A
```

```
x_homogeneous = null(A);

% Step 2: Find the Particular Solution A * x = b
% Solve the system A * x = b using the left division operator \
if rank(A) == length(b)  % Ensure A is solvable for b
x_particular = A \ b; % Use MATLAB's backslash operator
 for solving linear systems
else
x_particular = 'No unique particular solution, system is
inconsistent or underdetermined.';
end

% Step 3: Find the Total Solution: x = x_homogeneous + x_particular
% The total solution is the sum of homogeneous and particular solutions.
% If the system has a unique particular solution, compute the total solution.
if ischar(x_particular)
x_total = x_particular; % Return error message if no unique
solution exists
else
x_total = x_particular + x_homogeneous; % Add the homogeneous
solution to the particular one
end
% Display the results
disp('Homogeneous Solution:');
disp(x_homogeneous);
disp('Particular Solution:');
disp(x_particular);
disp('Total Solution:');
disp(x_total);
end
```

Input

```
 % Define the system of equations
 A = [1 2 3;4 5 6;7 8 9];
 b = [5; 10; 15];

 % Call the function to solve the system
 [x_homogeneous, x_particular, x_total] = Q10_solve_system(A, b);
```

Output

```
Homogeneous Solution:
0.4082
-0.8165
0.4082

Particular Solution:
No unique particular solution, system is inconsistent or underdetermined.
Total Solution:
No unique particular solution, system is inconsistent or underdetermined.
```

Exercise Problem

Q1. Take a system of linear algebraic equations $AX = b$ that is consistent. Verify the solution of defined MATLAB function.
Q2. Take a system of equations in which coefficient matrix is of size 3×4. Verify the solution by defined MATLAB function $Q10\_solve\_system(A, b)$.

Practical No. 18

Aim

To write a MATLAB program that plots the diagram of deterministic finite automaton (DFA).

Problem

Write a MATLAB function that takes states, alphabets, transition matrix, start state, and accept states as input and display the deterministic finite automaton (DFA) diagram as output.

Theory

Deterministic finite automaton (DFA) is defined as 5 tuples

$$DFA = (Q, \Sigma, \delta, q_0, F).$$

Here,
Q is a finite set of states.

Σ is the finite set of input symbols, known as alphabet.
δ is the transition function, a mapping $\delta : Q \times \Sigma \to Q$, which specifies the next state for a given current state and input symbol.
$q_0 \in Q$ is the start state in which the automaton begins when processing at input string.
$F \subseteq Q$ is the set of accepting (or final) states, where the automaton can accept the input string.

Algorithm

Step1: Define a states Q;
Step2: Define alphabet δ;
Step3: Define a transition matrix M;
Step4: Define the start state s;
Step5: Define the accept state a;
Step6: Recall the function $Q9\_plot\_dfa(Q, \delta, M, s, a)$;
Step7: See the output plotted graph.

Program

```
function Q9_plot_dfa(states, alphabet,
transitions, start_state, accept_states)
%This function is designed to  plot the Diagram of DFA determined by an
%Automaton.
% states: List of all states in the DFA (e.g., 1, 2, 3,...)
% alphabet: List of symbols in the alphabet (e.g., {'a', 'b'})
% transitions: Transition matrix, where transitions(i, j)
 is the next state from state i on input alphabet(j)
% start_state: The initial state (integer)
% accept_states: A vector of accept states (e.g., [2, 4])

% Number of states
num_states = length(states);
num_alphabet = length(alphabet);

% Create a directed graph for the DFA
edges = [];
labels = {};

% Iterate over all states and their transitions
for i = 1:num_states
for j = 1:num_alphabet
next_state = transitions(i, j);
if next_state ~= 0  % Ignore transitions to non-existent states
edges = [edges; i, next_state];
```

```
labels = [labels; alphabet{j}];
end
end
end

% Create a directed graph
G = digraph(edges(:, 1), edges(:, 2));

% Plot the DFA
figure;
p = plot(G, 'Layout', 'circle', 'NodeLabel', states);

% Highlight the start state
highlight(p, start_state, 'NodeColor', 'g', 'MarkerSize', 7);

% Highlight the accept states
highlight(p, accept_states, 'NodeColor', 'r', 'MarkerSize', 7);

% Set up labels for the transitions
edge_labels = cell(size(edges, 1), 1);
for i = 1:length(labels)
edge_labels{i} = labels{i};
end

% Add the edge labels (transition symbols)
labelnode(p, 1:num_states, num2cell(states));
l = addedge(p, edges(:,1), edges(:,2), edge_labels);

% Title
title('DFA Diagram');

% Customize plot appearance
axis off;
set(gca, 'XTick', []);
set(gca, 'YTick', []);

% Add a legend or annotations if needed
legend('Start State', 'Accept State', 'Location', 'Best');
end
```

Input

```
>>  states = [1, 2, 3]; % States: 1 = q0, 2 = q1, 3 = q2
alphabet = {'0', '1'}; % Alphabet: {0, 1}
transitions = [2, 1; 2, 3; 2, 1]; % Transition matrix:
% 1 -> (0) -> 2, (1) -> 1
% 2 -> (0) -> 2, (1) -> 3
% 3 -> (0) -> 2, (1) -> 1

start_state = 1; % Start state: q0
accept_states = 3; % Accept state: q2

% Plot the DFA diagram
Q9_plot_dfa(states, alphabet, transitions, start_state, accept_states);
```

Output

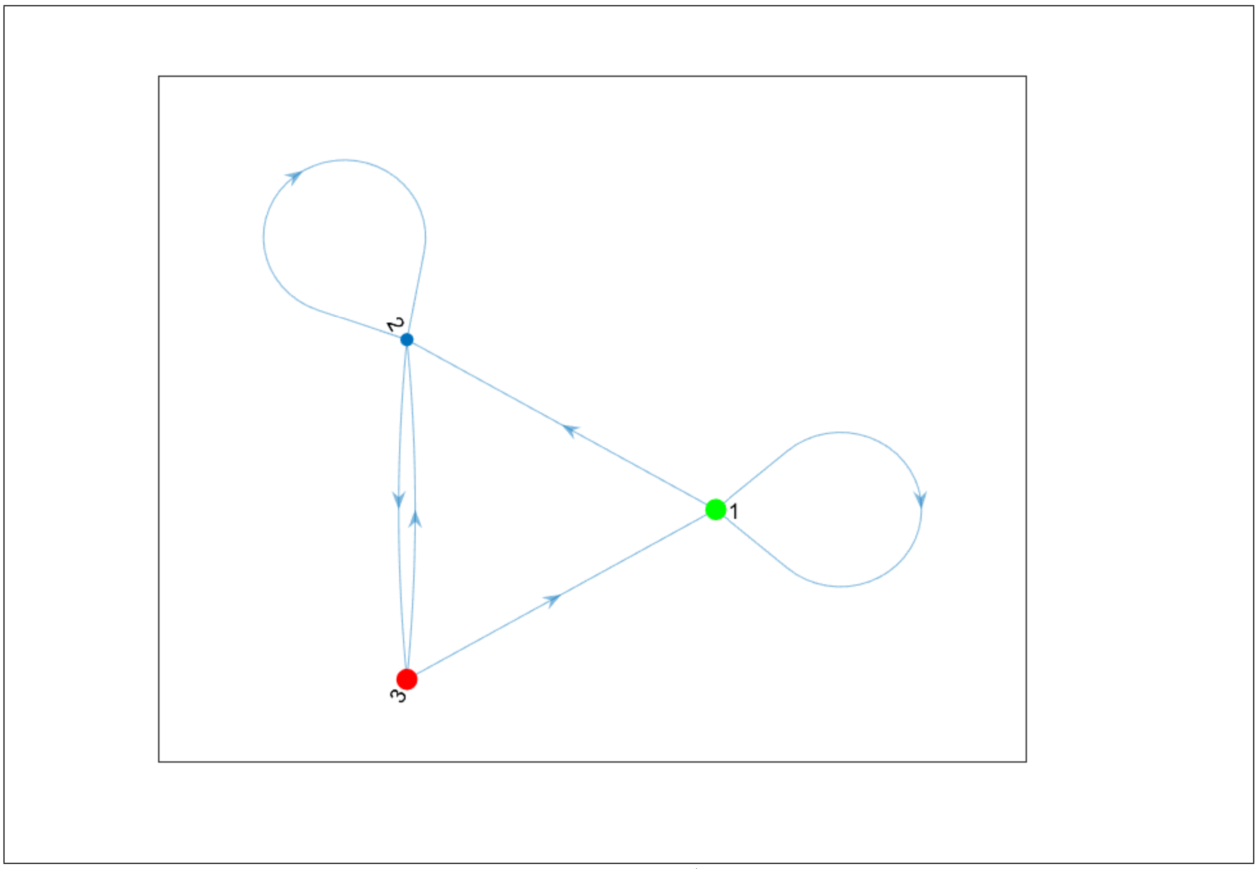


Figure 7.1: DFA graph

Exercise Problem

Q1. Using defined MATLAB function, display the DFA graph for the following statistics. states = $[1, 2, 3]$, alphabet = $\{a, b\}$, transition matrix = $[2, 3; 1, 2; 3, 1]$, start state = 1, and accept states = 2.

Practical No. 19

Aim

To write a MATLAB program that analyze the asymptotic behavior of a user defined function.

Problem

Write a MATLAB program to analyze the asymptotic behavior of user defined numerical functions.

Theory

Behavior of function when input variable goes towards infinity is called asymptotic behavior of that function.

Algorithm

Step1: Run the given MATLAB code;
Step2: Give the expression of function that will be asked by program as input;
Step3: see the output plotted graph.

Program

```
% MATLAB program to analyze the asymptotic behavior of
 a user-defined function

% Prompt user for input function
func_str = input('Enter a function f(x) in terms
of x (e.g., "sin(x)/x"): ', 's');

% Convert the function string to an anonymous function handle
f = str2func(['@(x) ' func_str]);

% Define the range of x (use logspace fo better handling of large
 and small x values)
x_vals = logspace(-1, 3, 100); % x from 10^-1 to 10^3
y_vals = zeros(size(x_vals)); % Initialize y values

% Handle any potential issues with division by zero or undefined values
for i = 1:length(x_vals)
```

```
try
y_vals(i) = f(x_vals(i));
catch
y_vals(i) = NaN; % If the function is undefined, store NaN
end
end

% Plot the function
figure;
semilogx(x_vals, y_vals, 'LineWidth', 2);
grid on;
xlabel('x');
ylabel('f(x)');
title(['Asymptotic Behavior of f(x) = ', func_str]);
xlim([min(x_vals) max(x_vals)]);
ylim([-1.5 1.5]); % Adjust y-limits to zoom in on
typical function behavior

% Display asymptotic behavior in the command window
fprintf('Asymptotic behavior of the function as x -> infinity:\n');
try
y_inf = f(1e3);
fprintf('f(x) as x -> infinity: %.5f\n', y_inf);
catch
fprintf('f(x) as x -> infinity is undefined\n');
end

fprintf('Asymptotic behavior of the function as x -> 0:\n');
try
y_zero = f(1e-3);
fprintf('f(x) as x -> 0: %.5f\n', y_zero);
catch
fprintf('f(x) as x -> 0 is undefined\n');
end
```

Input

```
>> Q8_Asymptotic_Behavior_f
Enter a function f(x) in terms of x (e.g., "sin(x)/x"): sin(x)/x
```

Output

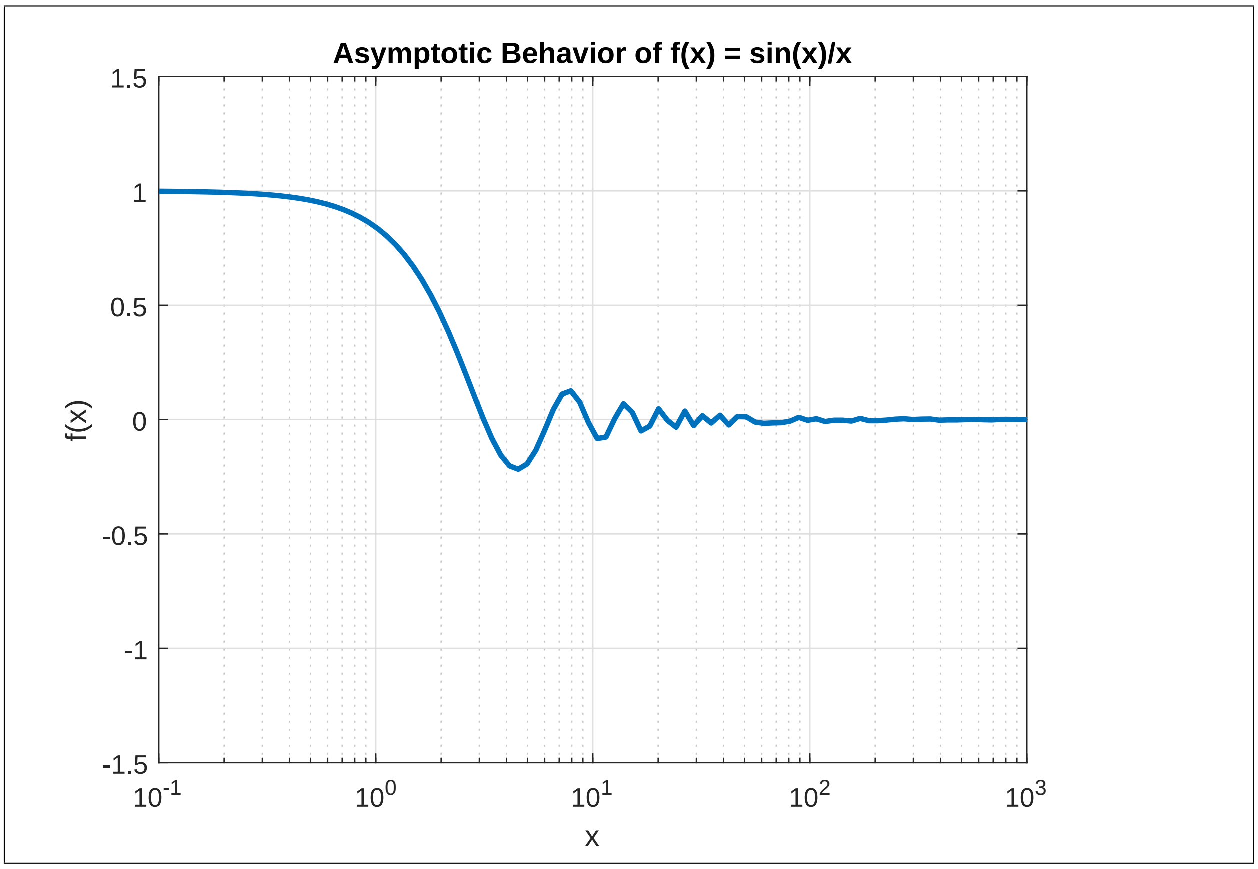


Figure 7.2: Asymptotic graph of $\frac{\sin(x)}{x}$.

Exercise Problem

Q1. Use the written MATLAB program and analyze the asymptotic behavior of $\frac{\cos(x)}{x}$.
Q2. Define a numerical function of your choice and analyze the asymptotic behavior of that function using written MATLAb code.

Practical No. 20

Aim

To write a MATLAB program to find a reduced machine (that reduces the states in a deterministic finite automation (DFA)).

Problem

Write a MATLAB program that takes input from user and minimize the number of states in DFA.

Theory

Deterministic finite automaton (DFA) is defined as 5 tuples

$$DFA = (Q, \Sigma, \delta, q_0, F).$$

Here,
Q is a finite set of states.
Σ is the finite set of input symbols, known as alphabet.
δ is the transition function, a mapping $\delta : Q \times \Sigma \to Q$, which specifies the next state for a given current state and input symbol.
$q_0 \in Q$ is the start state in which the automaton begins when processing at input string.
$F \subseteq Q$ is the set of accepting (or final) states, where the automaton can accept the input string.

Algorithm

Step1: Define a state S;
Step2: Define alphabet a;
Step3: Define a transition matrix M;
Step4: Define initial state $s1$;
Step5: Define final state $s2$;
Step6: Recall the user defined MATLAB function $minimizeDFA(S, a, M, s1, s2)$;
Step7: See the output in MATLAB command window.

Program

```
function [minimizedDFA, minimizedStates] =
minimizeDFA(states, alphabet, transition, initialState, finalStates)
% states: Cell array of state names (e.g., {'q0', 'q1', 'q2'})
% alphabet: Cell array of alphabet symbols (e.g., {'0', '1'})
% transition: Matrix representing transitions,
where transition(i, j) is the next state from state i on input symbol j.
% initialState: The index of the initial state.
% finalStates: Indices of final (accepting) states.

% Output:
% minimizedDFA: The transition table of the minimized DFA.
% minimizedStates: The new state names of the minimized DFA.

numStates = length(states);
numSymbols = length(alphabet);

% Step 1: Initialize the partition
% Partition the states into accepting and non-accepting sets.
```

```
acceptingStates = finalStates;
nonAcceptingStates = setdiff(1:numStates, acceptingStates);
partition = {acceptingStates, nonAcceptingStates};

% Step 2: Refining the partition
change = true;
while change
change = false;
newPartition = {};

% Go through each group in the current partition
for i = 1:length(partition)
group = partition{i};
% Group states by their transitions on each input symbol
transitionGroups = containers.Map;

for state = group
% Get the transitions for this state
transitions = zeros(1, numSymbols);
for j = 1:numSymbols
nextState = transition(state, j); % Find the next state for this input
transitions(j) = find(nextState == [partition{:}]); % Find
the group of the next state
end

% Use the transition tuple as a key to group states
key = mat2str(transitions); % Use the transition vector as a key
if isKey(transitionGroups, key)
transitionGroups(key) = [transitionGroups(key), state];
else
transitionGroups(key) = state;
end
end

% Add new groups formed by transitions
newPartition = [newPartition, values(transitionGroups)];
end

% If the partition has changed, update it
if length(newPartition) ~= length(partition)
partition = newPartition;
change = true;
end
end
```

```
% Step 3: Assign new state names to each partition
minimizedStates = cell(1, length(partition));
for i = 1:length(partition)
minimizedStates{i} = ['q' num2str(i)];
end

% Step 4: Construct the minimized transition table
minimizedDFA = zeros(length(partition), numSymbols);
for i = 1:length(partition)
for j = 1:numSymbols
% Find the next state of each state in the group under the given input
nextState = transition(partition{i}(1), j); % Take
the first state of the group
for k = 1:length(partition)
if ismember(nextState, partition{k})
minimizedDFA(i, j) = k; % Assign the corresponding minimized state
break;
end
end
end
end

% Step 5: Determine the new initial state and final states
minimizedInitialState = find(cellfun(@(x)
ismember(initialState, x), partition));
minimizedFinalStates = find(cellfun(@(x)
any(ismember(x, finalStates)), partition));

% Display the minimized DFA
disp('Minimized DFA States:');
disp(minimizedStates);
disp('Minimized DFA Transition Table:');
disp(minimizedDFA);
disp(['Minimized Initial State: q', num2str(minimizedInitialState)]);
disp(['Minimized Final States: q', num2str(minimizedFinalStates)]);
end
```

Input

```
states = {'q0', 'q1', 'q2', 'q3'};  % States
alphabet = {'0', '1'};  % Alphabet
transition = [
2, 1;   % From q0, on 0 go to q1, on 1 go to q2
1, 2;   % From q1, on 0 go to q1, on 1 go to q2
4, 3;   % From q2, on 0 go to q4, on 1 go to q3
4, 3;   % From q3, on 0 go to q4, on 1 go to q3
];  % Transition table
initialState = 1;  % Initial state q0
finalStates = [2];  % Final state q1

[minimizedDFA, minimizedStates] =
 minimizeDFA(states, alphabet, transition, initialState, finalStates);
```

Output

```
Minimized DFA States:
'q1'
'q2'

Minimized DFA Transition Table:
2     1
2     1

Minimized Initial State: q1
Minimized Final States: q2
```

Exercise Problem

Q1. Write a similar MATLAB code for Kleen's theorem.
Q2. Give the input statistics of DFA of your choice and verify output of defined MATLAB function.

Chapter 8

Probability Theory

Practical No. 1

Aim

To solve a given problem of Conditional theorem with the help of MATLAB.

Problem

In a bag of 5 red and 3 blue marbles, one marble is drawn at random and not replaced. Then, a second marble is drawn. What is the probability that the second marble drawn is blue, given that the first marble drawn was red?

Theory

Let:

- R1 be the event that the first marble drawn is red.
- B2 be the event that the second marble drawn is blue.

We want to find $\mathbf{P}(B2 \mid R1)$, the probability of drawing a blue marble second given that a red marble was drawn first.

- **step 1:** Calculate the probability of drawing a red marble first ($\mathbf{P}$(R1)):
  There are 5 red and 3 blue marbles, for a total of 8 marbles.
  P(R1) = Number of Red Marbles / Total Marbles = 5/8.
- **step 2:** Calculate the probability of drawing a blue marble second given that a red was drawn first ($\mathbf{P}(B2 \mid R1)$): After drawing one red marble, we have 4 red and 3 blue marbles left, totaling 7 marbles.
  $\mathbf{P}(B2 \mid R1)$= Remaining Blue Marbles / Remaining Total Marbles = 3/7.

So, the final answer is $\mathbf{P}(B2 \mid R1) = 3/7 \approx 0.429$.

Algorithm

Step 1: Define total marbles as total_marbles = 8.
Step 2: Define number of red and blue marbles as red_marbles = 5; and blue_marbles = 3.
Step 3: Calculate Probability of drawing a red marble first as P_R1 = red_marbles / total_marbles.
Step 4: Update number of marbles after drawing one red marble as remaining_marbles = total_marbles 1 and remaining_blue_marbles = blue_marbles.
Step 5: Calculate Probability of drawing a blue marble second, given first was red P_B2_given_R1 = remaining_blue_marbles / remaining_marbles.
Step 6: Display the rsesult as fprintf('The probability of drawing a blue marble second given that the first was red is: %.3f $\backslash n'$, P_B2_given_R1).

Program

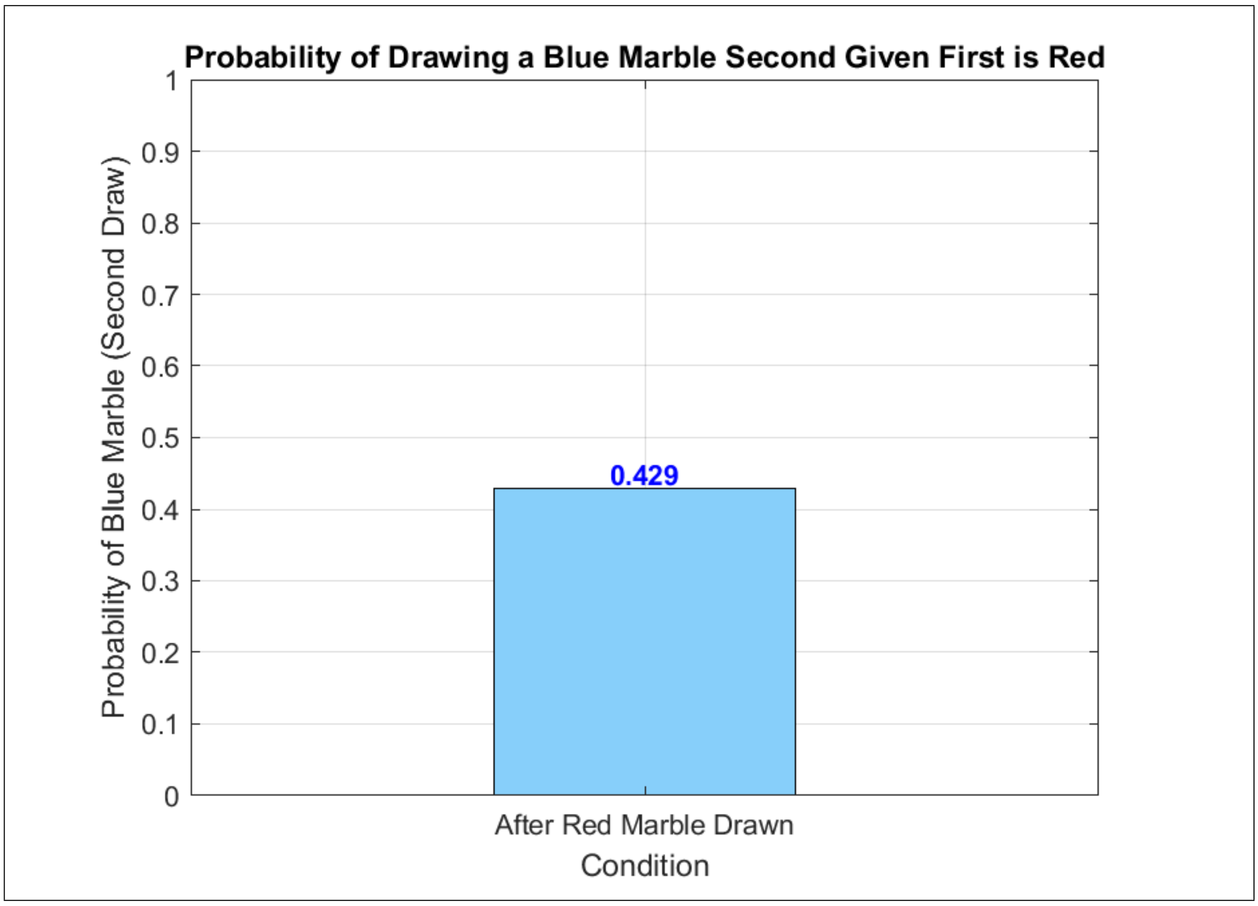
```
% Total marbles
total_marbles = 8;

% Number of red and blue marbles
red_marbles = 5;
blue_marbles = 3;

% Probability of drawing a red marble first
P_R1 = red_marbles / total_marbles;

% Update number of marbles after drawing one red marble
remaining_marbles = total_marbles - 1;
remaining_blue_marbles = blue_marbles;

% Probability of drawing a blue marble second, given first was red
P_B2_given_R1 = remaining_blue_marbles / remaining_marbles;

% Display the outcome
fprintf('The probability of drawing a blue marble second given that the first was red is: %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Define the condition and probability
condition = {'After Red Marble Drawn'};
probability = 3/7;  % Probability of blue on the second draw given the first was
red

% Create a bar chart
figure;
```

```
bar(probability, 'FaceColor', [0.53, 0.81, 0.98]);  % Light blue color
set(gca, 'xticklabel', condition);

% Label the chart
title("Probability of Drawing a Blue Marble Second Given First is Red");
xlabel('Condition');
ylabel('Probability of Blue Marble (Second Draw)');
ylim([0 1]);

% Annotate the bar with the probability value
text(1, probability + 0.02, num2str(probability, '%.3f'), ...
    'HorizontalAlignment', 'center', 'Color', 'blue', 'FontWeight', 'bold');
```

Output

The output of the program for various inputs is as following:

```
The probability of drawing a blue marble second given that the first was red is:
0.429
```



General Matlab program

```
% Ask the user for input values
total_marbles = input('Enter the total number of marbles: ');
% Total number of marbles
red_marbles = input('Enter the number of red marbles: ');
% Number of red marbles
blue_marbles = input('Enter the number of blue marbles: ');
% Number of blue marbles

% Check if the inputs are valid
if red_marbles + blue_marbles > total_marbles
    disp('Error: The sum of red and blue marbles cannot exceed the total number of
    marbles.');
    return;
elseif red_marbles < 0 || blue_marbles < 0 || total_marbles <= 0
    disp('Error: Invalid input. Please enter positive values.');
    return;
end

% Probability of drawing a red marble first
P_R1 = red_marbles / total_marbles;

% Update the number of marbles after drawing one red marble
remaining_marbles = total_marbles - 1;
remaining_blue_marbles = blue_marbles;

% Probability of drawing a blue marble second, given the first was red
P_B2_given_R1 = remaining_blue_marbles / remaining_marbles;

% Display the calculated probability in the command window
fprintf('The probability of drawing a blue marble second given that the first was red
is: %.3f\n', P_B2_given_R1);

% Condition description and its probability for plotting
condition = {'After Red Marble Drawn'};
% Text for the condition
probability = P_B2_given_R1;
% Probability of drawing blue second after red first

% Create a bar chart to visualize the probability
figure;
bar(probability, 'FaceColor', [0.53, 0.81, 0.98]);  % Light blue color
set(gca, 'xticklabel', condition);  % Set the x-axis label to the condition
```

```
% Label the chart
title("Probability of Drawing a Blue Marble Second Given First is Red");
xlabel('Condition');
ylabel('Probability of Blue Marble (Second Draw)');
ylim([0 1]);  % Set y-axis range from 0 to 1

% Annotate the bar with the probability value
text(1, probability + 0.02, num2str(probability, '%.3f'), ...
    'HorizontalAlignment', 'center', 'Color', 'blue', 'FontWeight', 'bold');

% Display the chart with a grid
grid on;
```

Exercise Problem

A deck of 52 playing cards is shuffled, and one card is drawn at random. Without replacing it, a second card is drawn. What is the probability that the second card is an ace, given that the first card drawn was a king?

Practical No. 2

Aim

To solve a given problem of Independent events with the help of MATLAB.

Problem

In a factory, the probability that a randomly selected product is defective is P(D) = 0.02. The probability that it was made on Machine A is P(A) = 0.3. Assume that being defective is independent of which machine the product is made on.

1. Calculate the probability that a product is defective and made on Machine A, P(A ∩ D).
2. Calculate the probability that a product is not defective and made on Machine A.

Theory

- **Step 1:** Interpret the Given Data
  We are given $\mathbf{P}(\mathrm{D}) = 0.02$, the probability that a product is defective.
  Since 'defective' and 'made on Machine A' are independent events, we can use the rule for independent events:
  $\mathbf{P}(\mathrm{A} \cap \mathrm{D}) = \mathbf{P}(\mathrm{A}) * \mathbf{P}(\mathrm{D})$

- **Step 2:** Calculate **P**(A ∩ D)
  Using the independence formula:
  P(A ∩ D) = **P**(A) * **P**(D)= 0.3 * 0.02 = 0.006
  So, the probability that a product is defective and made on Machine A is 0.006.
- **Step 3:** Calculate **P**(not D ∩ A)
  First, find **P**(not D). Since the probability of a product being defective is **P**(D) = 0.02, the probability that it is not defective is:
  P(not D) = 1 - **P**(D) = 1 - 0.02 = 0.98

Now, we use the fact that 'not defective' and 'made on Machine A' are also independent events:
P(not D ∩ A) = **P**(not D) * **P**(A) = 0.98 * 0.3 = 0.294
Thus, the probability that a product is not defective and made on Machine A is 0.294.

Algorithm

Step 1: Define probability as P_ D = 0.02: Probability of a product being defective and P_ A = 0.3: Probability of a product being made on Machine A.
Step 2: Calculate combined probabilities: P_ A_ and P_ D: Probability that a product is defective and made on Machine A, calculated as P_ A * P_ D.
Step 3: Calculate P_ not_ D = 1 - P_ D: Probability that a product is not defective.
Step 4: Calculate P_ not_D_ and_ A: Probability that a product is not defective and made on Machine A, calculated as P_ not_ D * P_ A.
Step 5: Display results: Print both probabilities using fprintf.

Program

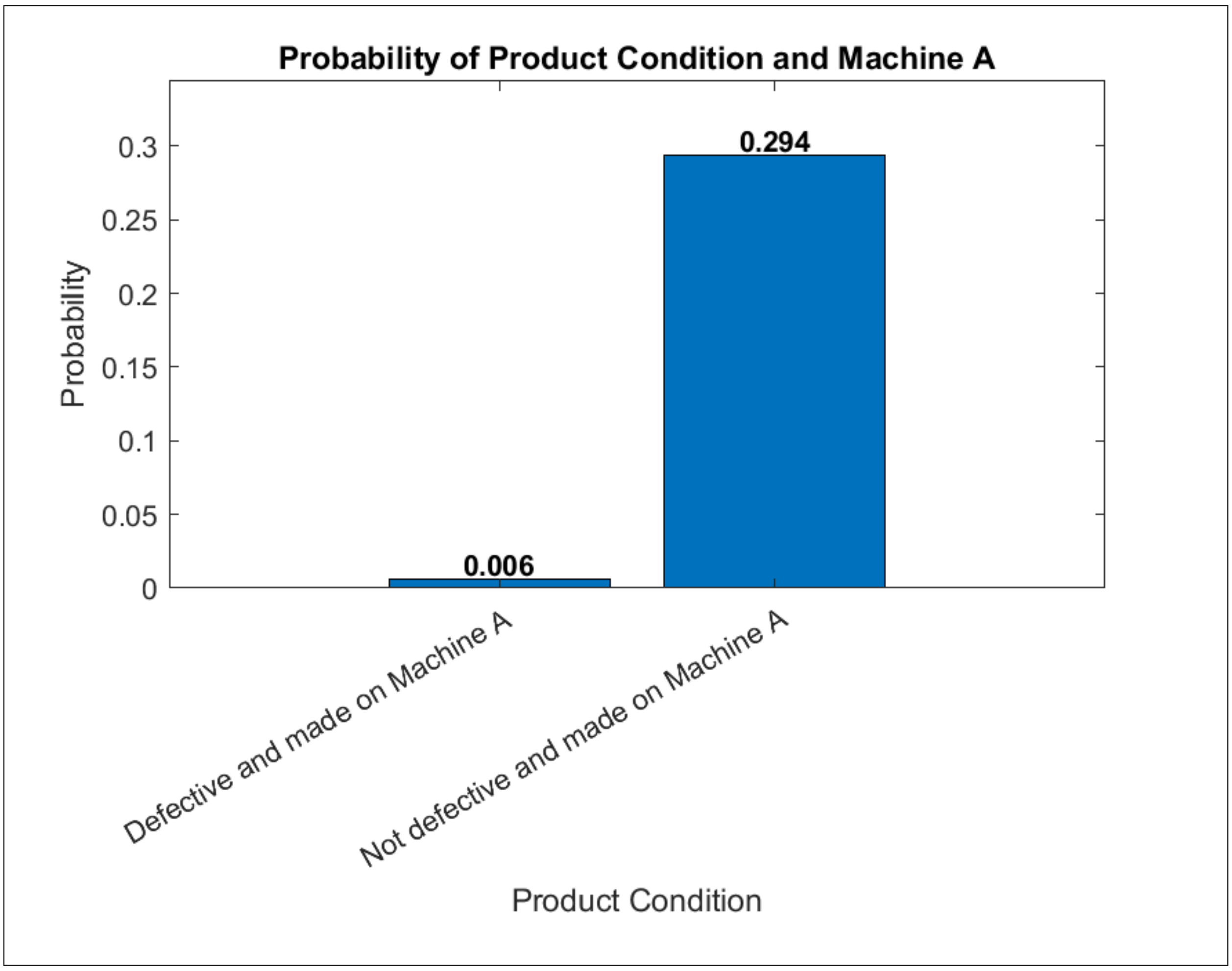
Below is the MATLAB code to calculate these probabilities.

```
% Given probabilities
P_D = 0.02;         % Probability that a product is defective
P_A = 0.3;          % Probability that a product is made on Machine A

% 1. Probability that a product is defective and made on Machine A
P_A_and_D = P_A * P_D;

% 2. Probability that a product is not defective and made on Machine A
P_not_D = 1 - P_D;        % Probability that a product is not defective
P_not_D_and_A = P_not_D * P_A;

% Display the results
fprintf('The probability that a product is defective and made on Machine A is:
%.3f\n', P_A_and_D);
fprintf('The probability that a product is not defective and made on Machine A is:
%.3f\n', P_not_D_and_A);
```

Output

The output of the program for various inputs is as following:

```
The probability that a product is defective and made on Machine A is: 0.006
The probability that a product is not defective and made on Machine A is: 0.294
```



General Matlab program

```
% Given probabilities
P_D = 0.02;        % Probability that a product is defective
P_A = 0.3;         % Probability that a product is made on Machine A

% 1. Probability that a product is defective and made on Machine A
P_A_and_D = P_A * P_D;

% 2. Probability that a product is not defective and made on Machine A
P_not_D = 1 - P_D;        % Probability that a product is not defective
P_not_D_and_A = P_not_D * P_A;

% Display the results
fprintf('The probability that a product is defective and made on Machine A is:
%.3f\n', P_A_and_D);
fprintf('The probability that a product is not defective and made on Machine A is:
```

```
%.3f\n', P_not_D_and_A);
% Define probabilities
p_A = 0.3;         % Probability of being made on Machine A
p_D = 0.02;        % Probability of being defective
p_not_D = 1 - p_D; % Probability of not being defective

% Calculations for combined probabilities
p_A_and_D = p_A * p_D;
p_A_and_not_D = p_A * p_not_D;

% Set up labels and values for visualization
categories = {'Defective and made on Machine A', 'Not defective and made on Machi
A'};
values = [p_A_and_D, p_A_and_not_D];

% Create bar chart
figure;
bar(values, 'FaceColor', 'flat');
title('Probability of Product Condition and Machine A');
xlabel('Product Condition');
ylabel('Probability');
xticklabels(categories);
ylim([0, max(values) + 0.05]); % Add some padding above bars

% Set colors for each bar
colormap([1 0 0; 0 1 0]); % Red for defective, Green for not defective

% Display probabilities on top of each bar
for i = 1:length(values)
    text(i, values(i) + 0.01, sprintf('%.3f', values(i)), ...
        'HorizontalAlignment', 'center', 'FontWeight', 'bold');
end
```

Exercise Problem

In a warehouse, the probability that a product is fragile is P(F)=0.15, and the probability that it is stored in the top row is P(T)=0.4. Assume that being fragile is independent of the storage location.

1. Calculate the probability that a product is both fragile and stored in the top row.
2. Calculate the probability that a product is not fragile and is stored in the top row.

Practical No. 3

Aim

To solve a given problem of Bayes theorem with the help of MATLAB.

Problem

There are two bags, Bag A and Bag B. Bag A contains 4 red balls and 6 blue balls, while Bag B contains 7 red balls and 3 blue balls. A bag is selected at random, and a ball is drawn from it. If the ball drawn is red, what is the probability that it came from Bag B?

Theory

- **Step 1:** Determine Initial Probabilities for Selecting Each Bag:
  Since a bag is chosen at random, we have:
  P(A) = 0.5
  P(B) = 0.5
- **Step 2:** Calculate the Probability of Drawing a Red Ball from Each Bag:
  For Bag A: **P**(R | A) = (Number of Red Balls in Bag A) / (Total Balls in Bag A)
  = 4/10 = 0.4
  For Bag B: **P**(R | B) = (Number of Red Balls in Bag B) / (Total Balls in Bag B)
  = 7/10 = 0.7
- **Step 3:** Calculate the Total Probability of Drawing a Red Ball (**P**(R)): Using the law of total probability:
  P(R) = **P**(R | A) * **P**(A) + **P**(R | B) * **P**(B)
  Substituting the values:
  P(R) = (0.4 * 0.5) + (0.7 * 0.5) = 0.55
- **Step 4:** Apply Baye's Theorem to Find **P**(B — R):
  Using Baye's theorem:
  P(B | R) = (**P**(R | B) * **P**(B)) / **P**(R)
  Substituting the values:
  P(B | R) = (0.7 * 0.5) / 0.55 ≈ 0.636

So, the probability that the red ball came from Bag B is approximately 0.636.

Algorithm

Step 1: Set initial probabilities as P_ A = 0.5 and P_ B = 0.5.
Step 2: Set conditional probabilities as P_R_ given_ A = 0.4 (from Bag A) and P_R_ given_ B = 0.7 (from Bag B).
Step 3: Calculate total probability of drawing a red ball as P_ R = (P_ R_ given_ A * P_

A) + (P_ R_ given_ B * P_ B).
Step 4: Apply Bayes theorem to find probability of Bag B given a red ball P_ B_ given_ R = (P_ R_ given_ B * P_ B) / P_ R.
Step 5: Display the result using fprintf.

Program

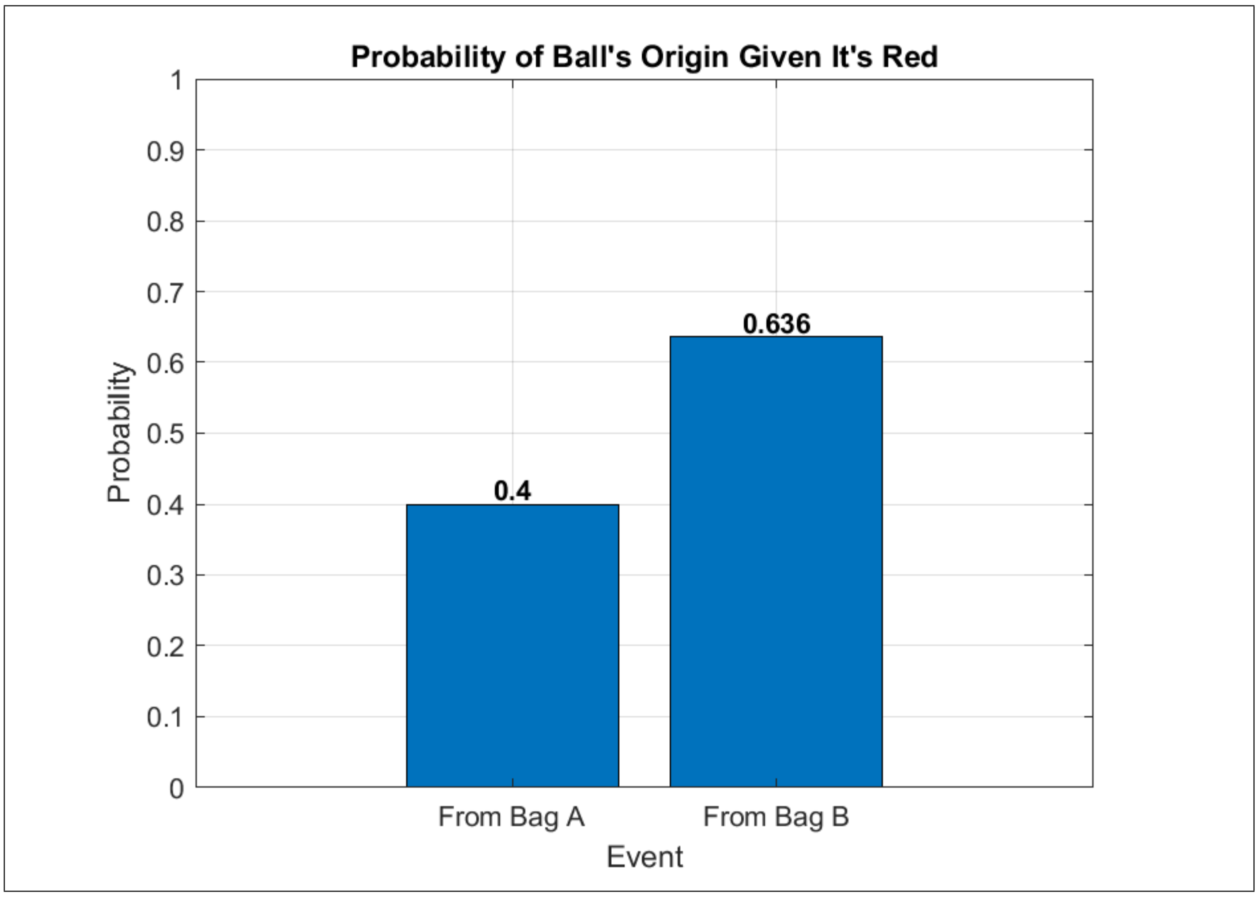
```
% Initial probabilities of selecting each bag
P_A = 0.5;
P_B = 0.5;

% Probability of drawing a red ball from each bag
P_R_given_A = 4 / 10;  % Bag A: 4 red balls out of 10
P_R_given_B = 7 / 10;  % Bag B: 7 red balls out of 10

% Total probability of drawing a red ball
P_R = (P_R_given_A * P_A) + (P_R_given_B * P_B);

% Applying Baye's theorem to find P(B | R)
P_B_given_R = (P_R_given_B * P_B) / P_R;

% Display the result
fprintf('The probability that the red ball came from Bag B is: %.3f\n', P_B_given_R);
```

Output

```
 The probability that the red ball came from Bag B is: 0.636
```

General Matlab Program

```
% Initial probabilities of selecting each bag
P_A = 0.5;
P_B = 0.5;

% Probability of drawing a red ball from each bag
P_R_given_A = 4 / 10;  % Bag A: 4 red balls out of 10
P_R_given_B = 7 / 10;  % Bag B: 7 red balls out of 10

% Total probability of drawing a red ball
P_R = (P_R_given_A * P_A) + (P_R_given_B * P_B);

% Applying Baye's theorem to find P(B | R)
P_B_given_R = (P_R_given_B * P_B) / P_R;

% Display the result
fprintf('The probability that the red ball came from Bag B is: %.3f\n', P_B_given_R);

% Define events and their probabilities
```



```
events = {‘From Bag A’, ‘From Bag B’};
probabilities = [0.4, 0.636];

% Create a bar chart
figure;
bar(probabilities, ‘FaceColor’, ‘flat’);
set(gca, ‘xticklabel’, events);

% Set colors for each bar
colors = [0.53, 0.81, 0.98; 1, 0.55, 0.41];  % Light blue and salmon colors
for i = 1:length(probabilities)
    b.CData(i,:) = colors(i,:);
end

% Label the chart
title("Probability of Ball’s Origin Given It’s Red");
xlabel(‘Event’);
ylabel(‘Probability’);
ylim([0 1]);
```

```
% Annotate the bars with probability values
text(1:length(probabilities), probabilities + 0.02, ...
    string(probabilities), 'HorizontalAlignment', 'center', ...
    'Color', 'black', 'FontWeight', 'bold');

% Display the chart
grid on;
```

Exercise Problem

There are two bags, Bag C and Bag D. Bag C contains 5 green balls and 5 yellow balls, while Bag D contains 3 green balls and 7 yellow balls. A bag is selected at random, and a ball is drawn from it. If the ball drawn is green, what is the probability that it came from Bag D?

Practical No. 4

Aim

To solve a given problem of probability mass function with the help of MATLAB.

Problem

A bag contains 3 red balls and 2 green balls. Two balls are drawn at random without replacement, and let Y represent the number of red balls drawn.

1. Write down the probability mass function (PMF) of Y.
2. Calculate the expected value (mean) of Y.

Theory

- **Step 1: Define the Probability Mass Function (PMF)**
  The possible values of Y are 0, 1, or 2 (i.e., drawing 0, 1, or 2 red balls). We need to calculate $\mathbf{P}(Y = 0)$, $\mathbf{P}(Y = 1)$, and $\mathbf{P}(Y = 2)$.

- **Step 2: Total Ways to Draw 2 Balls**
  The total number of ways to draw 2 balls from 5 (3 red + 2 green) is given by:
  Total Ways = C(5, 2) = 5! / (2!(5-2)!) = 10

- **Step 3: Calculate Each Probability**
  For $\mathbf{P}(Y = 0)$: This means both balls drawn are green.
  $\mathbf{P}(Y = 0) = C(2, 2) / C(5, 2) = 1 / 10$

For **P**(Y = 1): This means one red and one green ball is drawn.
P(Y = 1) = (C(3, 1) * C(2, 1)) / C(5, 2) = (3 * 2) / 10 = 6 / 10 = 3 / 5

For **P**(Y = 2): This means both balls drawn are red.
P(Y = 2) = C(3, 2) / C(5, 2) = 3 / 10

- **Step 4: Summary of PMF**
  P(Y = 0) = 1/10
  P(Y = 1) = 3/5
  P(Y = 2) = 3/10
- **Step 5: Calculate the Expected Value (Mean) of Y**
  E(Y) = 0 * **P**(Y = 0) + 1 * **P**(Y = 1) + 2 * **P**(Y = 2)
  E(Y) = 0 + 0.6 + 0.6 = 1.2

Algorithm

Step 1: Define ball counts: Set num_ red = 3 (number of red balls) and num_ green = 2 (number of green balls). Calculate total_ balls = num_ red + num_ green.
Step 2: Calculate total selection ways: Use total_ ways = nchoosek(total_ balls, 2) to find the number of ways to choose 2 balls from the total.
Step 3: Calculate probability of drawing 0 red balls: Compute P_ Y_ 0 = nchoosek(num_ green, 2) / total_ ways (2 green balls).
Step 4: Calculate probability of drawing 1 red ball: Compute P_ Y_ 1 = (nchoosek(num_ red, 1) * nchoosek(num_ green, 1)) / total_ ways (1 red and 1 green ball).
Step 5: Calculate probability of drawing 2 red balls: Compute P_ Y_ 2 = nchoosek(num_ red, 2) / total_ ways (2 red balls).
Step 6: Calculate expected value of Y: Use E_ Y = 0 * P_ Y_ 0 + 1 * P_ Y_ 1 + 2 * P_ Y_ 2 to find the expected value.
Step 7: Display results: Print P_ Y_ 0, P_ Y_ 1, P_ Y_ 2, and E_ Y with fprintf.

Program

```
% Define the number of red and green balls
num_red = 3;
num_green = 2;
total_balls = num_red + num_green;

% Total ways to choose 2 balls from 5
total_ways = nchoosek(total_balls, 2);

% Calculate probabilities
P_Y_0 = nchoosek(num_green, 2) / total_ways; % P(Y = 0)
P_Y_1 = (nchoosek(num_red, 1) * nchoosek(num_green, 1)) / total_ways; % P(Y = 1)
```

```
P_Y_2 = nchoosek(num_red, 2) / total_ways; % P(Y = 2)

% Expected value E(Y)
E_Y = 0 * P_Y_0 + 1 * P_Y_1 + 2 * P_Y_2;


% Display results
fprintf('Probability Mass Function (PMF):\n');
fprintf('P(Y = 0) = %.2f\n', P_Y_0);
fprintf('P(Y = 1) = %.2f\n', P_Y_1);
fprintf('P(Y = 2) = %.2f\n', P_Y_2);
fprintf('Expected Value E(Y) = %.2f\n', E_Y);
```

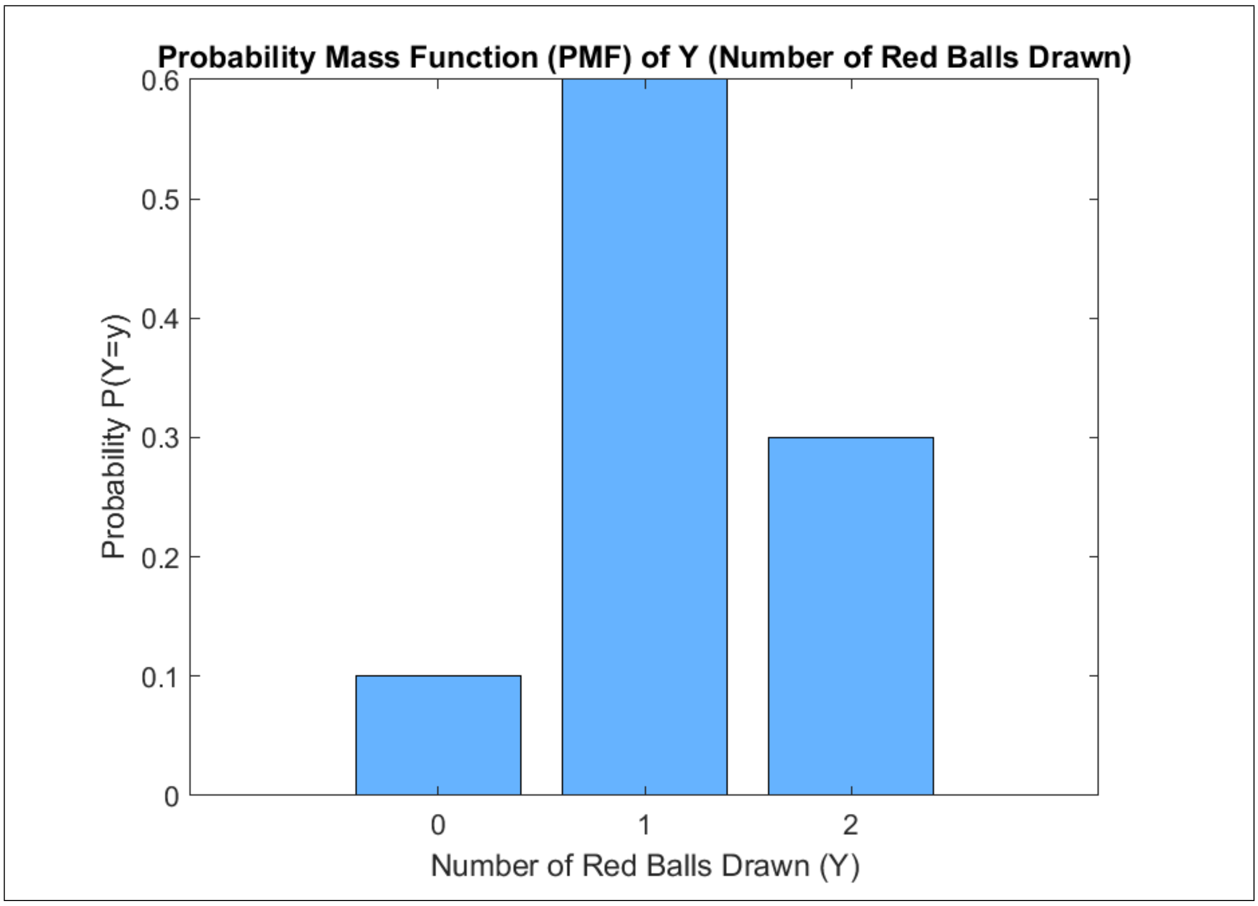
Output

The output of the program for various inputs is as following:

```
Probability Mass Function (PMF):
P(Y = 0) = 0.10
P(Y = 1) = 0.60
P(Y = 2) = 0.30
Expected Value E(Y) = 1.20
```

General Matlab Program

```
% Define the number of red and green balls
num_red = 3;
num_green = 2;
total_balls = num_red + num_green;

% Total ways to choose 2 balls from 5
total_ways = nchoosek(total_balls, 2);

% Calculate probabilities
P_Y_0 = nchoosek(num_green, 2) / total_ways; % P(Y = 0)
P_Y_1 = (nchoosek(num_red, 1) * nchoosek(num_green, 1)) / total_ways; % P(Y = 1)
P_Y_2 = nchoosek(num_red, 2) / total_ways; % P(Y = 2)

% Expected value E(Y)
E_Y = 0 * P_Y_0 + 1 * P_Y_1 + 2 * P_Y_2;


% Display results
fprintf('Probability Mass Function (PMF):\n');
```



```
fprintf('P(Y = 0) = %.2f\n', P_Y_0);
fprintf('P(Y = 1) = %.2f\n', P_Y_1);
fprintf('P(Y = 2) = %.2f\n', P_Y_2);
fprintf('Expected Value E(Y) = %.2f\n', E_Y);
% Define values and probabilities for Y (number of red balls drawn)
y_values = [0, 1, 2];              % Possible values of Y
probabilities = [0.1, 0.6, 0.3];  % Corresponding probabilities P(Y=y)

% Calculate the expected value for Y
expected_value = sum(y_values .* probabilities);
```

Exercise problem

A discrete random variable X represents the number of times a coin lands on heads in 3 tosses of a fair coin. Let X take values from 0 to 3, representing the possible outcomes.

1. Write down the probability mass function (PMF) of X.
2. Calculate the expected value (mean) of X.

Practical No. 5

Aim

To solve a given problem of probability density function with the help of MATLAB.

Problem

Given that X has a probability density function:

$$f(x) = \begin{cases} 3x^2, & 0 \leq x \leq 1 \\ 0, & \text{otherwise} \end{cases}$$

1. Find the cumulative distribution function (CDF) F(x) of X.
2. Use F(x) to find $\mathbf{P}(0.2 \leq X \leq 0.6)$.

Theory

- **Step 1: Finding the CDF F(x)**
  To find the CDF, integrate the PDF from 0 to x:

  $$F(x) = \int_0^x 3t^2 dt = x^3$$

  Thus, the CDF F(x) is:

  $$F(x) = \begin{cases} 0, & x < 0 \\ x^3, & 0 \leq x \leq 1 \\ 1, & x > 1 \end{cases}$$

- **Step 2:** Calculating $\mathbf{P}(0.2 \leq X \leq 0.6)$

  Using the CDF F(x), we can calculate $\mathbf{P}(0.2 \leq X \leq 0.6)$ as:
  $\mathbf{P}(0.2 \leq X \leq 0.6)$=F(0.6) - F(0.2)= 0.216 - 0.008 = 0.208.
  So, $\mathbf{P}(0.2 \leq X \leq 0.6)$=0.208.

Algorithm

Step 1: Define the Probability density function (PDF): f = @(x) 3 * `x^2`.
Step 2: Define the Cumulative distribution function (CDF): F = @(x) `x^3`$*(x >= 0\&x <= 1) + (x > 1)$
Step 3: Compute the CDF values at specific points: F_ 0_ 6 = F (0.6) and F_ 0_ 2 = F (0.2).

Step 4: Calculate the probability between two values: P = F_ 0_ 6 - F_ 0_ 2.
Step 5: Display the results: Print the values of F(0.6), F(0.2), and the probability P(0.2 \leqX\leq 0.6) using fprintf.

Program

```
% Define the PDF as an anonymous function
f = @(x) 3 * x.^2;

% Define the CDF by integrating the PDF from 0 to x
F = @(x) x.^3 .* (x >= 0 & x <= 1) + (x > 1);

% Compute F(0.6) and F(0.2)
F_0_6 = F(0.6);
F_0_2 = F(0.2);

% Calculate P(0.2 <= X <= 0.6)
P = F_0_6 - F_0_2;

% Display the results
fprintf('F(0.6) = %.3f\n', F_0_6);
fprintf('F(0.2) = %.3f\n', F_0_2);
fprintf('P(0.2 <= X <= 0.6) = %.3f\n', P);
```

Output

The output of the program for various inputs is as following:

```
F(0.6) = 0.216
F(0.2) = 0.008
P(0.2 <= X <= 0.6) = 0.208
```

Exercise Problem

Let f(x)=$\begin{cases} 2x & 0 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}$
1. Verify that f(x) is a valid probability density function.
2. Find the probability that X lies between 0.5 and 1, i.e., P($0.5 \leq X \leq 1$).

Practical No. 6

Aim

To solve a given problem of moment generating function with the help of MATLAB.

Problem

Given that X has a probability density function:

$$\begin{cases} 2x & 0 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

1. Derive the moment generating function $M_X(t) = E[e^{tX}]$.
2. Use $M_X(t)$ to find the first moment of X.

Theory

- **Step 1: Finding the Moment Generating Function** $M_X(t)$
  To find $M_X(t)$, we need to compute the expected value $E[e^{tX}]$., which is defined as:
  $$M_X(t) = E[e^{tX}] = \int_{-\infty}^{\infty} e^{tx} f(x)dx = \int_0^1 e^{tx}(2x)dx = \frac{2(-e^t + te^t + 1)}{t^2}$$
- **Step 2: : Finding E[X] Using** $M_X(t)$
  To find the moments, we need the derivatives of $M_X(t)$ at t = 0.
  Calculate E[X] = $M_X'(t)$=0.667.

Algorithm

Step 1: Declare symbolic variable: Use syms to declare t as a symbolic variable.
Step 2: Define the Moment-Generating Function (MGF): Define the MGF function M_X(t) based on the given formula using t.
Step 3: Differentiate MGF: Take the first derivative of M_X with respect to t and assign it to M_X_prime.
Step 4: Calculate expected value: Use the limit function to evaluate the derivative at t = 0, which represents E[X], and store this in E_X.
Step 5: Display the expected value: Convert E_X to a numeric format using ‘double‘ and print it with ‘fprintf‘.

Program

```
syms t
```

```
% Define the MGF based on derived formula
M_X = (2 * (-exp(t) + t * exp(t) + 1)) / t^2;

% First derivative of M_X with respect to t
M_X_prime = diff(M_X, t);

% Calculate E[X] = M_X'(0)
E_X = limit(M_X_prime, t, 0);

% Display the results
fprintf('E[X] = %.3f\n', double(E_X));
```

Output

The output of the program for various inputs is as following:

```
E[X] = 0.667
```

Exercise problem

Let X be a continuous random variable with the moment generating function:

$$M_X(t) = \frac{1}{(1-2t)^3} for \quad t < \frac{1}{2}$$

1.Use $M_X(t)$ to find E[X]
2.Compute Var(X).

Practical No. 7

Aim

To solve a given problem of Bernoulli random variable with the help of MATLAB.

Problem

Let X be a Bernoulli random variable with probability of success p = 0.3.
1. Write the probability mass function (PMF) of X.
2. Compute E[X] and Var(X).
3. Suppose we perform 5 independent Bernoulli trials with the same p. Calculate the probability of observing exactly 3 successes.

Theory

- **Step 1: Probability Mass Function (PMF)**
  For a Bernoulli random variable X with success probability p = 0.3:
  $$P(X = x) = \begin{cases} p = 0.3, & \texttt{if } x = 1 \\ 1 - p = 0.7, & \texttt{if } x = 0 \end{cases}$$
  Thus, **P**(X = 1) = 0.3 and **P**(X = 0) = 0.7.

- **Step 2: Expectation and Variance**
  The expectation E[X] of a Bernoulli random variable is: E[X] = p=0.3.
  The variance Var(X) of a Bernoulli random variable is: Var(X) = p(1 - p)= 0.21

- **Step 3: Probability of Exactly 3 Successes in 5 Trials**
  Let Y be the number of successes in 5 independent Bernoulli trials with p = 0.3. Then Y follows a Binomial distribution: Y ≈ Binomial(n = 5, p = 0.3).
  The probability of observing exactly 3 successes (i.e.,**P**(Y = 3)) is given by the binomial formula:
  P(Y = k) = C(n, k) * `p^k` * `(1-p)^(n-k)`= C(5, 3) * `0.3^3` * `(1-0.3)^(5-3)`=0.1631

Algorithm

Step 1: Define Probability of Success (‘p’): Set the probability of a single success, ‘p = 0.3’.
Step 2: Calculate Expectation and Variance: Compute the expectation, ‘E_X = p’ and the variance, ‘Var_X = p * (1 - p)’, for a single Bernoulli trial.
Step 3: Display Expectation and Variance: Print ‘E[X]’ and ‘Var[X]’ to the console using ‘fprintf’.
Step 4: Define Parameters for Binomial Probability: Set the number of trials (‘n = 5’) and the desired number of successes (‘k = 3’).
Step 5: Calculate Binomial Probability ‘P (Y = 3)’ and use ‘nchoosek’ to calculate the binomial coefficient.
Step 6: Display Binomial Probability: Print ‘P(Y = 3)’ to the console using ‘fprintf’.

Program

```
% Given probability of success
p = 0.3;

% Step 2: Calculate expectation and variance
E_X = p;
```

```
Var_X = p * (1 - p);

% Display results for expectation and variance
fprintf('E[X] = %.2f\n', E_X);
fprintf('Var[X] = %.2f\n', Var_X);

% Step 3: Calculate probability of exactly 3 successes in 5 trials
n = 5;    % number of trials
k = 3;    % desired number of successes

% Calculate binomial probability P(Y = 3)
P_Y_eq_3 = nchoosek(n, k) * p^k * (1 - p)^(n - k);

% Display result for P(Y = 3)
fprintf('P(Y = 3) = %.4f\n', P_Y_eq_3);
```

Output

The output of the program for various inputs is as following:

```
E[X] = 0.30
Var[X] = 0.21
P(Y = 3) = 0.1323
```

Exercise Problem

Consider a Bernoulli random variable Z with a success probability of p=0.2.
1.Define the probability mass function (PMF) for Z.
2.Calculate the expected value E[Z] and the variance Var(Z).
3.If you conduct 10 independent Bernoulli trials with this probability, find the probability of observing at least 3 successes.

Practical No. 8

Aim

To solve a given problem of Binomial random variable with the help of MATLAB.

Problem

A factory produces light bulbs, and 5% of them are defective. A quality control inspector selects 20 bulbs at random.

1. What is the probability that exactly 2 bulbs are defective?
2. What is the probability that at most 1 bulb is defective?

Theory

- **Step 1: Given information**
  Probability of a defective bulb, p = 0.05
  Number of trials (bulbs selected), n = 20

  Let X be the number of defective bulbs among the 20 chosen, which follows a binomial distribution:
  X ≈ Binomial(n = 20, p = 0.05)

- **Step 2: Probability that exactly 2 bulbs are defective**
  The probability of exactly k defective bulbs out of n trials is given by the binomial probability formula:
  P(X = k) = C(n, k) `* p^k * (1-p)^(n-k)`

  For k = 2:
  P(X = 2) = C(20, 2) `* (0.05)^2 * (0.95)^18` =0.1887

- **Step 3: Probability that at most 1 bulb is defective**
  To find the probability that at most 1 bulb is defective, we need to sum the probabilities of X = 0 and X = 1:
  P(X <= 1) = P(X = 0) + P(X = 1)= 0.7358

Algorithm

Step 1: Define Parameters: Set the total number of bulbs selected ('n = 20') and the probability of a bulb being defective ('p = 0.05').
Step 2: Calculate Probability of Exactly 2 Defective Bulbs: Use 'binopdf(k, n, p)' to calculate the probability that exactly 2 bulbs are defective ('k = 2').
Step 3: Calculate Probability That at Most 1 Bulb Is Defective: Use 'binopdf' to find the probability of having 0 defective bulbs ('P_X_equals_0'). Similarly, use 'binopdf' to find the probability of having exactly 1 defective bulb ('P_X_equals_1').
Step 4: Sum these two probabilities to get the probability of at most 1 defective bulb ('P_X_at_most_1').
Step 5: Display Results: Print the probabilities for exactly 2 defective bulbs and at most 1 defective bulb using 'fprintf'.

Program

```
% Parameters
```

```
n = 20;     % number of bulbs selected
p = 0.05;   % probability of a bulb being defective

% Part 1: Probability that exactly 2 bulbs are defective
k = 2;
P_X_equals_2 = binopdf(k, n, p);

% Part 2: Probability that at most 1 bulb is defective (P(X <= 1))
P_X_equals_0 = binopdf(0, n, p);   % Probability of 0 defective bulbs
P_X_equals_1 = binopdf(1, n, p);   % Probability of 1 defective bulb
P_X_at_most_1 = P_X_equals_0 + P_X_equals_1;

% Display results
fprintf('Probability that exactly 2 bulbs are defective: %.4f\n', P_X_equals_2);
fprintf('Probability that at most 1 bulb is defective: %.4f\n', P_X_at_most_1);
```

Output

The output of the program for various inputs is as following:

```
Probability that exactly 2 bulbs are defective: 0.1887
Probability that at most 1 bulb is defective: 0.7358
```

Exercise Problem

A survey finds that 40% of people prefer coffee over tea. If 12 people are randomly selected,

1. What is the probability that exactly 5 people prefer coffee?
2. What is the probability that at least 8 people prefer coffee?

Practical No. 9

Aim

To solve a given problem of Geometric random variable with the help of MATLAB.

Problem

A fair coin is flipped repeatedly until it lands on heads.
1. What is the probability that the first heads appears on the 4th flip?
2. What is the probability that it takes at most 3 flips to get the first heads?

Theory

- **Step 1: Given Information:**
  Since the coin is fair, the probability of getting heads on any flip is p = 0.5.
  Let X be the number of flips until the first heads appears. X follows a geometric distribution:

  $X \approx \text{Geometric}(p = 0.5)$

- **Step 2: Probability that the first heads appears on the 4th flip**
  The probability that the first heads appears on the k-th flip in a geometric distribution is:
  P(X = k) = `(1 - p)^(k - 1)` * p

  For k = 4:
  P(X = 4) = `(1 - 0.5)^3`* 0.5 = `0.5^4` = 0.0625

  So, the probability that the first heads appears on the 4th flip is 0.0625.

- **Step 3: Probability that it takes at most 3 flips to get the first heads**
  To find the probability that it takes at most 3 flips, we need to find $P(X \leq 3)$:
  $P(X \leq 3) = P(X = 1) + P(X = 2) + P(X = 3)$

  Using the formula:

  P(X = k) = `(1 - p)^(k - 1)` * p

  Calculations:
  P(X = 1) = `(0.5)^0` * 0.5 = 0.5
  P(X = 2) = `(0.5)^1` * 0.5 = 0.25
  P(X = 3) = `(0.5)^2` * 0.5 = 0.125

  Summing these:

  $P(X \leq 3) = 0.5 + 0.25 + 0.125 = 0.875$

  So, the probability that it takes at most 3 flips to get the first heads is 0.875.

Algorithm

Step 1: Define Parameters: Set 'p = 0.5', representing the probability of getting heads on each coin flip.
Step 2: Calculate the Probability of First Heads on the 4th Flip.
Step 3: Calculate the Probability of Getting First Heads within the First 3 Flips. Sum these probabilities to get 'P_X_at_most_3', which represents the probability of getting the first heads within the first 3 flips.
Step 4: Display Results: Print 'P_X_equals_4' and 'P_X_at_most_3' using 'fprintf' to show the calculated probabilities in a formatted way.

Program

```
% Parameters
p = 0.5;  % Probability of getting heads on each flip

% Part 1: Probability that the first heads appears on the 4th flip
k = 4;
P_X_equals_4 = (1 - p)^(k - 1) * p;

% Part 2: Probability that it takes at most 3 flips to get the first heads
P_X_equals_1 = (1 - p)^(1 - 1) * p;  % Probability of first heads on 1st flip
P_X_equals_2 = (1 - p)^(2 - 1) * p;  % Probability of first heads on 2nd flip
P_X_equals_3 = (1 - p)^(3 - 1) * p;  % Probability of first heads on 3rd flip

P_X_at_most_3 = P_X_equals_1 + P_X_equals_2 + P_X_equals_3;

% Display results
fprintf('Probability that the first heads appears on the 4th flip: %.4f\n',
P_X_equals_4);
fprintf('Probability that it takes at most 3 flips to get the first heads:
%.4f\n', P_X_at_most_3);
```

Output

The output of the program for various inputs is as following:

```
Probability that the first heads appears on the 4th flip: 0.0625
Probability that it takes at most 3 flips to get the first heads: 0.8750
```

Exercise Problem

A salesperson makes calls to potential clients, and each call has a 20% chance of success (i.e., closing a sale), independently of other calls.

1. What is the probability that the first successful sale happens on the 6th call?
2. What is the probability that it takes fewer than 4 calls to make the first sale?

Practical No. 10

Aim

To solve a given problem of Poisson random variable with the help of MATLAB.

Problem

A bookstore experiences an average of 3 customer arrivals per hour (following Poisson Distribution)
1. What is the probability that exactly 5 customers arrive in an hour?
2. What is the probability that at most 2 customers arrive in an hour?

Theory

- **Step 1: Given Information:**
  The average rate of customer arrivals, $\lambda = 3$ per hour.
  Let X be the number of customers arriving in one hour. X follows a Poisson distribution:

  $X \approx \text{Poisson}(\lambda = 3)$

- **Step 2: Probability that exactly 5 customers arrive in an hour**
  The probability of observing exactly k events in a Poisson distribution is given by:
  P(X = k) = (λ^k* e^($-\lambda$))/ k!

  For k = 5:
  P(X = 5) = (3^5 * e^(-3)) / 5!= 0.1008

- **Step 3: Probability that at most 2 customers arrive in an hour**
  To find the probability that at most 2 customers arrive in an hour, we need to calculate $P(X \leq 2)$:

  $P(X \leq 2) = P(X = 0) + P(X = 1) + P(X = 2)= 0.4232$

  Using the Poisson formula, we calculate each probability:
  P(X = 0) = (3^0* e^(-3)) / 0!
  P(X = 1) = (3^1* e^(-3)) / 1!
  P(X = 2) = (3^2* e^(-3)) / 2!

Algorithm

Step 1: Define Parameters: Set 'lambda = 3', representing the average rate of customer arrivals per hour.
Step 2: Calculate the Probability of Exactly 5 Arrivals in an Hour: Define 'k = 5', specifying that we want the probability of exactly 5 customers arriving in an hour. Use the 'poisspdf' function with parameters 'k' and 'lambda' to compute 'P_X_equals_5', the probability of exactly 5 arrivals.
Step 3: Calculate the Probability of at Most 2 Arrivals in an Hour: Compute probabilities for 0, 1, and 2 customers arriving in an hour using 'poisspdf':
Step 4: Display Results: Print 'P_X_equals_5' and 'P_X_at_most_2' using 'fprintf' to display the calculated probabilities in a formatted manner.

Program

```
% Parameters
lambda = 3;  % average rate of customer arrivals per hour

% Part 1: Probability that exactly 5 customers arrive in an hour
k = 5;
P_X_equals_5 = poisspdf(k, lambda);

% Part 2: Probability that at most 2 customers arrive in an hour (P(X <= 2))
P_X_equals_0 = poisspdf(0, lambda);  % Probability of 0 customers arriving
P_X_equals_1 = poisspdf(1, lambda);  % Probability of 1 customer arriving
P_X_equals_2 = poisspdf(2, lambda);  % Probability of 2 customers arriving

P_X_at_most_2 = P_X_equals_0 + P_X_equals_1 + P_X_equals_2;

% Display results
fprintf('Probability that exactly 5 customers arrive in an hour: %.4f\n',
P_X_equals_5);
fprintf('Probability that at most 2 customers arrive in an hour:%.4f\n',
P_X_at_most_2);
```

Output

```
Probability that exactly 5 customers arrive in an hour: 0.1008
Probability that at most 2 customers arrive in an hour: 0.4232
```

Exercise problem

A doctors office receives an average of 6 patients per hour.
1. What is the probability that no patients arrive in a particular hour?
2. What is the probability that at least 4 patients arrive in a particular hour?

Practical No. 11

Aim

To solve a given problem of Uniform distribution with the help of MATLAB.

Problem

A continuous random variable X is uniformly distributed on the interval [2, 8].
1. Find the probability that X takes a value between 3 and 6.
2. Calculate the expected value and variance of X.
3. Find the probability that X is greater than 7.

Theory

- **Step 1: Setting Up the Uniform Distribution**
  Since X is uniformly distributed on [2, 8], the probability density function (PDF) of X is given by:
  f(x) = 1 / (b - a) = 1 / (8 - 2) = 1 / 6
  for x in [2, 8].

- **Step 2: Probability that X takes a value between 3 and 6**
  The probability that X lies between 3 and 6 can be calculated as:
  P(3 $\leq$ X $\leq$ 6) = $\int_3^6 f(x)dx = \int_3^6 \frac{1}{6}dx$

  This integral simplifies to:
  P(3 $\leq$ X $\leq$ 6) = (1/6) * (6 - 3) = (1/6) * 3 = 0.5

- **Step3: Expected Value and Variance of X**
  For a uniform random variable X on [a, b], the expected value E[X] and variance Var(X) are:
  E[X] = (a + b) / 2 = (2 + 8) / 2 = 5
  Var(X) = `((b-a)^2)`/ 12 = `((8-2)^2)` / 12 = 36 / 12 = 3

- **Step4: Probability that X is greater than 7**
  The probability that X is greater than 7 is:
  P(X > 7) = $\int_8^7 f(x)dx = \int_8^7 \frac{1}{6}dx$
  This simplifies to:
  P(X > 7) = (1/6) * (8 - 7) = (1/6) * 1 = 1/6 $\approx$ 0.1667

Algorithm

Step 1: Define the Interval: Set ‘a = 2’ and ‘b = 8’, defining the interval ‘[a, b]’ over which the uniform distribution for X is defined.
Step 2: Calculate the Probability that X is Between 3 and 6.
Step 3: Calculate Expected Value and Variance.
Step 4: Calculate the Probability that X is Greater than 7.
Step 5: Display Results.

Program

```
% Define the interval [a, b]
a = 2;
b = 8;

% Part 1: Probability that X takes a value between 3 and 6
p_3_6 = (6 - 3) / (b - a);

% Part 2: Expected value and variance
expected_value = (a + b) / 2;
variance = (b - a)^2 / 12;

% Part 3: Probability that X is greater than 7
p_greater_7 = (b - 7) / (b - a);

% Display results
fprintf('Probability that X is between 3 and 6: %.4f\n', p_3_6);
fprintf('Expected value of X: %.2f\n', expected_value);
fprintf('Variance of X: %.2f\n', variance);
fprintf('Probability that X is greater than 7: %.4f\n', p_greater_7);
```

Output

The output of the program for various inputs is as following:

```
Probability that X is between 3 and 6: 0.5000
Expected value of X: 5.00
Variance of X: 3.00
Probability that X is greater than 7: 0.1667
```

Exercise problem

A random variable Z is uniformly distributed between 0 and 10.
1. What is the probability density function (PDF) of Z?
2. Calculate the probability that Z falls between 2 and 8.
3. Determine the median value of Z.

Practical No. 12

Aim

To solve a given problem of exponential random variable with the help of MATLAB.

Problem

The time until a machine fails is modeled by an exponential random variable with a mean of 5 hours.
1. Find the probability that the machine fails within 2 hours.
2. Find the probability that the machine operates for more than 6 hours before failing.

Theory

- **Step 1: Setting Up the Exponential Distribution**
  For an exponential random variable T with mean $\mu = 5$ hours, the rate parameter λ is:
  $\lambda = 1 / \mu = 1 / 5 = 0.2$
  The probability density function (PDF) for an exponential distribution is:
  $f(t) = \lambda * \exp(-\lambda * t)$ for $t \leq 0$.

- **Step 2: Probability that the Machine Fails Within 2 Hours**
  The probability that $T \leq 2$ is given by the cumulative distribution function (CDF):
  $P(T \leq 2) = 1 - \exp(-\lambda * 2) = 1 - \exp(-0.2 * 2)$
  Substitute $\lambda = 0.2$:
  $P(T \leq 2) = 1 - \exp(-0.4) \approx 0.3297$

- **Step 3: Probability that the Machine Operates for More Than 6 Hours**
  The probability that $T > 6$ is:
  $P(T > 6) = \exp(-\lambda * 6) = \exp(-0.2 * 6)$
  Substitute $\lambda = 0.2$:
  $P(T > 6) = \exp(-1.2) \approx 0.3012$.

Algorithm

Step 1: Define the Rate Parameter: Set 'lambda = 1 / 5', representing the failure rate (average time between failures is 5 hours).
Step 2: Calculate the Probability that the Machine Fails within 2 Hours.
Step 3: Calculate the Probability that the Machine Operates for More than 6 Hours.
Step 4: Display Results.

Program

```
% Define the rate parameter lambda
lambda = 1 / 5;

% Part 1: Probability that the machine fails within 2 hours
p_within_2 = 1 - exp(-lambda * 2);
```

```
% Part 2: Probability that the machine operates for more than 6 hours
p_greater_6 = exp(-lambda * 6);

% Display results
fprintf('Probability that the machine fails within 2 hours: %.4f\n',
p_within_2);
fprintf('Probability that the machine operates for more than 6 hours: %.4f\n',
p_greater_6);
```

Output

The output of the program for various inputs is as following:

```
Probability that the machine fails within 2 hours: 0.3297
Probability that the machine operates for more than 6 hours: 0.3012
```

Exercise Problem

The lifetime of a light bulb is modeled by an exponential random variable with a mean lifetime of 1000 hours.
1. Calculate the probability that the light bulb lasts more than 1200 hours.
2. Find the probability that the light bulb fails between 800 and 1200 hours.
3. Determine the variance of the light bulb's lifetime.

Practical No. 13

Aim

To solve a given problem of Gamma random variable with the help of MATLAB.

Problem

Let Y be a Gamma random variable with shape parameter $\alpha = 5$ and scale parameter $\beta = 1.5$.
1. Calculate the probability P(Y $\leq$ 7).
2. Determine the probability P(6 $\leq$ Y $\leq$ 10).

Theory

- **Step 1:** The Gamma distribution with shape parameter α and scale parameter β has the probability density function:
  $f(y) = \frac{y^{\alpha-1}e^{\frac{-y}{\beta}}}{\beta^{\alpha}\Gamma\alpha}$, where $\Gamma(\alpha)$ is the Gamma function.
- **Step 2: Calculate P(Y $\leq$ 7)**
  The cumulative distribution function (CDF) of a Gamma random variable can be used to calculate **P**(Y $\leq$ 7).
- **Step 3: Calculate P(6 $\leq$ Y $\leq$ 10)**
  This is the probability that Y falls between 6 and 10, which can be expressed as: **P**(6 $\leq$ Y $\leq$ 10) = **P**(Y $\leq$ 10) - **P**(Y $\leq$ 6).

  Using the CDF again, we can calculate P(Y $\leq$ 10) and P(Y $\leq$ 6).

Algorithm

Step 1: Define Parameters: Set 'alpha = 5' and 'beta = 1.5', which are the shape and scale parameters for the Gamma distribution.
Step 2: Calculate P(Y≤7): Use 'gamcdf (7, alpha, beta)' to calculate 'P_Y_leq_7', the probability that Y takes a value less than or equal to 7.
Step 3: Calculate P (6 $\leq$ Y $\leq$ 10): Calculate 'P_Y_leq_10' as 'gamcdf(10, alpha, beta)', the probability that Y is less than or equal to 10.
Step 4: Display Results: Print 'P_Y_leq_7' and 'P_6_leq_Y_leq_10' using 'disp' to show the calculated probabilities in a formatted way.

Program

```
% Given parameters
alpha = 5;     % Shape parameter
beta = 1.5;    % Scale parameter

% Part 1: Calculate P(Y <= 7)
P_Y_leq_7 = gamcdf(7, alpha, beta);

% Part 2: Calculate P(6 <= Y <= 10)
P_Y_leq_10 = gamcdf(10, alpha, beta);
P_Y_leq_6 = gamcdf(6, alpha, beta);
P_6_leq_Y_leq_10 = P_Y_leq_10 - P_Y_leq_6;

% Display results
disp("Part 1: P(Y <= 7) = " + P_Y_leq_7);
disp("Part 2: P(6 <= Y <= 10) = " + P_6_leq_Y_leq_10);
```

Output

```
Part 1: P(Y <= 7) = 0.49921
Part 2: P(6 <= Y <= 10) = 0.42321
```

Exercise problem

A machine part has a lifetime that follows a Gamma distribution with a shape parameter of 4 and a mean lifetime of 12 hours.
1. Find the scale parameter β.
2. Calculate the probability that the part lasts more than 10 hours.
3. If the part has already lasted 10 hours, find the probability that it will last an additional 5 hours.

Practical No. 14

Aim

To solve a given problem of Normal distribution with the help of MATLAB.

Problem

The scores on a standardized exam are normally distributed with a mean of 500 and a standard deviation of 100.
Let X represent the exam scores.
1. What is the probability that a randomly selected student scores above 600?
2. What is the probability that a student scores between 450 and 550?

Theory

- **Step 1: Given data**
  Mean $\mu = 500$, Standard deviation $\sigma = 100$
  For a normally distributed random variable X, we can use the Z-score formula to standardize the problem:
  Z = (X - μ) / σ
- **Step 2: Probability that a student scores above 600**
  To find P(X > 600), we need the Z-score for X = 600:

Z = (600 - 500) / 100 = 1. Now,**P**(X > 600) = **P**(Z > 1). We can use standard normal distribution tables to find this probability.

From the standard normal table:
P(Z > 1) ≈ 0.1587.

- **Step 3: Probability that a student scores between 450 and 550**
  We want to find **P**(450 < X < 550). Well calculate the Z-scores for both bounds: For X = 450:
  Z = (450 - 500) / 100 = -0.5.

  For X = 550:
  Z = (550 - 500) / 100 = 0.5.

  Now, **P**(450 < X < 550) = **P**(-0.5 < Z < 0.5).

  Using the standard normal table:
  P(-0.5 < Z < 0.5) ≈ 0.3829.

Algorithm

Step 1: Initialize Parameters: Set mean ('mu = 500') and standard deviation ('sigma = 100').
Step 2: Calculate Probability for Score Above 600.
a. Set the score threshold to 600.
b. Calculate the Z-score for 600.
c. Use the cumulative distribution function (CDF) to find the probability of scoring above 600.
Step 3: Calculate Probability for Score Between 450 and 550.
a. Set the lower bound to 450 and upper bound to 550.
b. Calculate the Z-scores for both 450 and 550.
c. Use the CDF to find the probability of scoring between 450 and 550.
Step 4: Display Results: Print the probabilities calculated for both cases.

Program

```
% Given parameters
mu = 500;
sigma = 100;

% 1. Probability that a student scores above 600
X1 = 600;
Z1 = (X1 - mu) / sigma;
P1 = 1 - normcdf(Z1);
```

```
fprintf('1. Probability that a student scores above 600: %.4f\n', P1);

% 2. Probability that a student scores between 450 and 550
X2_lower = 450;
X2_upper = 550;
Z2_lower = (X2_lower - mu) / sigma;
Z2_upper = (X2_upper - mu) / sigma;
P2 = normcdf(Z2_upper) - normcdf(Z2_lower);
fprintf('2. Probability that a student scores between 450 and 550: %.4f\n', P2);
```

Output

The output of the program for various inputs is as following:

```
Probability that a student scores above 600: 0.1587
Probability that a student scores between 450 and 550: 0.3829
```

Exercise Problem

The lifetime of a particular machine component is normally distributed with a mean of 2000 hours and a standard deviation of 150 hours. Let X represent the lifetime of the component.
1. What is the probability that a randomly selected component lasts less than 1850 hours?
2. What is the probability that a component lasts between 1900 and 2100 hours?

Practical No. 15

Aim

To solve a given problem of joint probability distribution with the help of MATLAB.

Problem

Let X and Y be two discrete random variables with the following joint probability distribution:

```
      Y=1    Y=2     Y=3
X=1  0.1    0.2     0.1
X=2  0.2    0.1     0.1
X=3  0.1    0.1     0.1
```

1. Find the marginal distributions of X and Y.
2. Calculate $\mathbf{P}(X = 2 \mid Y = 2)$.
3. Determine if X and Y are independent.

Theory

- **Step 1: Marginal Distributions of X and Y**

  1. Marginal distribution of X:

  To find the marginal probability of X, sum the joint probabilities across each row (for all values of Y):

  P(X=1) = 0.1 + 0.2 + 0.1 = 0.4
  P(X=2) = 0.2 + 0.1 + 0.1 = 0.4
  P(X=3) = 0.1 + 0.1 + 0.1 = 0.3

  So, the marginal distribution of X is:
  P(X) = {0.4, 0.4, 0.3}

  2. Marginal distribution of Y:

  To find the marginal probability of Y, sum the joint probabilities across each column (for all values of X):

P(Y=1) = 0.1 + 0.2 + 0.1 = 0.4
P(Y=2) = 0.2 + 0.1 + 0.1 = 0.4
P(Y=3) = 0.1 + 0.1 + 0.1 = 0.3

So, the marginal distribution of Y is:
P(Y) = {0.4, 0.4, 0.3}

- **Step 2: Calculate P(X = 2 | Y = 2)**

  The conditional probability **P**(X=2 | Y=2) is calculated using the formula:

  P(X=2 | Y=2) = **P**(X=2, Y=2) / **P**(Y=2)

  From the table, **P**(X=2, Y=2) = 0.1 and **P**(Y=2) = 0.4.

  P(X=2 | Y=2) = 0.1 / 0.4 = 0.25

- **Step 3: Determine if X and Y are Independent**

  To check if X and Y are independent, we need to verify if **P**(X=x, Y=y) = **P**(X=x) * **P**(Y=y) for all pairs (x, y).

  Let's check a few cases:

  For X=1 and Y=1:

  P(X=1, Y=1) = 0.1 and **P**(X=1) * **P**(Y=1) = 0.4 * 0.4 = 0.16

  Since 0.1 $\neq$ 0.16, X and Y are not independent.

Algorithm

Step 1: Define Joint Probability Matrix.
Step 2: Calculate Marginal Distribution of X.
Step 3: Calculate Marginal Distribution of Y.
Step 4: Calculate Conditional Probability P(X=2 | Y=2).
Step 5: Check for Independence.

Program

```
% Define the joint probability distribution matrix
P_XY = [0.1 0.2 0.1;
        0.2 0.1 0.1;
        0.1 0.1 0.1];

% Calculate the marginal distribution of X
P_X = sum(P_XY, 2); % Sum along rows
disp('Marginal distribution of X:');
disp(P_X);

% Calculate the marginal distribution of Y
P_Y = sum(P_XY, 1); % Sum along columns
disp('Marginal distribution of Y:');
disp(P_Y);

% Calculate P(X=2 | Y=2)
P_X2_Y2 = P_XY(2,2); % P(X=2, Y=2)
P_Y2 = P_Y(2);       % P(Y=2)
P_X_given_Y = P_X2_Y2 / P_Y2;
disp('P(X=2 | Y=2):');
disp(P_X_given_Y);

% Check independence by comparing P(X=x, Y=y) with P(X=x) * P(Y=y)
independence_check = false;
for i = 1:3
    for j = 1:3
        if abs(P_XY(i,j) - P_X(i) * P_Y(j)) > 1e-6
            independence_check = true;
            break;
        end
    end
end

if independence_check
    disp('X and Y are not independent.');
else
    disp('X and Y are independent.');
end
```

Output

```
Marginal distribution of X:
    0.4000
    0.4000
    0.3000

Marginal distribution of Y:
    0.4000    0.4000    0.3000

P(X=2 | Y=2):
    0.2500

X and Y are not independent.
```

Exercise problem

Let X and Y be jointly distributed random variables with the joint probability function:

$$\mathbf{P}(X=x,Y=y)=(x+y)/30,\ x,y=1,2,3.$$

1. Verify that this is a valid probability distribution.
2. Find the expected values E(X)) and E(Y).
3. Calculate the covariance Cov(X,Y).

Practical No. 16

Aim

To solve a given problem of Joint probability with the help of MATLAB.

Problem

A fair six-sided die and a fair four-sided die are rolled. Let X be the outcome of the six-sided die, and Y be the outcome of the four-sided die.
1. Find the joint probability $\mathbf{P}(X = 2, Y = 3)$.
2. What is the probability that the sum of the two dice is 7?
3. Are X and Y independent?

Theory

- **Step 1: To find $\mathbf{P(X = 2, Y = 3)}$**
  Since the dice are fair, each outcome is equally likely. There are 6 4 = 24 possible

outcomes when both dice are rolled.
The probability of any specific outcome (X = x, Y = y) is:
P(X = x, Y = y) = 1/24
Thus,
P(X = 2, Y = 3) = 1/24 ≈ 0.0417

- **Step 2: Probability that the Sum of the Two Dice is 7**
  The possible pairs (X, Y) where the sum is 7 are: (X = 3, Y = 4), (X = 4, Y = 3), (X = 5, Y = 2), (X = 6, Y = 1). Each of these pairs has a probability of 1/24.
  So,
  P(X + Y = 7) = 4 1/24 = 1/6 ≈ 0.1667

- **Step 3: Are X and Y Independent?**
  To check independence, we need to verify whether P(X = x, Y = y) = P(X = x) P(Y = y) for all values of x and y.
  Since each outcome is equally likely and both dice are fair:
  P(X = x) = 1/6 for x = 1, 2, 3, 4, 5, 6
  P(Y = y) = 1/4 for y = 1, 2, 3, 4.

  The joint probability **P**(X = x, Y = y) = 1/24.
  For independence, we would need **P**(X = x) **P**(Y = y) = 1/6 1/4 = 1/24, which holds true for all x and y. Therefore, X and Y are independent.

Algorithm

Step 1: Calculate total outcomes.
Step 2: Compute P(X = 2, Y = 3) and display it.
Step 3: Compute P (X + Y = 7) and display it.
Step 4: Check if X and Y are independent by comparing joint probabilities.
Step 5: Display the result of the independence check.

Program

```
% Define the probabilities
total_outcomes = 6 * 4;
prob_X2_Y3 = 1 / total_outcomes;

% Display Part 1: P(X=2, Y=3)
fprintf('P(X=2, Y=3) = %.4f\n', prob_X2_Y3);

% Part 2: Calculate probability of sum equal to 7
```

```
favorable_outcomes_sum7 = 4;
prob_sum7 = favorable_outcomes_sum7 / total_outcomes;

% Display Part 2: P(X + Y = 7)
fprintf('P(X + Y = 7) = %.4f\n', prob_sum7);

% Part 3: Independence check
prob_X = 1 / 6;
prob_Y = 1 / 4;
joint_prob = prob_X * prob_Y;

% Check if joint probability matches the outcome probability
is_independent = (joint_prob == prob_X2_Y3);

% Display Part 3: Independence
if is_independent
    fprintf('X and Y are independent.\n');
else
    fprintf('X and Y are not independent.\n');
end
```

Output

```
P(X=2, Y=3) = 0.0417
P(X + Y = 7) = 0.1667
X and Y are independent.
```

Exercise problem

Two fair dice are rolled, one with 8 sides (numbered 1 to 8) and one with 6 sides (numbered 1 to 6). Let A represent the outcome of the 8-sided die and B represent the outcome of the 6-sided die.
1. Find the joint probability **P**(A=5, B=2).
2. What is the probability that the sum of the two dice is 9?
3. Are A and B independent?

Practical No. 17

Aim

To solve a given problem of Bivariate variables with the help of MATLAB.

Problem

Let X and Y be continuous random variables with joint probability density function (PDF):

$$f_{X,Y}(x,y) = \begin{cases} c(x+y), & 0 \le x \le 1,\ 0 \le y \le 1 \\ 0, & \text{otherwise} \end{cases}$$

1. Determine the value of **c** such that $f_{X,Y}(x,y)$ is a valid joint PDF.
2. Find the marginal probability density functions $f_X(x)$ and $f_Y(y)$.
3. Calculate the conditional density $f_{X|Y}(x \mid y)$ for 0 ≤ y ≤ 1.

Theory

- **Step 1: To find the value of c so that $f_{X,Y}(x,y)$ is a valid joint PDF**
  To ensure that $f_{X,Y}(x,y)$ is a valid joint probability density function, the integral over all possible values of x and y must equal 1:

  $$\int_0^1 \int_0^1 f_{X,Y}(x,y)dxdy = 1$$

  Since $f_{X,Y}(x,y) = c(x+y)$ within the range $0 \le x \le 1,\ 0 \le y \le 1$, we can set up the integral:

  $$\int_0^1 \int_0^1 c(x+y)dxdy = 1$$

  Evaluating this integral:

  $$c = 1.$$

  Thus, the joint PDF is: $f_{X,Y}(x,y) = (x+y), 0 \le x \le 1,\ 0 \le y \le 1.$

- **Step 2: To find the Marginal PDFs $f_X(x)$ and $f_Y(y)$.**
  1. Find $f_X(x)$**:**

  The marginal PDF $f_X(x)$ is obtained by integrating the joint PDF over y:
  $f_X(x) = \int_0^1 f_{X,Y}(x,y)dy = \int_0^1 (x+y)dy = x + \frac{1}{2}, 0 \le x \le 1.$

  2. Find $f_Y(y)$**:**

  The marginal PDF $f_Y(y)$ is obtained by integrating the joint PDF over y:
  $f_Y(y) = \int_0^1 f_{X,Y}(x,y)dy = \int_0^1 (x+y)dy = y + \frac{1}{2}, 0 \le y \le 1.$

- **Step 3: To find the Conditional PDF $f_{X|Y}(x \mid y)$**

  The conditional PDF $f_{X|Y}(x \mid y)$ is defined as:

$$f_{X|Y}(x \mid y) = \frac{f_{X,Y}(x,y)}{f_Y(y)} = \frac{x+y}{y+\frac{1}{2}}, \quad 0 \le y \le 1.$$

Algorithm

Step 1: Define symbolic variables x, y, and c.
Step 2: Define joint PDF.
Step 3: Compute normalization constant.
Step 4: Give the output the value of the normalization constant c.
Step 5: Substitute c into PDF.
Step 6: Print marginal PDF of X.
Step 7: Compute marginal PDF of Y.
Step 8: Compute conditional PDF of X given Y.

Program

```
syms x y c
fxy = c * (x + y);
integral_xy = int(int(fxy, x, 0, 1), y, 0, 1);
c_val = solve(integral_xy == 1, c);
fprintf('The value of c is: %f\n', double(c_val));

fxy = subs(fxy, c, c_val);

fx = int(fxy, y, 0, 1);
fprintf('Marginal PDF f_X(x) = %s\n', fx);

fy = int(fxy, x, 0, 1);
fprintf('Marginal PDF f_Y(y) = %s\n', fy);

fxy_val = subs(fxy, c, c_val);
fX_given_Y = fxy_val / fy;
fprintf('Conditional PDF f_X|Y(x|y) = %s\n', simplify(fX_given_Y));
```

Output

```
The value of c is: 1.000000
Marginal PDF f_X(x) = x + 1/2
Marginal PDF f_Y(y) = y + 1/2
Conditional PDF f_X|Y(x|y) = (x + y)/(y + 1/2)
```

Exercise Problem

Suppose X and Y are continuous random variables with the joint probability density function given by:

$$f_{X,Y}(x,y) = \begin{cases} ce^{-(x+2y)}, & x \geq 0,\ y \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

1. Determine the constant c to ensure that $f_{X,Y}(x,y)$ is a valid joint PDF.
2. Find the marginal probability density functions $f_X(x)$ and $f_Y(y)$.
3. Calculate the conditional density $f_{Y|X}(y \mid x)$ for x $\geq$ 0.

Practical No. 18

Aim

To solve a given problem of Markovs inequality with the help of MATLAB.

Problem

Suppose X is a non-negative random variable with an expected value E[X] = 5.
1. Use Markov's inequality to find an upper bound for **Pr**(X $\geq$ 10).
2. Use the same inequality to find an upper bound for **Pr**(X $\geq$ 20).

Theory

- **Step 1:** Markov's Inequality states that for any non-negative random variable X and any a > 0,

$$\mathbf{Pr}(X \geq a) \leq \frac{E[X]}{a}$$

  We are given: E[X] = 5

- **Step 2: Upper bound for Pr(X $\geq$ 10)**
  Using Markov's inequality:

  Pr(X $\geq$ 10) $\leq$ E[X] / 10 = 5 / 10 = 0.5.

  So, the probability that X $\geq$ 10 is at most 0.5.

- **Step 3: : Upper bound for Pr(X $\geq$20)**
  Using Markov's inequality again:

  Pr(X $\geq$ 20) $\leq$ E[X] / 20 = 5 / 20 = 0.25.

  So, the probability that X $\geq$ 20 is at most 0.25.

Algorithm

Step 1: Set the Expected Value.
Step 2: Define Threshold Values.
Step 3: Apply Markovs Inequality
Step 4: Compute Upper Bound for a1.
Step 5: Compute Upper Bound for a2.
Step 6: Display the Results.

Program

```
% Given expected value
E_X = 5;

% Values of a for which we want to compute the upper bound
a1 = 10;
a2 = 20;

% Using Markov's inequality to compute upper bounds
P_X_geq_a1 = E_X / a1;
P_X_geq_a2 = E_X / a2;

% Display the results
fprintf('Upper bound for P(X >= 10) using Markov''s inequality: %.2f\n',
P_X_geq_a1);
fprintf('Upper bound for P(X >= 20) using Markov''s inequality: %.2f\n',
P_X_geq_a2);
```

Output

```
Upper bound for P(X >= 10) using Markov's inequality: 0.50
Upper bound for P(X >= 20) using Markov's inequality: 0.25
```

Exercise Problem

Let Y be a non-negative random variable with E[Y]=8.
1. Use Markov's inequality to find an upper bound for **Pr**(Y $\geq$ 16).
2. Use the same inequality to find an upper bound for **Pr**(Y $\geq$ 24).

Practical No. 19

Aim

To solve a given problem of modes of convergence with the help of MATLAB.

Problem

Let $(X_n)_{n\geq 1}$ be a sequence of random variables defined by $(X_n) = \frac{1}{n}$.
1. Calculate $\mathbf{P}(\mid X_n - 0 \mid > \epsilon)$ for $\epsilon = 0.1$, 0.05, and 0.01 with n = 10, 20, and 50.
2. Determine if X_n converges to 0 in probability as n ∞.

Theory

- **Step 1:** Since $X_n = 1/n$, we have:
  $\mathbf{P}(\mid X_n - 0 \mid > \epsilon) = P(\mid 1/n \mid > \epsilon)$.
- **Step 2:** This inequality can be simplified as follows:
  P(1/n > ϵ) = 1 if 1/n > ϵ, otherwise P(1/n > ϵ) = 0.

For each value of n and ϵ, we can evaluate whether P($\mid X_n - 0 \mid > \epsilon$) = 1 or 0.

Algorithm

Step 1: Define input values.
Step 2: Initialize Probability matrix.
Step 3: Loop through each n and ϵ.
Step 4: Display results.

Program

```
% Define the parameters
n_values = [10, 20, 50];
```

```
epsilon_values = [0.1, 0.05, 0.01];

% Initialize a matrix to store the probabilities
probabilities = zeros(length(n_values), length(epsilon_values));

% Loop through each combination of n and epsilon
for i = 1:length(n_values)
    n = n_values(i);
    X_n = 1 / n;
    for j = 1:length(epsilon_values)
        epsilon = epsilon_values(j);
        if X_n > epsilon
            probabilities(i, j) = 1;  % Probability is 1 if X_n > epsilon
        else
            probabilities(i, j) = 0;  % Probability is 0 otherwise
        end
    end
end

% Display the results
fprintf('Results for P(|X_n - 0| > epsilon):\n');
fprintf('   n   epsilon=0.1   epsilon=0.05   epsilon=0.01\n');
for i = 1:length(n_values)
    fprintf('%4d   %8.2f       %8.2f         %8.2f\n', n_values(i),
    probabilities(i, :));
end
```

Output

The output of the program for various inputs is as following:

```
Results for P(|X_n - 0| > epsilon):
   n   epsilon=0.1   epsilon=0.05   epsilon=0.01
  10       0.00           1.00             1.00
  20       0.00           0.00             1.00
  50       0.00           0.00             1.00
```

Exercise Problem

Let $(Y_n)_{n\geq 1}$ be a sequence of random variables defined by $Y_n = \frac{2}{n}$.
1. Calculate P($| Y_n - 0 |> \epsilon$) for $\epsilon = 0.1$, 0.05, and 0.01 with n=5, 10, and 20.
2. Determine if Y_n converges to 0 in probability as n ∞.

Practical No. 20

Aim

To solve a given problem of weak and strong laws of Large numbers with the help of MATLAB.

Problem

A factory produces light bulbs, and the lifetime of each light bulb (in hours) is a random variable with a mean of 1,000 hours and a standard deviation of 100 hours. A random sample of 200 light bulbs is selected. Using the Weak Law of Large Numbers, estimate the probability that the sample average lifetime is within 10 hours of the expected lifetime.

Theory

We can solve this using Chebyshevs Inequality.

- **Step 1: Define the Range of Sample Mean**

  We are asked to find the probability that the sample mean is within 10 hours of the population mean. This means we want:

  $$\mathbf{P}(\mid \bar{X} - \mu \mid < 10)$$

- **Step 2: Use Chebyshev's Inequality**

  For a random sample mean X based on n samples, Chebyshev's inequality states:

  $$\mathbf{P}(\mid \bar{X} - \mu \mid \geq k) \leq \sigma^2 \backslash (nk^2)$$

  where:
  σ^2 is the population variance
  n is the sample size
  k is the desired bound (in this case, 10 hours)

- **Step 3: Calculate the Variance of the Sample Mean**
  The variance of the sample mean, Var$\bar{X}$, is given by:
  Var$\bar{X} = \frac{\sigma^2}{n}$
  Given $\sigma = 100$ and n = 200,

  we find: Var$\bar{X}$ = 100 / 200 = 10000 / 200 = 50

- **Step 4: Apply Chebyshev's Inequality**
  We want $\mathbf{P}(\mid \bar{X} - \mu \mid < 10)$, which is the complement of $\mathbf{P}(\mid \bar{X} - \mu \mid \geq 10)$.
  Using Chebyshevs inequality:
  $\mathbf{P}(\mid \bar{X} - \mu \mid \geq 10) \leq 50/10^2 = 50/100 = 0.5$

  Thus:
  $\mathbf{P}(\mid \bar{X} - \mu \mid < 10) \geq 1 - 0.5 = 0.5$

This implies that there is at least a 50 % probability that the sample mean will be within 10 hours of the expected lifetime.

Algorithm

Step 1: Define the parameters for the mean ('mu'), standard deviation ('sigma'), sample size ('n'), and the distance from the mean ('k').
Step 2: Calculate Chebyshev's Bound.
Step 3: Simulate Sample Means.
Step 4: Run Simulations.

Program

```
% Parameters
mu = 1000;            % Mean lifetime in hours
sigma = 100;          % Standard deviation in hours
n = 200;              % Sample size
k = 10;               % Distance from the mean

% Using Chebyshev's Inequality
var_Xbar = sigma^2 / n;
chebyshev_bound = var_Xbar / k^2;
prob_within_k = 1 - chebyshev_bound;

fprintf('Probability (by Chebyshev's inequality) that sample mean is within
%d hours: %.2f\n', k, prob_within_k);

% Simulation to verify Chebyshev's bound
num_simulations = 10000;  % Number of simulations
sample_means = zeros(num_simulations, 1);

for i = 1:num_simulations
    sample = normrnd(mu, sigma, [1, n]);  % Generate a sample of size n
    sample_means(i) = mean(sample);       % Calculate sample mean
end
```

Output

The output of the program for various inputs is as following:

```
Probability (by Chebyshev's inequality) that sample mean is within 10 hours:
0.50
```

Exercise Problem

The height of adult males in a certain city is normally distributed with a mean of 175 cm and a standard deviation of 10 cm. Suppose a random sample of 100 adult males is taken. Using the Weak Law of Large Numbers, estimate the probability that the sample average height is within 1.5 cm of the mean.

Chapter 9

Analytic Geometry

Practical No. 1

Aim

To solve the given problems with the help of MATLAB.

Problem

Find the projection of the vector $\vec{a} = 2\hat{\imath} + 3\hat{\jmath} + 2\hat{k}$ on the vector $\vec{b} = \hat{\imath} + 2\hat{\jmath} + \hat{k}$.

Theory

The projection of the vector $\vec{a}$ on $\vec{b} = \frac{(\vec{a}.\vec{b})}{|\vec{b}|}$.
As $(\vec{a}.\vec{b}) = (2\hat{\imath} + 3\hat{\jmath} + 2\hat{k}).(\hat{\imath} + 2\hat{\jmath} + \hat{k})$ and $|\vec{b}| = \sqrt{1^2 + 2^2 + 1^2} = \sqrt{6}$,
Hence, the projection of the vector $\vec{a}$ on $\vec{b}$ is $= \frac{(\vec{a}.\vec{b})}{|\vec{b}|} = \frac{10}{\sqrt{6}}$.

Algorithm

Step 1: Define vectors $\vec{a}$ and $\vec{b}$;

Step 2: Calculate the dot product $\vec{a} \cdot \vec{b}$;

Step 3: Calculate the magnitude $|\vec{b}|$;

Step 4: Compute the projection of $\vec{a}$ on $\vec{b}$ as $\frac{\vec{a} \cdot \vec{b}}{|\vec{b}|}$.

Program

The following MATLAB script implements the algorithm.

```
a = [2, 3, 2];
b = [1, 2, 1];
dot_ab = dot(a, b);
dot_bb = dot(b, b);
proj_a_on_b = dot_ab/(sqrt(dot_bb));
disp('Projection of a on b:');
disp(proj_a_on_b);
```

Output

The output of the program is as follows:

```
Projection of a on b:
4.082482
```

Exercise

Exercise 1: Find the projection of the vector $\vec{p} = 4\hat{i} + \hat{j} - 3\hat{k}$ on the vector $\vec{q} = \hat{i} + \hat{j} + 2\hat{k}$.
Exercise 2: Find the projection of the vector $\vec{a} = 2\hat{i}+3\hat{j}+5\hat{k}$ on the vector $\vec{b} = -\hat{i}+4\hat{j}-\hat{k}$.
Exercise 3: Find the projection of the vector $\vec{u} = -\hat{i}+2\hat{j}+\hat{k}$ on the vector $\vec{v} = 3\hat{i}-4\hat{j}+\hat{k}$.

Practical No. 2

Problem

Find the direction cosine of the vector joining the points A(1,2,-3) and B(-1,-2,1) directed from A to B.

Theory

The vector $\vec{AB} = \vec{B} - \vec{A}$. Therefore, $\vec{AB} = -2\hat{i} - 4\hat{j} + 4\hat{k}$.

$$|\vec{AB}| = \sqrt{4+16+16} = \sqrt{36} = 6.$$

Hence, the direction cosine of $\vec{AB}$ is $= \frac{\vec{AB}}{|\vec{AB}|} = \left(\frac{-1}{3}, \frac{-2}{3}, \frac{2}{3}\right)$.

Algorithm

Step 1: Define points A and B;

Step 2: Calculate vector $\vec{AB} = B - A$;

Step 3: Calculate the magnitude $|\vec{AB}|$;

Step 4: Determine direction cosines as $\frac{\vec{AB}}{|\vec{AB}|}$.

Program

The following MATLAB script implements the algorithm.

```
A = [1, 2, -3];
B = [-1, -2, 1];
AB = B - A;
magnitude_AB = norm(AB);
cos_alpha =AB(1) /magnitude_AB;
cos_beta = AB(2) /magnitude_AB;
cos_gamma = AB(3) /magnitude_AB;
disp('Direction Cosines of the vector from A to B:');
disp(['cos(alpha) = ', num2str(cos_alpha)]);
disp(['cos(beta) = ', num2str(cos_beta)]);
disp(['cos(gamma) = ', num2str(cos_gamma)]);
```

Output

The output of the program is as follows:

```
Direction Cosines of the vector from A to B:
cos(alpha) = −0.33333
cos(beta) = −0.66667
cos(gamma) = 0.66667
```

Exercise

Exercise 1: Find the direction cosine of the vector joining the points $A(1, 2, 3)$ and $B(-1, -2, 1)$ directed from A to B.
Exercise 2: Find the direction cosine of the vector joining the points $A(1, 2, -3)$ and $B(-1, -2, 1)$ directed from A to B.

Practical No. 3

Problem

Find the radius and center of the sphere $2x^2 + 2y^2 + 2z^2 + 4x + 4z - 44 = 0$.

Theory

The general equation of the sphere is

$$(x-a)^2+(y-b)^2+(z-c)^2=r^2,$$

where (a,b,c) is the center of the sphere, and r is radius. Given the equation

$$\begin{aligned}
&2x^2+2y^2+2z^2+4x+4z-44=0\\
&\implies 2x^2+4x+2y^2+2z^2+4z-44=0\\
&\implies x^2+2x+y^2+z^2+2z-22=0\\
&\implies x^2+2x+1-1+y^2+z^2+2z+1-1-22=0\\
&\implies (x+1)^2+(y-0)^2+(z+1)^2=(\sqrt{24})^2\\
&\implies \text{center}=(-1,0,-1),\\
&\text{radius}=\sqrt{24}.
\end{aligned}$$

Algorithm

Step 1: Use MATLAB symbolic variables to simplify the equation to standard form;

Step 2: Rewrite as $(x+1)^2+(y-0)^2+(z+1)^2=\sqrt{24}^2$;

Step 3: Identify the center as $(-1,0,-1)$ and radius as $\sqrt{24}$;

Step 4: Display results using `disp()`.

Program

The following MATLAB script implements the algorithm.

```
syms x y z
eq = x*x + y*y + z*z + 2*x + 2*z - 22 == 0;
eq = eq / 2;
eq = subs(eq, x*x + 2*x, (x + 1)*(x + 1) - 1);
eq = subs(eq, z*z + 2*z, (z + 1)*(x + 1) - 1);
eq = simplify(eq + 2);
center = [-1, 0, -1];
radius = sqrt(24);
disp('Simplified Equation:');
disp(eq);
disp('Center of the sphere:');
disp(center);
disp('Radius of the sphere:');
disp(radius);
```

Output

The output of the program is as follows:
Simplified Equation:
$(x+1)^2 + (z+1)^2 + y^2 == 24$
Center of the sphere:
$(-1, 0, -1)$
Radius of the sphere:
4.8990

Exercise

Exercise 1: Find the radius and center of the sphere $x^2 + y^2 + z^2 - 2x = 0$.
Exercise 2: Find the radius and center of the sphere $3x^2 + 3y^2 + 3z^2 + 6x + 6z - 66 = 0$.
Exercise 3: Find the radius and center of the sphere $x^2 + y^2 + z^2 + x + z - 10 = 0$.

Practical No. 4

Problem

Find the vector equation of the line

$$\frac{x+3}{2} = \frac{y-5}{4} = \frac{z+6}{4}.$$

Theory

Let t be any variable, such that

$$\begin{aligned} \frac{x+3}{2} = \frac{y-5}{4} &= \frac{z+6}{4} = t \\ \implies x &= 2t - 3, \\ y &= 4t + 5, \\ z &= 4t - 6. \end{aligned}$$

Hence, the vector equation is

$$\begin{aligned} \vec{r} &= x\hat{\imath} + y\hat{\jmath} + z\hat{k} \\ &= (-3\hat{\imath} + 5\hat{\jmath} - 6\hat{k}) + t(2\hat{\imath} + 4\hat{\jmath} + 4\hat{k}). \end{aligned}$$

Algorithm

Step 1: Set equal parts to a parameter t to express x, y, and z in terms of t;

Step 2: Write the vector equation $\vec{r} = \vec{P} + t \cdot \vec{d}$.

Program

The following MATLAB script implements the algorithm.

```
syms t;
P = [-3, 5, -6];
d = [2, 4, 2];
r_t = P + t * d;
disp('The vector equation of the line is:');
disp('r(t) = ');
disp(r_t);
```

Output

The output of the program is as follows:

```
The vector equation of the line is:
r(t) = [2 * t - 3, 4 * t + 5, 2 * t - 6]
```

Exercise

Exercise 1: Find the vector equation of the line

$$\frac{x-3}{2} = \frac{y-7}{3} = \frac{z+5}{3}.$$

Exercise 2: Find the vector equation of the line

$$\frac{x}{2} = \frac{y+5}{-2} = \frac{z-6}{4}.$$

Exercise 3: Find the vector equation of the line

$$\frac{x-5}{-2} = \frac{y-1}{4} = \frac{z+1}{3}.$$

Practical No. 5

Problem

Find the directional derivatives of $\emptyset = 3x^2yz - 4y^2z^3$ in the direction of the vector $3\hat{\imath} - 4\hat{\jmath} + 2\hat{k}$ at the point (2,-1,3).

Theory

Given that

$$\emptyset = 3x^2yz - 4y^2z^3$$

and

$$\begin{aligned}
\vec{a} &= 3\hat{\imath} - 4\hat{\jmath} + 2\hat{k},\\
\hat{a} &= \frac{\vec{a}}{|\vec{a}|},\\
&= \frac{3\hat{\imath} - 4\hat{\jmath} + 2\hat{k}}{\sqrt{3^2 + (-4)^2 + 2^2}},\\
&= \frac{3\hat{\imath} - 4\hat{\jmath} + 2\hat{k}}{\sqrt{29}},\\
grad\emptyset &= \nabla\emptyset\\
&= (\hat{\imath}\frac{\partial}{\partial x} + \hat{\jmath}\frac{\partial}{\partial y} + \hat{k}\frac{\partial}{\partial z})(3x^2yz - 4y^2z^3)\\
&= \hat{\imath}(6xyz) + \hat{\jmath}(3x^2z - 8yz^3) + \hat{k}(3x^2y - 12y^2z^2)\\
\frac{\partial\emptyset}{\partial s} &= \frac{3\hat{\imath} - 4\hat{\jmath} + 2\hat{k}}{\sqrt{29}}.(\hat{\imath}(6xyz) + \hat{\jmath}(3x^2z - 8yz^3) + \hat{k}(3x^2y - 12y^2z^2)).\\
&= \frac{18xyz - 12x^2z + 32yz^3 + 6x^2y - 24y^2z^2}{\sqrt{29}}.
\end{aligned}$$

Directional derivative at point $(2, -1, 3)$ is $\frac{-1356}{\sqrt{29}}$.

Algorithm

Step 1: Calculate $\nabla\phi$ (gradient);

Step 2: Normalize $\vec{a}$;

Step 3: Compute the dot product $\nabla\phi \cdot \vec{a}$ at the given point.

Program

The following MATLAB script implements the algorithm.

```
syms x y z
phi = 3 * x^2 * y * z - 4 * y^2 * z^3;
grad_phi = gradient(phi, [x, y, z]);
point = [2, -1, 3];
grad_phi_at_point = subs(grad_phi, [x, y, z], point);
d = [3, -4, 2];
d_unit = d/norm(d);
```

$directional\_derivative = dot(grad\_phi\_at\_point, d\_unit);$
disp('The directional derivative of phi at the point (2, -1, 3) in the direction of [3, -4, 2] is:');
$disp(vpa(directional\_derivative, 4));$

Output

The output of the program is as follows:
The directional derivative of phi at the point (2, -1, 3) in the direction of [3, -4, 2] is:
-251.8

Exercise

Exercise 1: Find the directional derivatives of $\emptyset = x^2y + y^3$ in the direction of the vector $3\hat{i} + 4\hat{j}$ at the point (1,2).
Exercise 2: Find the directional derivatives of $\emptyset = x^2y - y^2$ in the direction of the vector $2\hat{i} - 4\hat{j}$ at the point (-1,2).
Exercise 3: Find the directional derivatives of $\emptyset = x^2yz + y^2 + xyz$ in the direction of the vector $\hat{i} + \hat{j} + \hat{k}$ at the point (1,2,-1).

Practical No. 6

Problem

Find k if the following pair of the circles are orthogonal

$$x^2 + y^2 + 2by - k = 0,$$

$$x^2 + y^2 + 2ax + 8 = 0.$$

Theory

General form of the equation of a circle is

$$x^2 + y^2 + 2h_1x + 2k_1y + c = 0$$

where (h_1, k_1) is the center of the circle, r_1 is the radius.
From the first circle

$$\begin{aligned} &x^2 + y^2 + 2by - k = 0\\ &\implies (h_1, k_1) = (0, b),\\ &c_1 = -k. \end{aligned}$$

From the second circle

$$\begin{aligned}&x^2 + y^2 + 2ax + 8 = 0\\ &\implies (h_2, k_2) = (a, 0),\\ &c_2 = 8.\end{aligned}$$

Condition for orthogonality of two circle

$$2h_1h_2 + 2k_1k_2 = c_1 + c_2.$$

Hence, the value of k is 8.

Algorithm

Step 1: Define symbolic variables a, b, and k;

Step 2: Use orthogonality condition and solve for k using `solve()`;

Step 3: Display the value of k.

Program

The following MATLAB script implements the algorithm.
syms a b k
$h_1 = 0;$
$k_1 = b;$
$c\_1 = -k;$
$h_2 = a;$
$k_2 = 0;$
$c\_2 = 8;$
$orthogonality\_condition = 2 * h1 * h2 + 2 * k1 * k2 == c\_1 + c\_2;$
$k\_value = solve(orthogonality\_condition, k);$
disp('The value of k for which the circles are orthogonal is:');
$disp(k\_value);$

Output

The output of the program is as follows:
The value of k for which the circles are orthogonal is:
8

Exercise

Exercise 1: Find k if the following pair of the circles are orthogonal

$$x^2 + y^2 + 4by - k = 0,$$

$$x^2 + y^2 + 8ax + 6 = 0.$$

Exercise 2: Find k if the following pair of the circles are orthogonal

$$x^2 + y^2 + 2by - 3k = 0,$$

$$x^2 + y^2 + 2ax + 16 = 0.$$

Exercise 3: Find k if the following pair of the circles are orthogonal

$$x^2 + y^2 + 2by - 5k = 0,$$

$$x^2 + y^2 + 2ax + 10 = 0.$$

Practical No. 7

Problem

Find the equation of the circle with center (2,3) and touching the line $3x - 4y + 1 = 0$.

Theory

Given the center $C = (2, 3)$ and radius r = Perpendicular distance from C to $Ax+By+C = 0$. Therefore,

$$r = \frac{|Ah + Bk + C|}{\sqrt{A^2 + B^2}}$$

$$= \frac{|3(2) - 4(3) + 1|}{\sqrt{3^2 + 4^2}} = 1$$

Equation of the circle

$$\begin{aligned}
&(x - h)^2 + (y - k)^2 = r^2 \\
&\implies (x - 2)^2 + (y - 3)^2 = 1 \\
&\implies x^2 + y^2 - 4x - 6y + 12 = 0.
\end{aligned}$$

Algorithm

Step 1: Calculate radius as the perpendicular distance from $(2, 3)$ to the line using formula;

Step 2: Substitute the center and radius into $(x - h)^2 + (y - k)^2 = r^2$;

Step 3: Display the equation.

Program

The following MATLAB script implements the algorithm.

```
syms x y
h = 2;
k = 3;
A = 3;
B = -4;
C = 1;
r = abs(A * h + B * k + C)/sqrt(A^2 + B^2);
disp('Radius of the circle:');
disp(r);
circle_eq = (x - h)^2 + (y - k)^2 - r^2;
disp('Equation of the circle:');
disp(circle_eq);
```

Output

The output of the program is as follows:

```
Radius of the circle:
1
Equation of the circle:
(x - 2)^2 + (y - 3)^2 - 1
```

Exercise

Exercise 1: Find the equation of the circle with center (1,-2) and touching the line $x + 2y - 4 = 0$.
Exercise 2: Find the equation of the circle with center (3,4) and touching the line $6x - 4y + 10 = 0$.
Exercise 3: Find the equation of the circle with center (-2,-3) and touching the line $-3x + 4y - 1 = 0$.

Practical No. 8

Problem

To find the shortest distance between the two skew lines given by

$$\frac{x-2}{3}=\frac{y+1}{2}=\frac{z-6}{2}$$

and

$$\frac{x-6}{3}=\frac{1-y}{2}=\frac{z+8}{0}$$

.

Theory

We use the formula

$$\text{Distance}=\frac{|(\vec{d_1}\times\vec{d_2})\cdot\vec{PQ}|}{|\vec{d_1}\times\vec{d_2}|},$$

For the first line

$$\frac{x-2}{3}=\frac{y+1}{2}=\frac{z-6}{2},$$

we can write

$$\vec{r_1}=(2,-1,6)+t(3,2,2),$$

So, the direction vector $\vec{d_1}$ of the first line is

$$\vec{d_1}=(3,2,2),$$

$$P=(2,-1,6).$$

For the second line

$$\frac{x-6}{3}=\frac{1-y}{2}=\frac{z+8}{0},$$

we can write

$$\vec{r_2}=(6,1,-8)+s(3,-2,0).$$

So, the direction vector $\vec{d_2}$ of the second line is

$$\begin{aligned}&\vec{d_2}=(3,-2,0),\\&Q=(6,1,-8),\\&\vec{PQ}=(6-2,1+1,-8-6)=(4,2,-14).\end{aligned}$$

Using the formula for the cross product

$$\vec{d_1}\times\vec{d_2}=\begin{vmatrix}\hat{i}&\hat{j}&\hat{k}\\3&2&2\\3&-2&0\end{vmatrix}$$

$$=\hat{i}(4)-\hat{j}(-6)+\hat{k}(-12)$$

$$|\vec{d_1} \times \vec{d_2}| = \sqrt{4^2 + 6^2 + (-12)^2}$$

$$|\vec{d_1} \times \vec{d_2}| = \sqrt{16 + 36 + 144} = \sqrt{196} = 14.$$

Now, we find the dot product $(\vec{d_1} \times \vec{d_2}) \cdot \vec{PQ}$:

$$\begin{aligned}(\vec{d_1} \times \vec{d_2}) \cdot \vec{PQ} &= (4, 6, -12) \cdot (4, 2, -14) \\ &= (4 \cdot 4) + (6 \cdot 2) + (-12 \cdot -14) \\ &= 16 + 12 + 168 = 196.\end{aligned}$$

Finally, the shortest distance between the two lines is:

$$\text{Distance} = \frac{|(\vec{d_1} \times \vec{d_2}) \cdot \vec{PQ}|}{|\vec{d_1} \times \vec{d_2}|} = \frac{|196|}{14} = 14.$$

Algorithm

Step 1: Define direction vectors and points for each line;

Step 2: Calculate cross product of direction vectors;

Step 3: Find distance using formula involving cross product and vector joining points;

Step 4: Display distance.

Program

The following MATLAB script implements the algorithm.

```
d1 = [3, 2, 2];
d2 = [3, −2, 0];
P = [2, −1, 6];
Q = [6, 1, −8];
PQ = Q − P;
cross_d1_d2 = cross(d1, d2);
magnitude_cross_d1_d2 = norm(cross_d1_d2);
dot_product = dot(cross_d1_d2, PQ);
distance = abs(dot_product)/magnitude_cross_d1_d2;
fprintf('The shortest distance between the two lines is: %.2f', distance);
```

Output

The output of the program is as follows:

```
The shortest distance between the two lines is: 14.00
```

Exercise

Exercise 1: To find the shortest distance between the two skew lines given by

$$\frac{x-1}{4} = \frac{y-2}{-2} = \frac{z-3}{1}$$

and

$$\frac{x-2}{3} = \frac{y-1}{1} = \frac{z-5}{-4}$$

.

Exercise 2: To find the shortest distance between the two skew lines given by

$$\frac{x+2}{-3} = \frac{y-3}{-1} = \frac{z+6}{-4}$$

and

$$\frac{x+6}{3} = \frac{1-y}{-2} = \frac{2-z}{4}$$

.

Exercise 3: To find the shortest distance between the two skew lines given by

$$\frac{-x+2}{-2} = \frac{-y+1}{-1} = \frac{z-1}{5}$$

and

$$\frac{-x-6}{-3} = \frac{1+y}{3} = \frac{2z+8}{4}$$

.

Practical No. 9

Problem

What is the moment about the point $\hat{\imath} + 2\hat{\jmath} - \hat{k}$ of a force represented by $3\hat{\imath} + \hat{k}$ acting through the point $2\hat{\imath} - \hat{\jmath} + 3\hat{k}$?

Theory

The moment $\vec{M}$ of a force $\vec{F}$ about a point with position vector $\vec{r_0}$ is given by

$$\vec{M} = \vec{r} \times \vec{F}$$

In this problem

$$\vec{r_0} = \vec{\imath} + 2\vec{\jmath} - \vec{k}.$$
$$\vec{r_p} = 2\vec{\imath} - \vec{\jmath} + 3\vec{k}.$$
$$\vec{F} = 3\vec{\imath} + \vec{k}.$$

where $\vec{r} = \vec{r}_p - \vec{r}_0$.

Therefore, $\vec{r} = (2\vec{i} - \vec{j} + 3\vec{k}) - (\vec{i} + 2\vec{j} - \vec{k}) = \vec{i} - 3\vec{j} + 4\vec{k}$, $\vec{M} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ 1 & -3 & 4 \\ 3 & 0 & 1 \end{vmatrix} \implies \vec{M} = -3\vec{i} + 11\vec{j} + 9\vec{k}$.

Algorithm

Step 1: Calculate the position vector between the two points;

Step 2: Compute the cross product with the force vector to find the moment.

Program

The following MATLAB script implements the algorithm.

```
r0 = [1; 2; −1];
rp = [2; −1; 3];
F = [3; 0; 1];
r = rp − r0;
M = cross(r, F);
disp('The moment of the force about the point is:');
disp(M);
```

Output

The output of the program is as follows:

```
The moment of the force about the point is:
[−3, 11, 9]
```

Exercise

Exercise 1: What is the moment about the point $3\hat{i} + 2\hat{j} + \hat{k}$ of a force represented by $2\hat{i} + \hat{k}$ acting through the point $\hat{i} + \hat{j} + \hat{k}$?
Exercise 2: What is the moment about the point $2\hat{i} + 3\hat{j} - 5\hat{k}$ of a force represented by $7\hat{i} + 11\hat{k}$ acting through the point $13\hat{i} - 17\hat{j} + 19\hat{k}$?
Exercise 3: What is the moment about the point $4\hat{i} + 9\hat{j} - 16\hat{k}$ of a force represented by $25\hat{i} + \hat{k}$ acting through the point $2\hat{i} - \hat{j} + 3\hat{k}$?

Practical No. 10

Problem

Write a unit vector in the direction of the sum of the vector $\vec{a} = 2\hat{\imath} - 5\hat{k}$ and $\vec{b} = 2\hat{\imath} - 4\hat{\jmath} + 5\hat{k}$.

Theory

Given vectors

$$\vec{a} = 2\hat{\imath} - 5\hat{k},$$
$$\vec{b} = 2\hat{\imath} - 4\hat{\jmath} + 5\hat{k}.$$

Let the sum of the vectors is

$$\vec{c} = (2\hat{\imath} - 5\hat{k}) + (2\hat{\imath} - 4\hat{\jmath} + 5\hat{k}) = 4\hat{\imath} - 4\hat{\jmath}.$$

The unit vector is given by

$$\begin{aligned}\hat{c} &= \frac{\vec{c}}{|\vec{c}|}\\ &= \frac{4\hat{\imath} - 4\hat{\jmath}}{|4\hat{\imath} - 4\hat{\jmath}|}\\ &= \frac{1}{\sqrt{2}}\hat{\imath} - \frac{1}{\sqrt{2}}\hat{\jmath}.\end{aligned}$$

Algorithm

Step 1: Add vectors $\vec{a}$ and $\vec{b}$ to get $\vec{c}$;

Step 2: Calculate the magnitude of $\vec{c}$ and normalize it.

Program

The following MATLAB script implements the algorithm.

```
a = [2; 0; −5];
b = [2; −4; 5];
sum_vector = a + b;
magnitude = norm(sum_vector);
unit_vector = sum_vector/magnitude;
disp('The unit vector in the direction of the sum of a and b is:');
disp(unit_vector);
```

Output

The output of the program is as follows:

```
The unit vector in the direction of the sum of a and b is:
[0.7071, −0.7071, 0]
```

Exercise

Exercise 1: Write a unit vector in the direction of the sum of the vector $\vec{a} = 3\hat{\imath} + 2\hat{\jmath} - \hat{k}$ and $\vec{b} = 2\hat{\imath} + \hat{\jmath} + \hat{k}$.
Exercise 2: Write a unit vector in the direction of the sum of the vector $\vec{a} = 3\hat{\imath} + \hat{k}$ and $\vec{b} = 3\hat{\imath} - 2\hat{\jmath} + 5\hat{k}$.
Exercise 3: Write a unit vector in the direction of the sum of the vector $\vec{a} = \hat{\imath} + \hat{\jmath} + \hat{k}$ and $\vec{b} = \hat{\imath} - \hat{\jmath} + \hat{k}$.

Practical No. 11

Problem

Find the equation of the line whose distance from the origin is 5 units and the angle which the line makes with the positive x-axis is 120°.

Theory

The equation of a line in the *normal form* is given by

$$x\cos\theta + y\sin\theta = d$$

where d is the perpendicular distance from the origin to the line, θ is the angle that the perpendicular to the line makes with the positive $x - axis$.

Given

$$d = 5, \quad \theta = 120^\circ,$$

substituting these values into the equation, we have

$$x\cos 120^\circ + y\sin 120^\circ = 5.$$

Thus, the equation becomes:

$$\begin{aligned} x\left(-\frac{1}{2}\right) + y\left(\frac{\sqrt{3}}{2}\right) &= 5 \\ -\frac{1}{2}x + \frac{\sqrt{3}}{2}y &= 5, \\ -x + \sqrt{3}\,y &= 10. \end{aligned}$$

Therefore, the equation of the line is

$$x - \sqrt{3}\,y = -10.$$

Algorithm

Step 1: Define the distance $d = 5$ and angle $\theta = 120°$;

Step 1: Use the formula $x \cos\theta + y \sin\theta = d$ to write the line equation;

Step 2: Simplify to obtain the final form of the equation.

Program

```
close all
clear all
clc
d = 5;
theta = 120;
% Convert angle to radians for trigonometric functions
theta_rad = deg2rad(theta);
A = cos(theta_rad);
B = sin(theta_rad);
C = -d;
% range for x values to plot the line
x = -15:0.1:15;
% Calculate y values using the line equation
y = (d - A * x) / B;
fprintf('The line equation is: %.2fx + %.2fy = %.2f\n', A, B, d);
% Plot the line
plot(x, y, 'b', 'LineWidth', 2);
hold on;
% Plot the origin point
plot(0, 0, 'ro', 'MarkerSize', 8, 'MarkerFaceColor', 'r');
text(0, 0, '  Origin', 'VerticalAlignment', 'top', 'HorizontalAlignment', 'left');
x_perpendicular = d * cos(theta_rad);
y_perpendicular = d * sin(theta_rad);
plot([0 x_perpendicular], [0 y_perpendicular], 'k--', 'LineWidth', 1.5);
text(x_perpendicular,y_perpendicular,sprintf('%.1f units',d),'VerticalAlignment','bottom');
% Add labels, title, and grid
xlabel('x');
ylabel('y');
title(sprintf('Line with Distance %.1f Units from Origin at %.1f with+Ve X-axis',d,theta));
grid on;
axis equal;
hold off;
```

Output

```
    The line equation is: -0.50x + 0.87y = 5.00
```

Exercise

Exercise 1: Find the equation of the line whose distance from the origin is 7 units and the angle which the line makes with the positive x-axis is 45°.
Exercise 2: Find the equation of the line whose distance from the origin is 5 units and the angle which the line makes with the positive x-axis is 150°.
Exercise 3: Find the equation of the line whose distance from the origin is 10 units and the angle which the line makes with the positive x-axis is 180°.

Practical No. 12

Problem

Find another polar representation of the point $(r, \theta) = \left(2, \frac{\pi}{3}\right)$ with $r < 0$.

Theory

In polar coordinates, a point (r, θ) can be represented in an alternative form by using the transformations

$$r \to -r \quad \text{and} \quad \theta \to \theta + \pi.$$

Given

$$(r, \theta) = \left(2, \frac{\pi}{3}\right).$$

We want to find an equivalent representation with $r < 0$. Using the transformations above

$$r = -2 \quad \text{and} \quad \theta = \frac{\pi}{3} + \pi.$$

Thus, the polar representation of the point with $r < 0$ is

$$(r, \theta) = \left(-2, \frac{4\pi}{3}\right).$$

Algorithm

Step 1: Convert the given coordinates using $r \to -r$ and $\theta \to \theta + \pi$;

Step 2: Write the new polar representation as $(r, \theta) = (-2, \frac{4\pi}{3})$.

Program

```
r = 2;
theta = pi/3;
% Find the equivalent polar coordinates with r < 0
r_neg = -r;
theta_neg = theta + pi;
fprintf('The same coordinate with r<0 is :');
fprintf('(%.2f,%.2f radians)\n',r_neg,theta_neg);
```

Output

```
The same coordinate with r<0 is : (-2.00,4.19 radians)
```

Exercise

Exercise 1: Find another polar representation of the point $(r, \theta) = \left(1, \frac{\pi}{4}\right)$ with $r < 0$.
Exercise 2: Find another polar representation of the point $(r, \theta) = \left(3, \frac{\pi}{6}\right)$ with $r < 0$.
Exercise 3: Find another polar representation of the point $(r, \theta) = \left(4, \frac{3\pi}{12}\right)$ with $r < 0$.

Practical No. 13

Problem

Find the asymptotes of the function

$$y = \frac{x^2 + 4x + 7}{x - 1}$$

(vertical, horizontal, or slant).

Theory

To find the asymptotes of the function

$$y = \frac{x^2 + 4x + 7}{x - 1},$$

we write in the form

$$y(x - 1) = x^2 + 4x + 7.$$

We analyze the following types of asymptotes
Vertical Asymptote: For a vertical asymptote equate to zero the coefficient of highest power of y in the equation,provided this is not merely a constant .

For this equation, $x - 1 = 0$ when $x = 1$. Therefore, there is a vertical asymptote at

$$x = 1.$$

Horizontal: For a horizontal asymptote equate to zero the coefficient of highest power of x in the equation,provided this is not merely a constant .

For this equation, the coefficient of highest power of x is constant. Therefore, there is no horizontal asymptote.

Slant: Put $x = 1$ and $y = m$ in second degree term, let the second degree term is

$$h_2(m) = 1 - m$$

and the zero's of $h_2(m)$ is $m = 1$.

Put $x = 1$ and $y = m$ in first degree term, let the first degree term is

$$h_1(m) = 4 + m.$$

Now, we find the value of C.

$$C = -\frac{h_1(m)}{h_2'(m)}$$

$$C = \frac{4 + m}{1}$$

at $m = 1$, C is 5. so the slant asymptote is:

$$y = x + 5$$

Vertical asymptote: $x = 1$
Slant asymptote: $y = x + 5$

Algorithm

Step 1: For the vertical asymptote, set the denominator to zero and solve for x;

Step 2: For the slant asymptote, perform polynomial division of the numerator by the denominator;

Step 3: Analyze results to identify asymptotes.

Program

```
close all
clear all
clc
% Step 1: Define variable and the function
syms x;
f = (x^2 + 4*x + 7) / (x - 1);
```

```
% Step 2: Find the vertical asymptote
[~, denom] = numden(f); % Separate numerator and denominator
vertical_asymptote = solve(denom == 0, x); % Solve for x where denominator is zero

% Step 3: Find the slant asymptote by polynomial division
[num, denom] = numden(f); % Separate numerator and denominator again for clarity
[quotient, remainder] = quorem(num, denom, x); % Polynomial division

% Display results
disp('Vertical Asymptote at x = ');
disp(vertical_asymptote);

disp('Slant Asymptote: y = ');
disp(quotient); % The quotient is the slant asymptote

% For plotting
fplot(f, [-10, 10], 'DisplayName', 'f(x)')
hold on
fplot(quotient, [-10, 10], '--r', 'DisplayName', 'Slant Asymptote')
xline(vertical_asymptote, '--k', 'DisplayName', 'Vertical Asymptote')
legend show
grid on
title('Graph of the Function and Its Asymptotes')
hold off
```

Output

```
Vertical Asymptote at x = 1
Slant Asymptote: y = x + 5
```

Exercise

Exercise 1: Find the asymptotes of the function

$$y = \frac{x^2 + 2x + 3}{x - 2}$$

(vertical, horizontal, or slant).
Exercise 2: Find the asymptotes of the function

$$y = \frac{x^2 + 9x + 20}{x + 4}$$

(vertical, horizontal, or slant).
Exercise 3: Find the asymptotes of the function

$$y = \frac{x^2 - 2x + 1}{x + 1}$$

(vertical, horizontal, or slant).

Practical No. 14

Problem

Plot the ellipsoid

$$\frac{x^2}{25} + \frac{y^2}{81} + \frac{z^2}{16} = 1.$$

Theory

The given equation represents an ellipsoid

$$\frac{x^2}{25} + \frac{y^2}{81} + \frac{z^2}{16} = 1.$$

To understand and plot this ellipsoid, we analyze its structure. This equation is in the standard form of an ellipsoid

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1,$$

where
$a^2 = 25 \Rightarrow a = 5$,
$b^2 = 81 \Rightarrow b = 9$,
$c^2 = 16 \Rightarrow c = 4$.
Thus, the ellipsoid has:
Semi-axis length 5 along the x-axis,
Semi-axis length 9 along the y-axis,
Semi-axis length 4 along the z-axis.

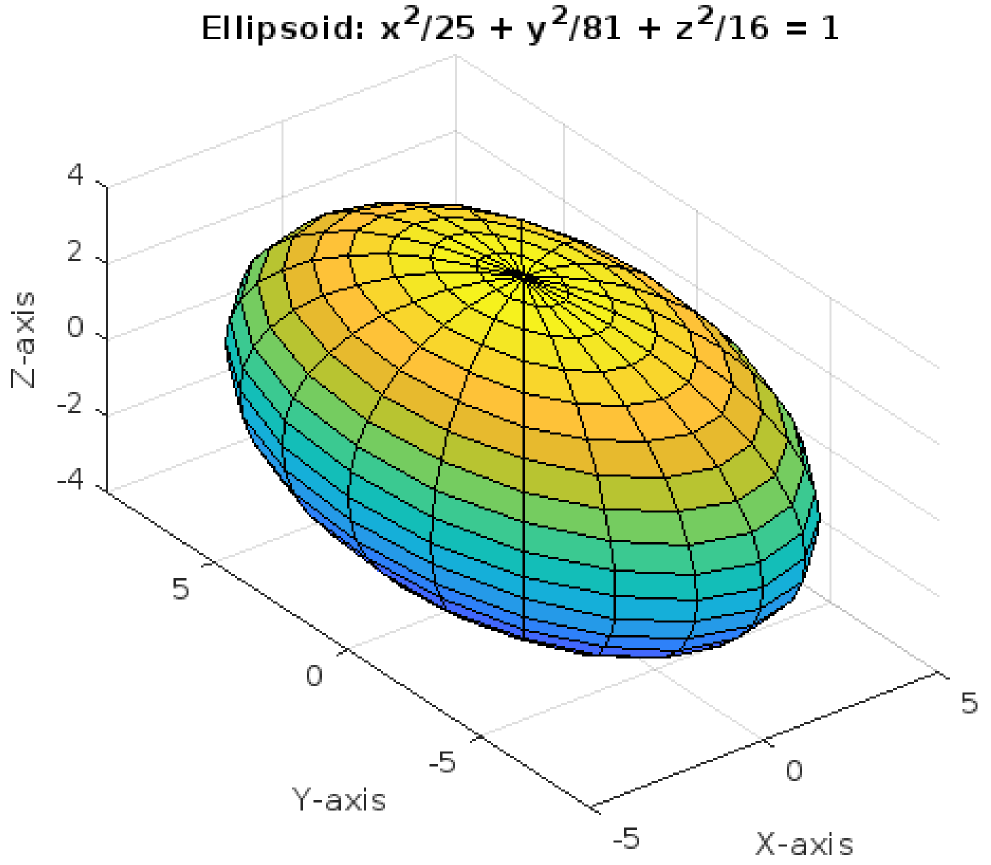
Algorithm

Step 1: Define semi-axis lengths based on the coefficients in the ellipsoid equation;

Step 2: Plot the ellipsoid using the defined axis lengths.

Program

```
[X, Y, Z] = ellipsoid(0, 0, 0, 5, 9, 4);
surf(X, Y, Z);
xlabel('X-axis');
ylabel('Y-axis');
zlabel('Z-axis');
title('Ellipsoid: x^2/25 + y^2/81 + z^2/16 = 1');
axis equal;
```



Output

```
An ellipsoid image.
```

Exercise

Exercise 1: Plot the ellipsoid

$$\frac{x^2}{64} + \frac{y^2}{49} + \frac{z^2}{36} = 1.$$

Exercise 2: Plot the ellipsoid

$$\frac{4x^2}{25} + \frac{9y^2}{100} + \frac{z^2}{225} = 1.$$

Exercise 3: Plot the ellipsoid

$$\frac{x^2}{121} + \frac{25y^2}{144} + \frac{z^2}{361} = 1.$$

Practical No. 15

Problem

Plot the surface and identify its type for the equation

$$x^2 + y^2 + z^2 - 4xy - 4xz - 4yz + 24 = 0.$$

Theory

To identify the type of surface represented by the equation

$$x^2 + y^2 + z^2 - 4xy - 4xz - 4yz + 24 = 0.$$

We proceed by analyzing and simplifying the equation using linear algebra techniques.

1. **Rewrite the equation in quadratic form:** This equation can be represented in the form
$$\mathbf{x}^T A\mathbf{x} + \mathbf{b}^T\mathbf{x} + c = 0,$$
where $\mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$, and A is the symmetric matrix associated with the quadratic terms.
From the given equation, we can identify A as follows:
$$A = \begin{pmatrix} 1 & -2 & -2 \\ -2 & 1 & -2 \\ -2 & -2 & 1 \end{pmatrix}.$$

2. **Determine the eigenvalues of A:** The type of surface can be identified by examining the eigenvalues of A. We calculate the eigenvalues by solving $\det(A - \lambda I) = 0$.
After computing, we find that the eigenvalues are
$$\lambda_1 = -3, \quad \lambda_2 = 3, \quad \lambda_3 = 3$$
Since, there is one negative and two positive eigenvalues, the surface is a **hyperboloid of one sheet**.

Algorithm

Step 1: Rewrite the equation in matrix form and determine the symmetric matrix;

Step 2: Compute the eigenvalues of the matrix to identify the type of surface;

Step 3: Plot the surface based on the identified type (hyperbolic).

Program

```
close all
clear all
clc
syms x y z;
eq = x^2 + y^2 + z^2 - 4*x*y - 4*x*z - 4*y*z + 24 == 0;
A = [1, -2, -2;
    -2, 1, -2;
    -2, -2, 1];

% Calculate the eigenvalues
eigenvalues = eig(A);
% Check if all eigenvalues are real and non-zero
isHyperbolic = isreal(eigenvalues) && all(eigenvalues ~= 0);
disp('Eigenvalues are:');
disp(eigenvalues);
if isHyperbolic
    disp('The surface is hyperbolic.');
else
    disp('The matrix A is not hyperbolic.');
end
% Convert symbolic expression to a function for plotting
fimplicit3(eq, [-10 10 -10 10 -10 10]);
xlabel('X'); ylabel('Y'); zlabel('Z');
title('Surface: x^2 + y^2 + z^2 - 4xy - 4xz - 4yz + 24 = 0');
axis equal;
```

Output

```
    Eigenvalues are:
    -3.0000
    3.0000
    3.0000
    The surface is hyperbolic.
```

Exercise

Exercise 1: Plot the surface and identify its type for the equation

$$x^2 + y^2 + 2z^2 + 2xy = 0.$$

Exercise 2: Plot the surface and identify its type for the equation

$$x^2 + y^2 + z^2 - 2xy - 2xz - 2yz + 12 = 0.$$

Exercise 3: Plot the surface and identify its type for the equation

$$2x^2 + 3y^2 + z^2 - 2xy - 2xz - 4yz + 24 = 0.$$

Practical No. 16

Problem

A surface is define as :

$$z = x^2 + y^2.$$

Find the equation of tangent plane to this surface at point $(1, 2, 5)$.

Theory

The equation of tangent plane to the surface $z = f(x, y)$, at point $(x_0, y_0, z_0$ define as,

$$z = f(x_0, y_0) + f_x(x_0, y_0)(x - x_0) + f_y(x_0, y_0)(y - y_0) \tag{9.1}$$

So, we have to find out the values of $f(x_0, y_0), f_x(x_0, y_0)$ and $f_y(x_0, y_0)$, where $x_0 = 1$ and $y_0 = 2$, and substituting these values in equation (1). Therefore, we get equation of tangent plane, i.e.,

$$z = 2x + 4y - 5.$$

Algorithm

Step 1: Calculate partial derivatives of $z = f(x, y)$ with respect to x and y;

Step 2: Substitute the point (x_0, y_0, z_0) and use the tangent plane formula.

Program

```
close all
clear all
clc
x0 = 1;
y0 = 2;
z0 = x0^2 + y0^2;

% Define the partial derivatives of z = x^2 + y^2
fx = 2 * x0;
fy = 2 * y0;
syms x y z
z_tangent = z0 + fx * (x - x0) + fy * (y - y0);
% Display the tangent plane equation
fprintf('Plane is: z = %.2f + %.2f*(x - %.2f) + %.2f*(y - %.2f)\n', z0, fx, x0, fy, y0);
fprintf('Plane: z = %.2f*x + %.2f*y + %.2f\n', fx, fy, (z0 - fx*x0 - fy*y0));
%define the surface
eq =  x^2 + y^2 - z;

% define the plane
eq1 = 2*x +4*y -z -5;

% Plot the surface in blue
fimplicit3(eq==0,[-5 5 -5 5 -5 8],'FaceAlpha',0.5,'EdgeColor','none','FaceColor','b');
xlabel('X'); ylabel('Y'); zlabel('Z');
title('Surface z = x^2 + y^2 and Tangent Plane at (1, 2, 5)');
axis equal;
hold on;

% Plot the tangent plane in green
fimplicit3(eq1==0,[-5 5 -5 5 -5 8],'FaceAlpha',0.5,'EdgeColor','none','FaceColor','g');
xlabel('X'); ylabel('Y'); zlabel('Z');
axis equal;
hold on;

% Mark the point of tangency in red
x0 = 1; y0 = 2; z0 = x0^2 + y0^2; % Point (1, 2, 5)
plot3(x0, y0, z0, 'ro', 'MarkerSize', 10, 'MarkerFaceColor', 'r')

% Add labels and legend
xlabel('x');
ylabel('y');
zlabel('z');
title('Surface z = x^2 + y^2 and Tangent Plane at (1, 2, 5)');
```

```
legend('Surface z = x^2 + y^2 (Blue)', 'Tangent Plane (Green)', 'Point of Tangenc
grid on
hold off
```

Output

```
    Plane is: z = 5.00 + 2.00*(x - 1.00) + 4.00*(y - 2.00)
    Plane: z = 2.00*x + 4.00*y + -5.00
```

Exercise

Exercise 1: A surface is define as :

$$z = x^2 - y^2.$$

Find the equation of tangent plane to this surface at point $(2, 1, 5)$.
Exercise 2: A surface is define as :

$$z = 2x^2 + 4y^2.$$

Find the equation of tangent plane to this surface at point $(3, 2, 4)$.
Exercise 3: A surface is define as :

$$z = x^2 - 2y^2.$$

Find the equation of tangent plane to this surface at point $(-1, -2, 0)$.

Practical No. 17

Problem-17

Find the equation of the cone whose vertex is at the origin and passes through the curve

$$\frac{x^2}{4} + \frac{y^2}{9} + \frac{z^2}{1} = 1, x + y + z = 1.$$

Theory

General form of a line through the origin. Any line passing through $(0, 0, 0)$ can be written as

$$\frac{x}{l} = \frac{y}{m} = \frac{z}{n}.$$

Let Coordinates of a point P on the line.The coordinates of any point P on this line can be represented as (lr, mr, nr), where r is a arbitrary constant. If this line intersects the given

curve, then the coordinates of P must satisfy the equation of the curve.
Substitute $x = lr$, $y = mr$, and $z = nr$ into

$$\frac{x^2}{4} + \frac{y^2}{9} + \frac{z^2}{1} = 1$$

leads to

$$\frac{l^2r^2}{4} + \frac{m^2r^2}{9} + n^2r^2 = 1.$$

Dividing by r^2, we get

$$\frac{l^2}{4} + \frac{m^2}{9} + n^2 = (l + m + n)^2.$$

After simplifying further, we obtain the required equation of the cone

$$27l^2 + 32m^2 + 72(lm + mn + nl) = 0.$$

Thus, the equation of the cone required is

$$27x^2 + 32y^2 + 72(xy + yz + zx) = 0.$$

Algorithm

Step 1: Define symbolic variables x, y, z, l, m, and n in MATLAB;

Step 2: Represent any point P on the cone as (lr, mr, nr);

Step 3: Substitute coordinates (lr, mr, nr) into the curve equation $\frac{x^2}{4} + \frac{y^2}{9} + z^2 = 1$;

Step 4: Simplify the equation by dividing by r^2;

Step 5: Derive the required equation of the cone;

Step 6: Display the result.

Program

```
syms x y z l m n r

% Define coordinates of the point P on the line
x = l * r;
y = m * r;
z = n * r;

% Substitute into the curve equation x^2/4 + y^2/9 + z^2 = 1
curve_eq = (x^2)/4 + (y^2)/9 + z^2 == 1;

% Substitute x = lr, y = mr, z = nr
```

```
curve_eq_sub = subs(curve_eq, {x, y, z}, {l*r, m*r, n*r});

curve_eq_simplified = simplify(curve_eq_sub / r^2);


cone_eq = 27 * l^2 + 32 * m^2 + 72 * (l * m + m * n + n * l);


disp('The equation of the cone is:')
disp(' 27 * x^2 + 32 * y^2 + 72 * (x * y + y * z + z * x)')
```

Output

```
The equation of the cone is:
27 * x^2 + 32 * y^2 + 72 * (x * y + y * z + z * x)
```

Exercise

Exercise 1: Find the equation of the cone whose vertex is at the origin and passes through the curve

$$\frac{x^2}{5}+\frac{y^2}{7}+\frac{z^2}{1}=1, x+y+z=1.$$

Exercise 2: Find the equation of the cone whose vertex is at the origin and passes through the curve

$$\frac{2x^2}{3}+\frac{y^2}{9}+\frac{z^2}{4}=1, x-y+z=1.$$

Exercise 3: Find the equation of the cone whose vertex is at the origin and passes through the curve

$$\frac{x^2}{9}+\frac{y^2}{16}+\frac{z^2}{25}=1, x+y+z=3.$$

Practical No. 18

Problem

Trace the curve $y^2(5-x)=x^2(5+x)$.

Theory

(i) **Symmetry:** The curve is not symmetrical about the x-axis.

(ii) **Origin:** The curve passes through the origin, and the tangents at the origin are given by:

$$y^2 = x^2 \Rightarrow y = x \quad \text{and} \quad y = -x.$$

Thus, the origin is a *node*.

(iii) **Asymptotes:** The curve has asymptotes at $x = 5$ and $x = -5$.

(iv) **Points of Intersection with Axes:**

- When $x = 0$, we get $y = 0$.
- When $y = 0$, the equation becomes:

$$x^2(5 + x) = 0.$$

  Solving for x, we find:

$$x = 0 \quad \text{or} \quad x = -5.$$

Therefore, the curve crosses the axes at $(0, 0)$ and $(-5, 0)$.

(v) **Solving for y in terms of x:**

$$y^2 = \frac{x^2(5 + x)}{5 - x}$$

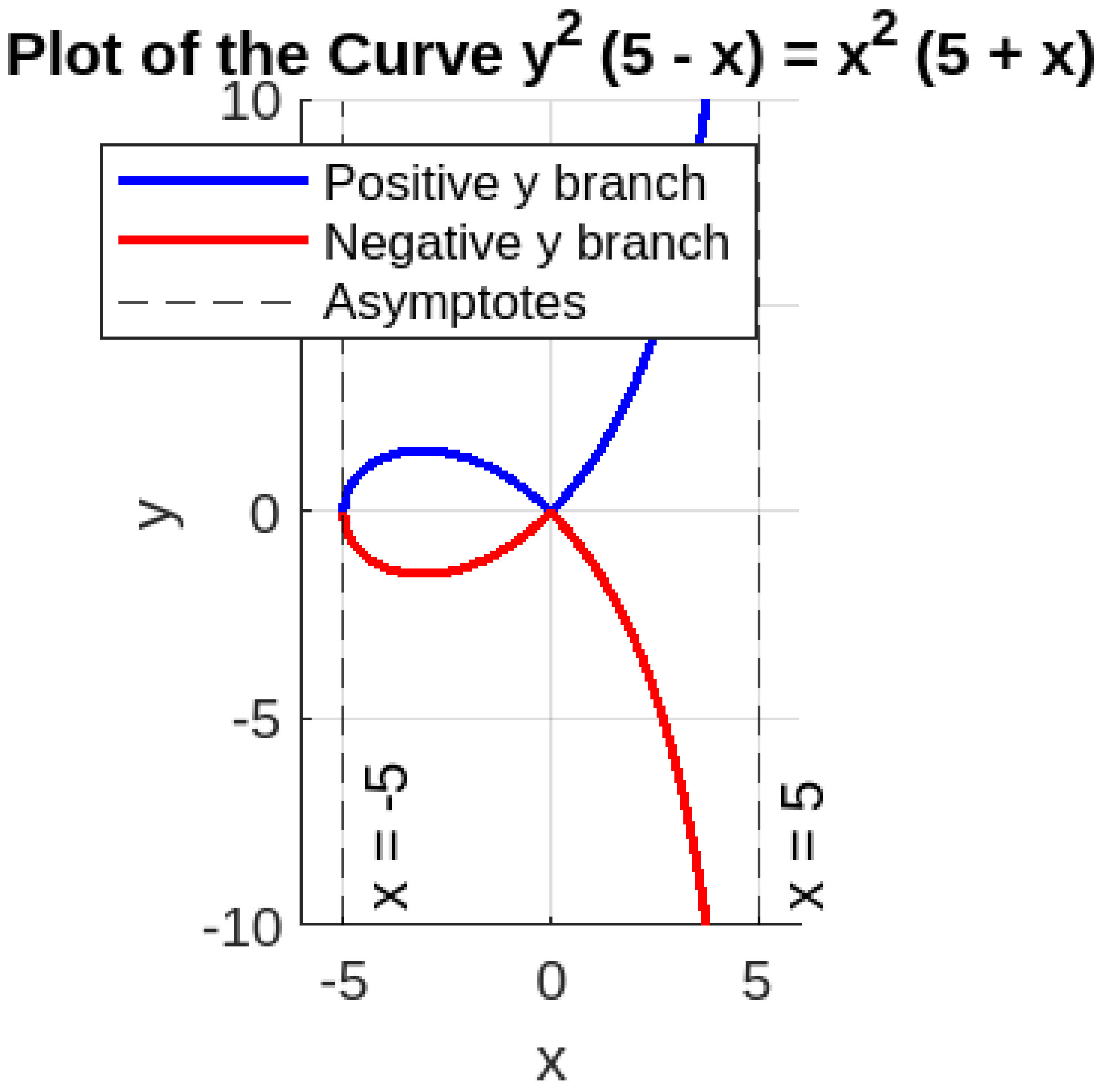
or

$$y = \pm x\sqrt{\frac{5 + x}{5 - x}}.$$

Behavior of y for values of x:

- When $x > 5$ or $x < -5$, y becomes imaginary because the term $\frac{5+x}{5-x}$ becomes negative.
  Therefore, no portion of the curve lies to the right of the line $x = 5$ or to the left of the line $x = -5$.

Hence the shape of the curve is shown in Fig. Hence the curve know as **Strophoid**



Algorithm

Step 1: Define x in a specified range using `linspace` in MATLAB;

Step 2: Compute the corresponding y values using the curve equation;

Step 3: Identify positive and negative branches of y;

Step 4: Plot both branches and add asymptotes at $x = \pm 5$;

Step 5: Label axes and display the final plot.

Program

```
x = linspace(-5, 5, 1000);
% Calculate y values using the curve equation:
% y^2 = (x^2 * (5 + x)) / (5 - x)
% y can be positive or negative, so we take both +sqrt and -sqrt
% Initialize y arrays to hold real parts of y values
y_positive = NaN(size(x));
y_negative = NaN(size(x));
for i = 1:length(x)
    % Check to ensure that the denominator (5 - x) is positive to get real y values
    if (5 - x(i)) > 0
        y_positive(i) = sqrt((x(i)^2 * (5 + x(i))) / (5 - x(i)));
        y_negative(i) = -sqrt((x(i)^2 * (5 + x(i))) / (5 - x(i)));
    end
end

% Plot the curve
figure;
hold on;
plot(x, y_positive, 'b', 'LineWidth', 1.5); % Plot positive branch
plot(x, y_negative, 'r', 'LineWidth', 1.5); % Plot negative branch
xlabel('x');
ylabel('y');
title('Plot of the Curve y^2 (5 - x) = x^2 (5 + x)');
grid on;
axis equal;
xlim([-6, 6]); % Extend x-axis slightly beyond the asymptotes for clarity
ylim([-10, 10]);

% Add asymptotes
xline(5, '--k', 'x = 5', 'LabelVerticalAlignment', 'bottom');
xline(-5, '--k', 'x = -5', 'LabelVerticalAlignment', 'bottom');

legend('Positive y branch', 'Negative y branch', 'Asymptotes');
hold off;
```

Output

Figure of Strophoid

Exercise

Exercise 1: Trace the curve $x^{2/3} + y^{2/3} = a^{2/3}$.
Exercise 2: Trace the curve $x^2y^2 = a^2(y^2 - x^2)$.
Exercise 3: Trace the curve $ay^2 = x^2(a - x)$, where $a > 0$.

Practical No. 19

Problem

Find the angle between the planes:

$$3x - 2y + 6z = 7$$

and

$$2x + y - 2z = 3.$$

Theory

For the plane $3x - 2y + 6z = 7$ the normal vector is

$$\vec{n}_1 = \langle 3, -2, 6 \rangle.$$

For the plane $2x + y - 2z = 3$ the normal is

$$\vec{n}_2 = \langle 2, 1, -2 \rangle.$$

Calculate the dot product of $\vec{n}_1$ and $\vec{n}_2$ as,

$$\vec{n}_1 \cdot \vec{n}_2 = (3)(2) + (-2)(1) + (6)(-2) = 6 - 2 - 12 = -8.$$

The magnitudes of $\vec{n}_1$ **and** $\vec{n}_2$ are 7 and 3, respectively. Calculate θ:

$$\cos\theta = \frac{\vec{n}_1 \cdot \vec{n}_2}{|\vec{n}_1||\vec{n}_2|} = \frac{-8}{7 \cdot 3} = \frac{-8}{21},$$
$$\theta = \cos^{-1}\left(\frac{-8}{21}\right).$$
$$\theta \approx \cos^{-1}(-0.381) \approx 112.62^\circ.$$

Algorithm

Step 1: Define the normal vectors $\vec{n}_1 = [3, -2, 6]$ and $\vec{n}_2 = [2, 1, -2]$ in MATLAB;

Step 2: Calculate the dot product `dot_product = dot(n1, n2)`;

Step 3: Compute the magnitudes of $\vec{n}_1$ and $\vec{n}_2$ using `norm()`;

Step 4: Calculate the angle θ as $\cos^{-1}\left(\frac{\text{dot\_product}}{|\vec{n}_1||\vec{n}_2|}\right)$;

Step 5: Display the angle in degrees.

Program

```
close all
clear all
clc
% Define the normal vectors for the two planes
n1 = [3, -2, 6];
n2 = [2, 1, -2];

dot_product = dot(n1, n2);

magnitude_n1 = norm(n1);
magnitude_n2 = norm(n2);
cos_theta = dot_product / (magnitude_n1 * magnitude_n2);

theta = acosd(cos_theta);
fprintf('The angle between the planes is approximately %.2f degrees.\n', theta);

% Define the grid for plotting the planes
[x, y] = meshgrid(-10:1:10, -10:1:10);

z1 = (7 - 3*x + 2*y) / 6;
z2 = (3 - 2*x - y) / -2;

% Plot the planes
figure;
hold on;

% Plot the first plane in blue
surf(x, y, z1, 'FaceColor', 'cyan', 'FaceAlpha', 0.5);

% Plot the second plane in red
surf(x, y, z2, 'FaceColor', 'magenta', 'FaceAlpha', 0.5);

% Plot the normal vectors
quiver3(0, 0, 0, n1(1), n1(2), n1(3), 'k', 'LineWidth', 2, 'MaxHeadSize', 0.5');
quiver3(0, 0, 0, n2(1), n2(2), n2(3), 'g', 'LineWidth', 2, 'MaxHeadSize', 0.5');

% Label the axes
xlabel('X');
ylabel('Y');
zlabel('Z');

% Set the title with the angle between the planes
title(sprintf('Angle between the planes: %.2f degrees', theta));
```

```
% Set the view angle for better visualization
view(3);
grid on;
axis equal;
% Plot the angle as an arc
t = linspace(0, deg2rad(theta), 100);
arc_x = cos(t);
arc_y = sin(t);
arc_z = zeros(size(t));
plot3(arc_x, arc_y, arc_z, 'k', 'LineWidth', 1.5, 'DisplayName', sprintf('Angle =

% Offset the arc to start from the tip of one normal vector
arc_offset = [0, 0, 0];
plot3(arc_x + arc_offset(1), arc_y + arc_offset(2), arc_z + arc_offset(3), 'k', '

% Add a text annotation to display the angle
text(0.5, 0.5, 0.5, sprintf('Angle: %.2f', theta), 'FontSize', 12, 'Color', 'k');
hold off;
```

Output

```
The angle between the planes is approximately 112.39 degrees.
```

Exercise

Exercise 1: Find the angle between the planes:

$$x + y + z = 3$$

and

$$2x - 3y - z = 1.$$

Exercise 2: Find the angle between the planes:

$$3x + y + 6z = 4$$

and

$$-x + y - 2z = 6.$$

Exercise 3: Find the angle between the planes:

$$2x - y + 3z = 8$$

and

$$x - y + 3z = 4.$$

Practical No. 20

Problem

Solve the system of equation of four variables which is define as

$$\begin{cases} 2x + 3y - z + w = 8, \\ x - 2y + 3z + 4w = -3, \\ 3x + y - 4z - w = 4, \\ 4x - 3y + 2z + 5w = 6. \end{cases}$$

Theory

We have to solve these system of equation using gauss-elimination method.

Step 1: Represent the System as a Matrix Equation

We begin by representing the system of equations in matrix form as:

$$A \cdot X = B$$

where

$$A = \begin{pmatrix} 2 & 3 & -1 & 1 \\ 1 & -2 & 3 & 4 \\ 3 & 1 & -4 & -1 \\ 4 & -3 & 2 & 5 \end{pmatrix}, \quad X = \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix}, \quad B = \begin{pmatrix} 8 \\ -3 \\ 4 \\ 6 \end{pmatrix}$$

Step 2: Solve for X using Gaussian Elimination

To solve for X, we apply Gaussian elimination, which involves performing row operations to transform matrix A into an upper triangular matrix. Then we perform back-substitution to solve for the variables x, y, z and w.

Row Operations

We will apply the following row operations:

- Use row operations to eliminate entries below the leading diagonal in the augmented matrix.
- Perform back-substitution to solve for the variables x, y, z, and w.

Final Solution

After applying the row operations, we obtain the values of x, y, z, and w.

Algorithm

Step 1: Define the coefficient matrix A and the constants vector B in MATLAB;

Step 2: Solve for vector X using `X = inv(A) * B`;

Step 3: Display the solution vector $[x, y, z, w]$.

Program

```
 % Define the coefficient matrix A and the constants vector B
 A = [2, 3, -1, 1;
     1, -2, 3, 4;
     3, 1, -4, -1;
     4, -3, 2, 5];

 B = [8; -3; 4; 6];

% Solve for the vector X
X = inv(A)*B;
disp('The solution for [x; y; z; w] is:');
disp(X);
```

Output

```
    The solution for [x; y; z; w] is:
    9.6667
    2.1667
    9.0000
   -8.8333
```

Exercise

Exercise 1: Solve the system of equation of four variables which is define as

$$\begin{cases} x + y + z + w = 1, \\ x - 2y + 3z - 4w = 2, \\ 2x + 3y - 4z + 5w = 3, \\ 3x - 4y + 5z - 6w = 4. \end{cases}$$

Exercise 2: Solve the system of equation of four variables which is define as

$$\begin{cases} x + 3y - z + w = 12, \\ x - 2y + 3z + 4w = -32, \\ x + y - 4z - w = 11, \\ x - 3y + 2z + 5w = 26. \end{cases}$$

Exercise 3: Solve the system of equation of four variables which is define as

$$\begin{cases} 2x + y - z + w = 0, \\ x - y + 3z + 4w = 3, \\ 3x + y - 5z - 3w = 9, \\ 4x - y + 7z + 5w = -6. \end{cases}$$

Appendices

Appendix A

MATLAB

Introduction to MATLAB

MATLAB (Matrix Laboratory) is a high-level programming language and environment primarily used for numerical computation, data analysis, and visualization. Developed in the 1980s, MATLAB has become one of the most popular tools in scientific computing and is widely used in various fields such as mathematics, engineering, physics, and economics.

At its core, MATLAB is designed to work with matrices, which makes it particularly useful for solving mathematical problems that involve linear algebra, abstract algebra, number theory, and differential equations. MATLABs strength lies in its ability to perform complex mathematical calculations with ease and in a highly efficient manner. In addition to its numerical capabilities, MATLAB also supports the development of algorithms, data visualization, and the integration of various hardware systems for real-time computation.

MATLAB Basics

MATLAB operates through a command-line interface, where users input commands that are executed immediately, and results are displayed in the Command Window. Users can also write scripts, which are sets of commands saved in a file for repeated use. MATLAB's syntax is simple and intuitive, making it accessible for beginners while being powerful enough for more advanced users. In command window, we run code by pressing **Enter** key. The script can be run by pressing **Ctrl + Enter** or pressing green triangle button.

Variables and Operators

In MATLAB, variables are assigned using the equal sign (`=`). For example, the following command assigns the value 5 to the variable `A`:

$$A = 5$$

MATLAB also supports standard arithmetic operations, such as:

- Addition: `+`
- Subtraction: `-`

- Multiplication: `*`
- Division: `/`
- Exponentiation: `^`

Linear Algebra in MATLAB

MATLAB was originally designed for matrix computation, and its functions are highly optimized for performing linear algebra operations.

Creating Matrices

Matrices are fundamental in MATLAB and can be created using square brackets. For example, the following code creates a 3x3 matrix:

$$A = [123; 456; 789]$$

This matrix represents:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Each row of the matrix is separated by a semicolon.

Matrix Operations

MATLAB provides several built-in functions to perform common matrix operations:

- Matrix addition: `C = A + B`
- Matrix multiplication: `C = A * B`
- Matrix transpose: `C = A'`
- Matrix inverse: `C = inv(A)`
- Determinant: `det(A)`
- Eigenvalues and eigenvectors: `[V, D] = eig(A)`

For instance, to compute the eigenvalues and eigenvectors of a matrix `A`, you would use:

$$[V, D] = \text{eig}(A)$$

where `V` is the matrix of eigenvectors and `D` is the diagonal matrix of eigenvalues.

*

Abstract Algebra in MATLAB

MATLAB is also useful in abstract algebra, particularly for tasks such as polynomial manipulation, solving systems of equations, and performing matrix operations that are central to group and ring theory.

Polynomials

In MATLAB, polynomials are represented by vectors of coefficients. For example, the polynomial $2x^3 + 3x^2 + x + 5$ is represented as:

$$p = [2, 3, 1, 5]$$

To evaluate a polynomial at a specific value of x, say $x = 2$, we use:

```
polyval(p, 2)
```

Solving Systems of Linear Equations

In abstract algebra, solving systems of linear equations is a common task. MATLAB provides a simple way to solve a system of equations $A\mathbf{x} = \mathbf{b}$ using the backslash operator:

$$x = A\backslash b$$

where `A` is the coefficient matrix and `b` is the column vector of constants.

Number Theory in MATLAB

MATLAB is also well-suited for number-theoretic computations, such as prime number generation, modular arithmetic, and greatest common divisor (GCD).

Prime Numbers

The function `isprime(n)` checks if the number n is prime. To generate all prime numbers up to a given limit, we can use:

```
primes(limit)
```

GCD and LCM

To calculate the greatest common divisor (GCD) of two numbers, we use:

$$\texttt{gcd}(36, 60)$$

which returns 12, the GCD of 36 and 60. Similarly, to compute the least common multiple (LCM), we use:

$$\texttt{lcm}(36, 60)$$

which returns 180, the LCM of 36 and 60.

Differential Equations in MATLAB

MATLAB provides extensive support for solving differential equations, including ordinary differential equations (ODEs) and partial differential equations (PDEs). The function `ode45` is widely used to solve first-order ODEs.

Solving ODEs

For example, to solve the first-order ODE:

$$\frac{dy}{dt} = -2y, \quad y(0) = 1$$

in MATLAB, we define the function and solve it using the following code:

```
ode = @(t, y) -2*y;
[t, y] = ode45(ode, [0 5], 1);
plot(t, y);
```

This code solves the ODE over the time interval $[0, 5]$ and plots the solution.

Appendix B

MATHEMATICA

Introduction to MATHEMATICA

MATHEMATICA is a comprehensive computational software system developed by Wolfram Research. It is used extensively for symbolic computation, numerical computation, data analysis, visualization, and algorithm development. MATHEMATICA is known for its ability to perform sophisticated calculations with ease, and it has found wide applications in mathematics, physics, engineering, economics, and other fields.

The system is designed to be highly versatile, allowing users to perform symbolic manipulations, solve complex equations, perform statistical analysis, and generate visualizations. MATHEMATICAs integrated environment supports a wide variety of mathematical and technical functions, making it an invaluable tool for students, researchers, and professionals.

MATHEMATICA Basics

MATHEMATICA uses a language that is both powerful and user-friendly. The system is built around a symbolic computation engine, which allows users to perform algebraic manipulations, including simplifying expressions, solving equations, and working with polynomials, matrices, and more. The MATHEMATICA environment allows for both interactive and programmatic use, enabling users to write scripts for repeatable tasks or simply input commands in real-time.

MATHEMATICA supports both functional and procedural programming, and its syntax is relatively intuitive. The system uses a notebook interface, where each document can contain code, text, and graphical output, making it a great tool for creating interactive reports and presentations. The code is run by pressing **SHIFT + ENTER** in MATHEMATICA notebook.

Basic Operations

In MATHEMATICA, simple arithmetic operations are straightforward. For example, to perform basic addition, subtraction, multiplication, and division, one can use:

```
2 + 3,  5 - 3,  4 * 2,  6 / 2
```

MATHEMATICA also provides the power operator (for exponentiation), which is denoted by `^`:

$$2^3 \quad \text{(which gives 8)}$$

Additionally, the `Sqrt` function is used for square roots:

$$\texttt{Sqrt[16]} \quad \text{(which gives 4)}$$

Linear Algebra in MATHEMATICA

MATHEMATICA is equipped with an extensive library of functions for performing matrix and vector operations. The system allows easy manipulation and computation of matrices, vectors, and other linear algebra concepts.

Creating Matrices

In MATHEMATICA, matrices are created using curly braces. For example, a 3x3 matrix can be created as follows:

$$A = \{\{1,2,3\},\{4,5,6\},\{7,8,9\}\}$$

This creates the matrix:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Matrix Operations

MATHEMATICA offers a wide variety of matrix operations. Some common operations include:

- Matrix addition: `A + B`
- Matrix multiplication: `A . B`
- Matrix transpose: `Transpose[A]`
- Matrix inverse: `Inverse[A]`
- Determinant: `Det[A]`
- Eigenvalues and eigenvectors: `Eigenvalues[A]` and `Eigenvectors[A]`

For example, to compute the eigenvalues of a matrix `A`, use:

```
Eigenvalues[A]
```

This command will return a list of the eigenvalues of the matrix `A`.

Abstract Algebra in MATHEMATICA

MATHEMATICA is well-suited for abstract algebra, including operations involving polynomials, groups, rings, and fields.

Polynomials

In MATHEMATICA, polynomials are represented as expressions. For example, the polynomial $2x^3 + 3x^2 + x + 5$ is written as:

$$2x^3 + 3x^2 + x + 5$$

To evaluate a polynomial at a specific value of x, say $x = 2$, use the `ReplaceAll` function:

$$2x^3 + 3x^2 + x + 5 \texttt{ /. x -> 2}$$

which will output the value of the polynomial when $x = 2$.

Solving Systems of Linear Equations

MATHEMATICA provides a simple function to solve systems of linear equations. Given a system of equations $A\mathbf{x} = \mathbf{b}$, you can use the function `LinearSolve[A, b]` to solve for $\mathbf{x}$:

```
LinearSolve[A, b]
```

Number Theory in MATHEMATICA

MATHEMATICA offers powerful tools for performing number-theoretic computations such as prime factorization, modular arithmetic, and computing greatest common divisors (GCD).

Prime Numbers

MATHEMATICA can be used to generate prime numbers and check if a number is prime. For example:

- To check if a number is prime, use `PrimeQ[n]`, where `n` is the number to be tested.
- To generate a list of primes up to a given number, use `Prime[Range[n]]`.

GCD and LCM

To calculate the greatest common divisor (GCD) of two numbers, use:

```
GCD[36, 60]
```

which will return 12, the GCD of 36 and 60. Similarly, to compute the least common multiple (LCM), use:

```
LCM[36, 60]
```

which returns 180, the LCM of 36 and 60.

Differential Equations in MATHEMATICA

MATHEMATICA offers sophisticated tools for solving both ordinary differential equations (ODEs) and partial differential equations (PDEs). The function `DSolve` is commonly used for solving ODEs symbolically.

Solving ODEs

For example, to solve the first-order ODE:

$$\frac{dy}{dt} = -2y, \quad y(0) = 1$$

you can use the following command in MATHEMATICA:

```
DSolve[y'[t] == -2 y[t], y[t], t]
```

This will return the solution $y(t) = e^{-2t}$.

MATHEMATICA can also handle systems of ODEs and higher-order differential equations, and provides numerous options for controlling the solution process.

Appendix C

Python

Introduction to Python

Python is a high-level, interpreted programming language known for its simplicity, readability, and versatility. Created by Guido van Rossum in the late 1980s and released in 1991, Python has become one of the most popular programming languages in the world. It is widely used in a variety of fields such as web development, data analysis, machine learning, scientific computing, automation, and more.

Pythons syntax is designed to be intuitive and easy to understand, which makes it an excellent language for both beginners and experienced developers. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Additionally, Python has a vast ecosystem of libraries and frameworks, making it an essential tool for many technical domains, including mathematics and scientific computation.

Python Basics

Python code is written in text files with the extension `.py`, and it can be executed in various environments, including interactive shells, scripts, or integrated development environments (IDEs) such as PyCharm or Jupyter Notebooks.

Pythons syntax is concise, and it emphasizes readability, making it easier for programmers to understand and maintain code. Some of the basic constructs in Python include variables, operators, data types, loops, and conditional statements.

Variables and Operators

In Python, variables are assigned using the equal sign (`=`). For example, to assign the value 5 to the variable `A`, you would write:

$$A = 5$$

Python supports a wide range of operators for performing arithmetic operations, such as:

- Addition: `+`
- Subtraction: `-`

- Multiplication: `*`
- Division: `/`
- Exponentiation: `**`

For example:

$$5 + 3 = 8, \quad 5 * 3 = 15, \quad 2 * *3 = 8$$

Linear Algebra in Python

Python is widely used in the mathematical community for tasks such as solving linear systems, matrix operations, and vector manipulations. The `NumPy` library is the core package for numerical computations in Python, providing efficient data structures such as arrays and matrices and functions to manipulate them.

Creating Arrays and Matrices

In Python, arrays and matrices can be created using the `numpy` library. To create a simple 1D array, use:

```
import numpy as np
A = np.array([1, 2, 3])
```

This creates the array:

$$A = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

To create a 2D matrix, use:

```
B = np.array([[1, 2], [3, 4]])
```

This creates the matrix:

$$B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Matrix Operations

Python, through `NumPy`, provides a wide range of functions to perform matrix operations:

- Matrix addition: `C = A + B`
- Matrix multiplication: `C = np.dot(A, B)` or `C = A @ B`
- Matrix transpose: `C = A.T`
- Matrix inverse: `C = np.linalg.inv(A)`
- Determinant: `det = np.linalg.det(A)`
- Eigenvalues and eigenvectors: `eigvals, eigvecs = np.linalg.eig(A)`

For example, to compute the determinant of a matrix `A`, use:

```
det = np.linalg.det(A)
```

Abstract Algebra in Python

Python can be used for performing various tasks in abstract algebra, such as polynomial manipulation, solving systems of equations, and working with groups and rings.

Polynomials

Polynomials in Python can be represented using `NumPy` or the `sympy` library for symbolic computation. For example, the polynomial $2x^3 + 3x^2 + x + 5$ can be created as:

```
from sympy import symbols, Poly

x = symbols('x')

poly = Poly(2*x**3 + 3*x**2 + x + 5, x)
```

To evaluate the polynomial at $x = 2$, use:

```
poly.subs(x, 2)
```

Solving Systems of Linear Equations

Python makes it easy to solve systems of linear equations. Given a system of equations $A\mathbf{x} = \mathbf{b}$, you can solve it using `NumPy` as follows:

```
A = np.array([[2, 1], [1, 3]])

b = np.array([5, 7])

x = np.linalg.solve(A, b)
```

This returns the solution vector `x`.

Number Theory in Python

Python has excellent libraries for number theory tasks, such as generating prime numbers, calculating the greatest common divisor (GCD), and performing modular arithmetic.

Prime Numbers

Python can be used to check if a number is prime and to generate a list of primes. The `sympy` library provides an easy way to work with prime numbers:

```
from sympy import isprime, primerange
```

`isprime(7)` (returns True)

`list(primerange(1, 20))` (returns [2, 3, 5, 7, 11, 13, 17, 19])

GCD and LCM

To calculate the greatest common divisor (GCD) of two numbers, use:

```
import math

math.gcd(36, 60)
```

(returns 12)

Similarly, to compute the least common multiple (LCM), use:

```
math.lcm(36, 60)
```

(returns 180)

Differential Equations in Python

Python offers powerful libraries like `SciPy` for solving ordinary differential equations (ODEs). The `scipy.integrate` module contains functions for numerical integration, such as `odeint`, which can be used to solve ODEs.

Solving ODEs

To solve the first-order ODE:

$$\frac{dy}{dt} = -2y, \quad y(0) = 1$$

you can define the ODE function and use `odeint` to solve it:

```
from scipy.integrate import odeint

def model(y, t):  return -2 * y

t = np.linspace(0, 5, 100)

y0 = 1

y = odeint(model, y0, t)
```

This will solve the ODE and return the solution as an array of values for $y(t)$.

NumPy: Numerical Computation in Python

`NumPy` is one of the most important libraries in Python for numerical computing. It provides support for multidimensional arrays, matrices, and a large collection of high-level mathematical functions to operate on these arrays.

Creating NumPy Arrays

NumPy arrays are more efficient than Python's built-in lists for mathematical operations. You can create a NumPy array as follows:

```
import numpy as np

A = np.array([1, 2, 3, 4, 5])
```

This creates a 1D array:

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \end{bmatrix}$$

For a 2D array (or matrix):

```
B = np.array([[1, 2], [3, 4], [5, 6]])
```

This creates the matrix:

$$B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

Array Operations

NumPy supports element-wise operations on arrays, such as:

- Addition: `A + B`
- Multiplication: `A * B`
- Scalar multiplication: `A * 3`

It also supports more complex operations like dot products, transposition, and linear algebra functions.

Matplotlib: Visualization in Python

`Matplotlib` is a plotting library for creating static, interactive, and animated visualizations in Python. It is widely used for plotting graphs and charts, including line plots, bar charts, histograms, scatter plots, and more.

Basic Plotting with Matplotlib

To create a basic line plot, you can use the following commands:

```
import matplotlib.pyplot as plt

x = np.linspace(0, 10, 100)
y = np.sin(x)
plt.plot(x, y)
plt.show()
```

This will generate a plot of the sine function from 0 to 10.

Customization of Plots

`Matplotlib` allows extensive customization of plots, including adding labels, titles, and legends:

```
plt.plot(x, y, label="Sine Wave")
plt.title("Sine Function")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```

Bibliography

[1] Abell, M. L., and Braselton, J. P. *The mathematica handbook.* Academic Press, 2014.

[2] Abell, M. L., and Braselton, J. P. *Mathematica by example.* Academic Press, 2017.

[3] Chapman, S. J. *MATLAB programming for engineers.* Brooks/Cole Publishing Co., 2001.

[4] Chun, W. *Core python programming*, vol. 1. Prentice Hall Professional, 2001.

[5] Dart-Enander, E., Part-Enander, E., and Sjoberg, A. *The MATLAB 5 handbook.* Addison-Wesley Longman Publishing Co., Inc., 1999.

[6] Downey, A. B. *How to think like a computer scientist.* 2003.

[7] Garg, A. K., and Thomas, A. *Development of Mathematics Practical Manual for B. Sc. B. Ed. Level.* Regional Institute of Education (NCERT), Bhopal, 2020.

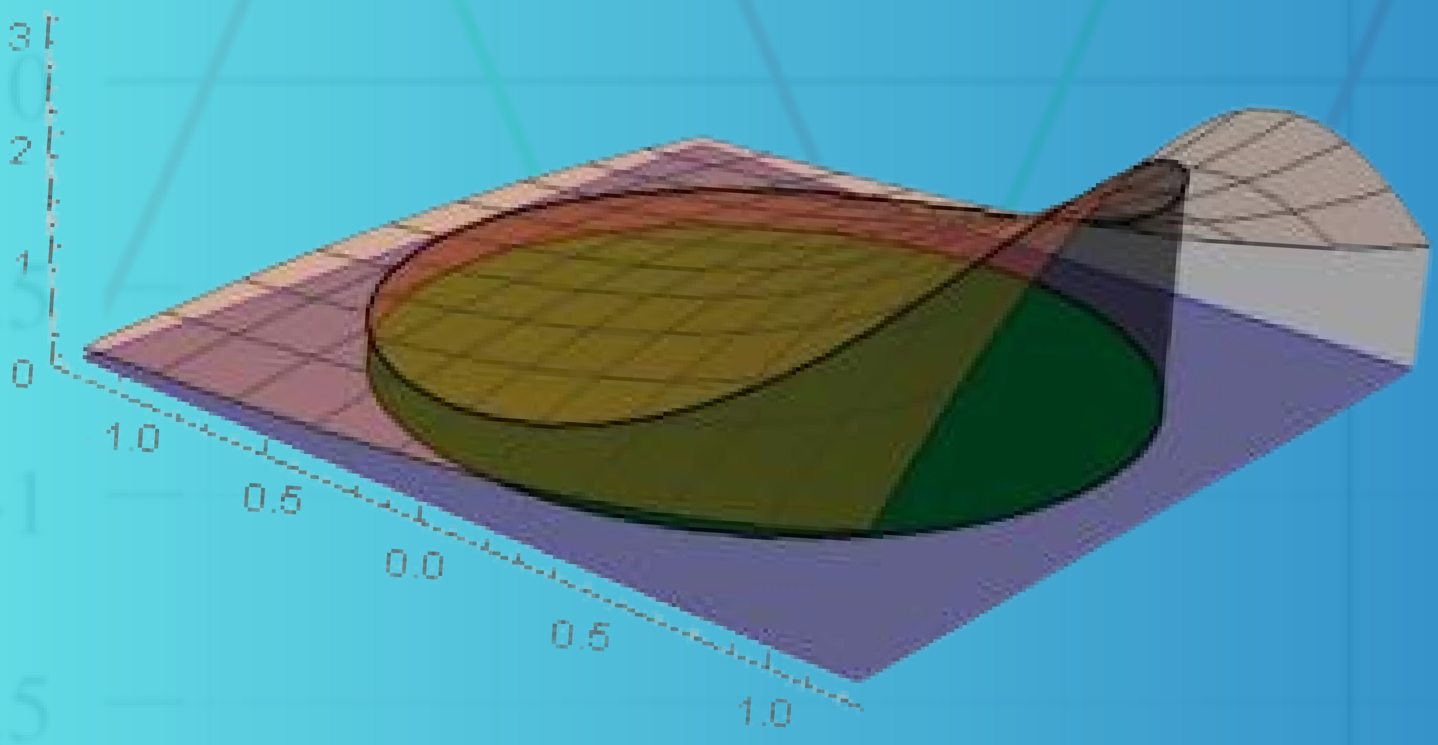
[8] Gilat, A. *MATLAB: An introduction with Applications.* John Wiley & Sons, 2017.

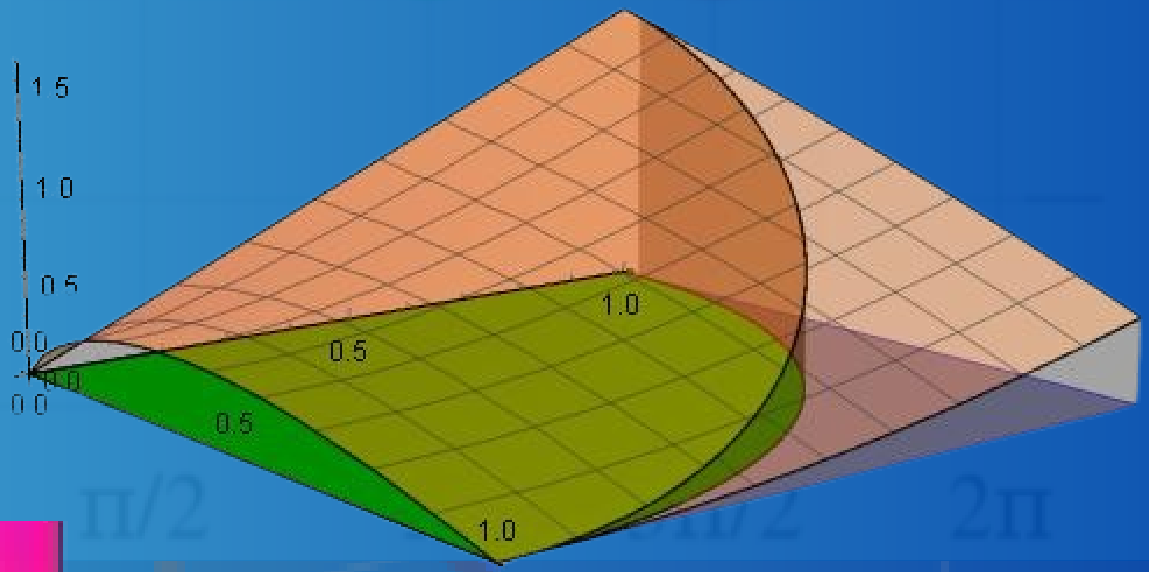
[9] Gowrishankar, S., and Veena, A. *Introduction to Python programming.* Chapman and Hall/CRC, 2018.

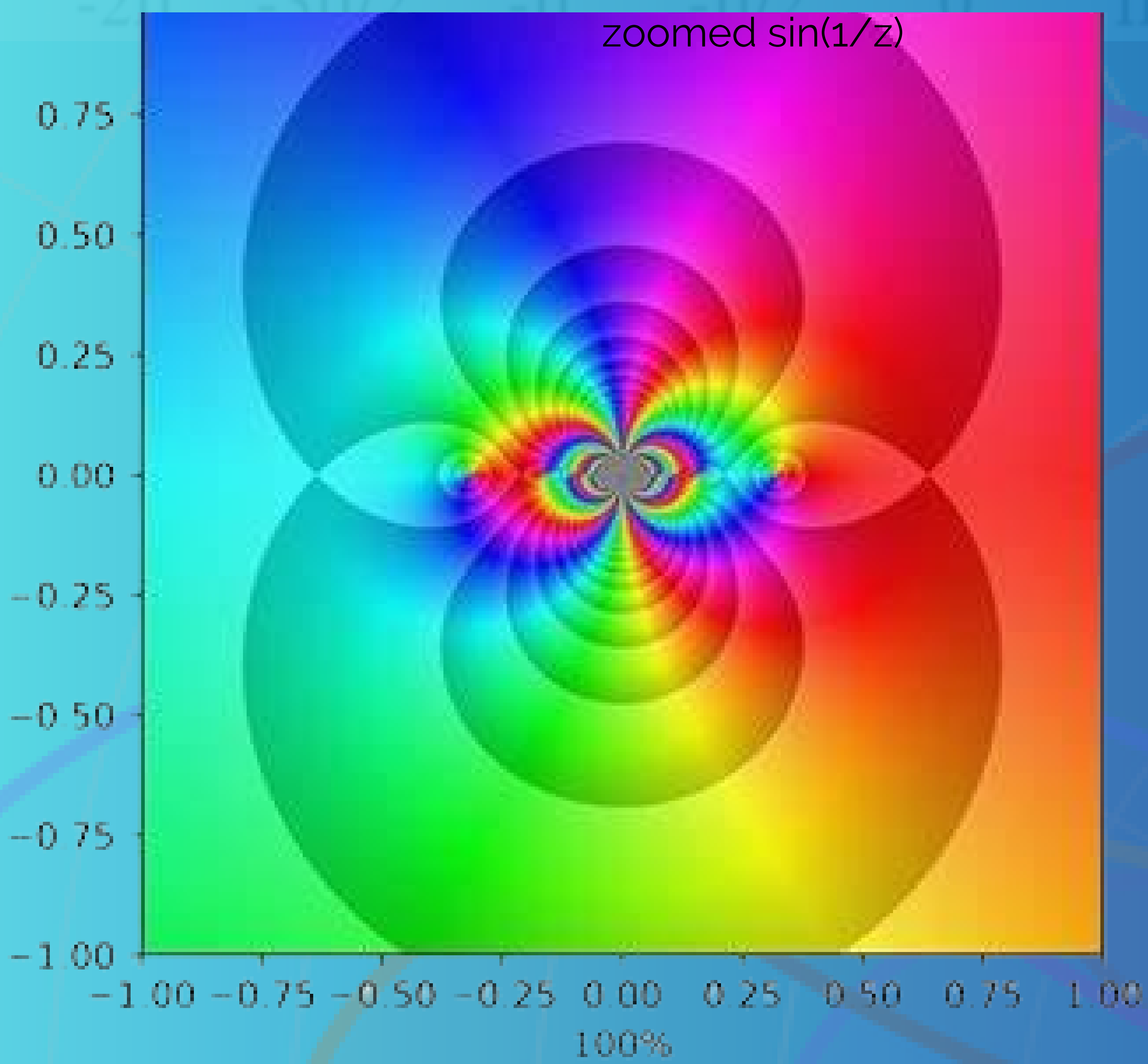
[10] Kattan, P. *Matlab for beginners*, vol. 1. Petra books, 2022.

[11] Lutz, M. *Programming python.* O'Reilly Media, Inc., 2001.

[12] Trott, M. *The Mathematica guidebook for programming.* Springer, 2013.





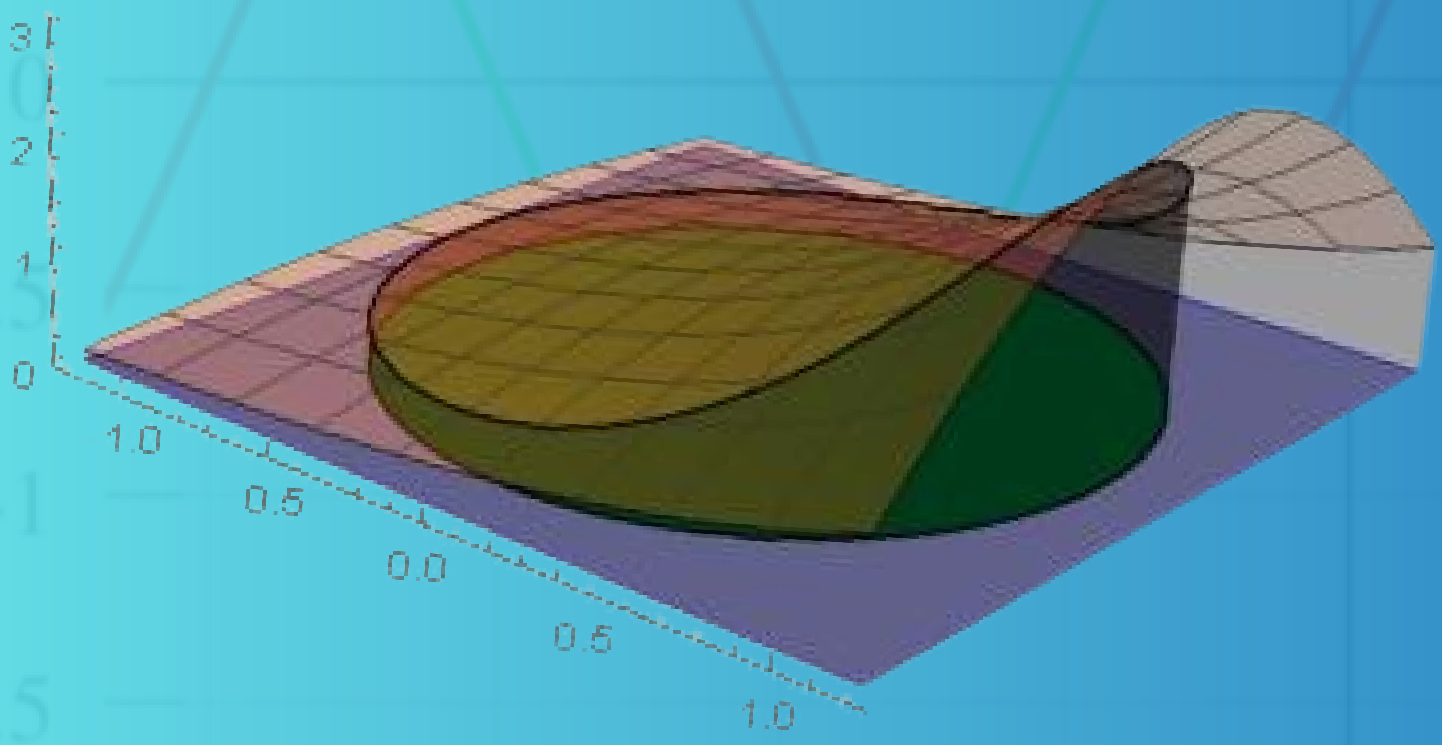
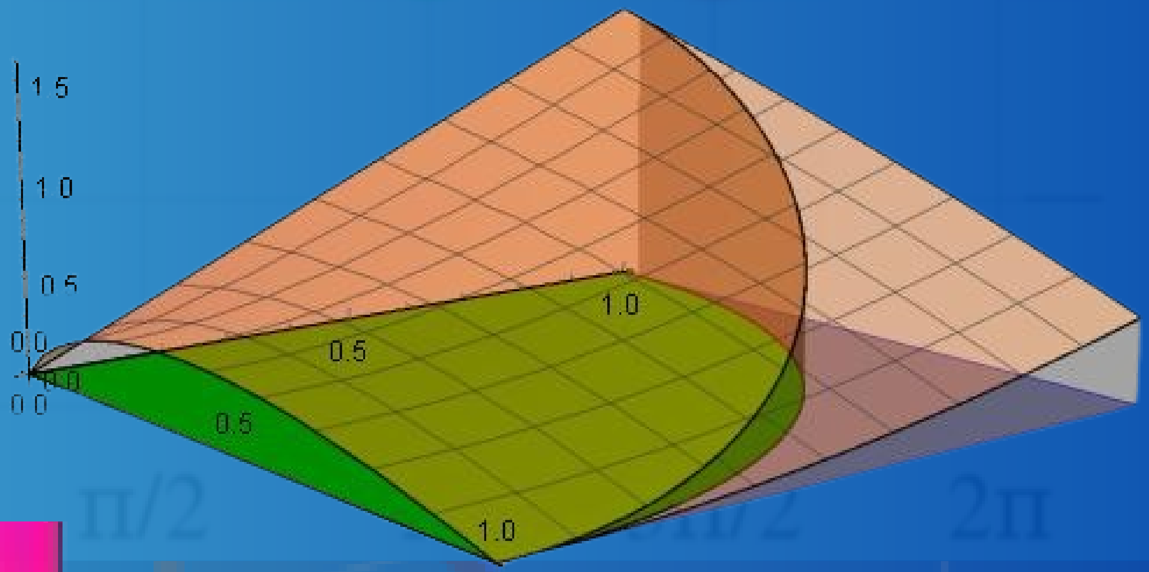
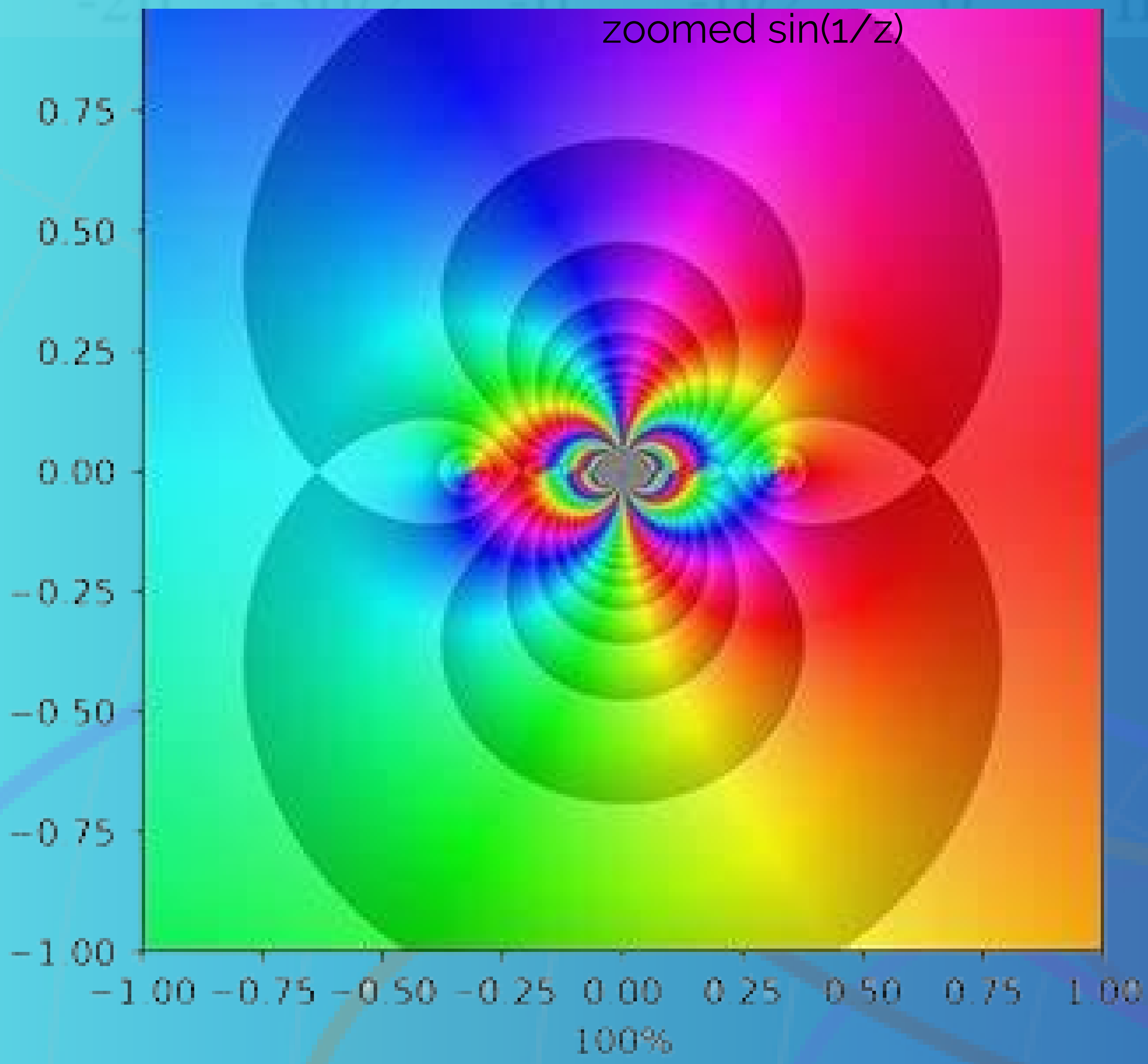


विद्यया ऽ मृतमश्नुते

एन सी ई आर टी
NCERT

REGIONAL INSTITUTE OF EDUCATION

National Council of Educational Research and Training
Shyamala Hills, Bhopal, 102002

विद्यया ऽ मृतमश्नुते

एन सी ई आर टी
NCERT

REGIONAL INSTITUTE OF EDUCATION
National Council of Educational Research and Training
Shyamala Hills, Bhopal, 102002