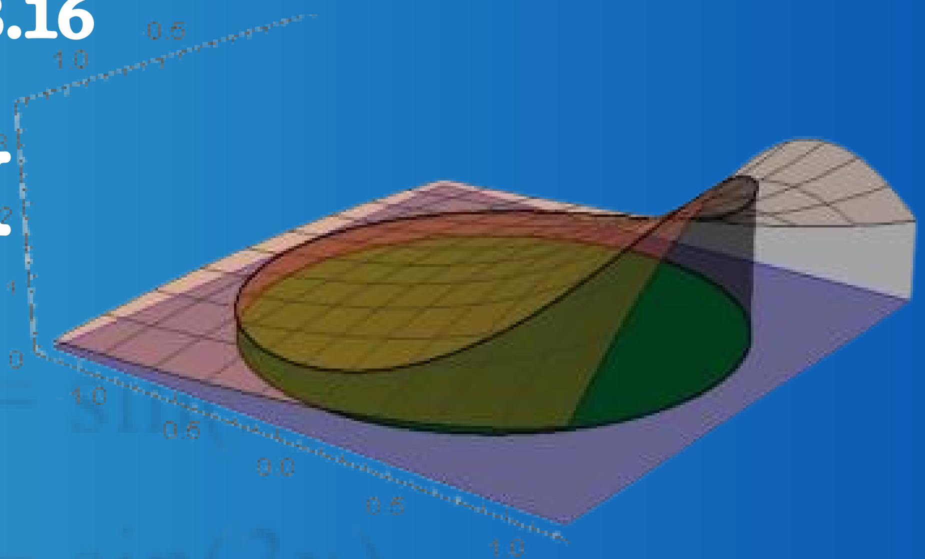
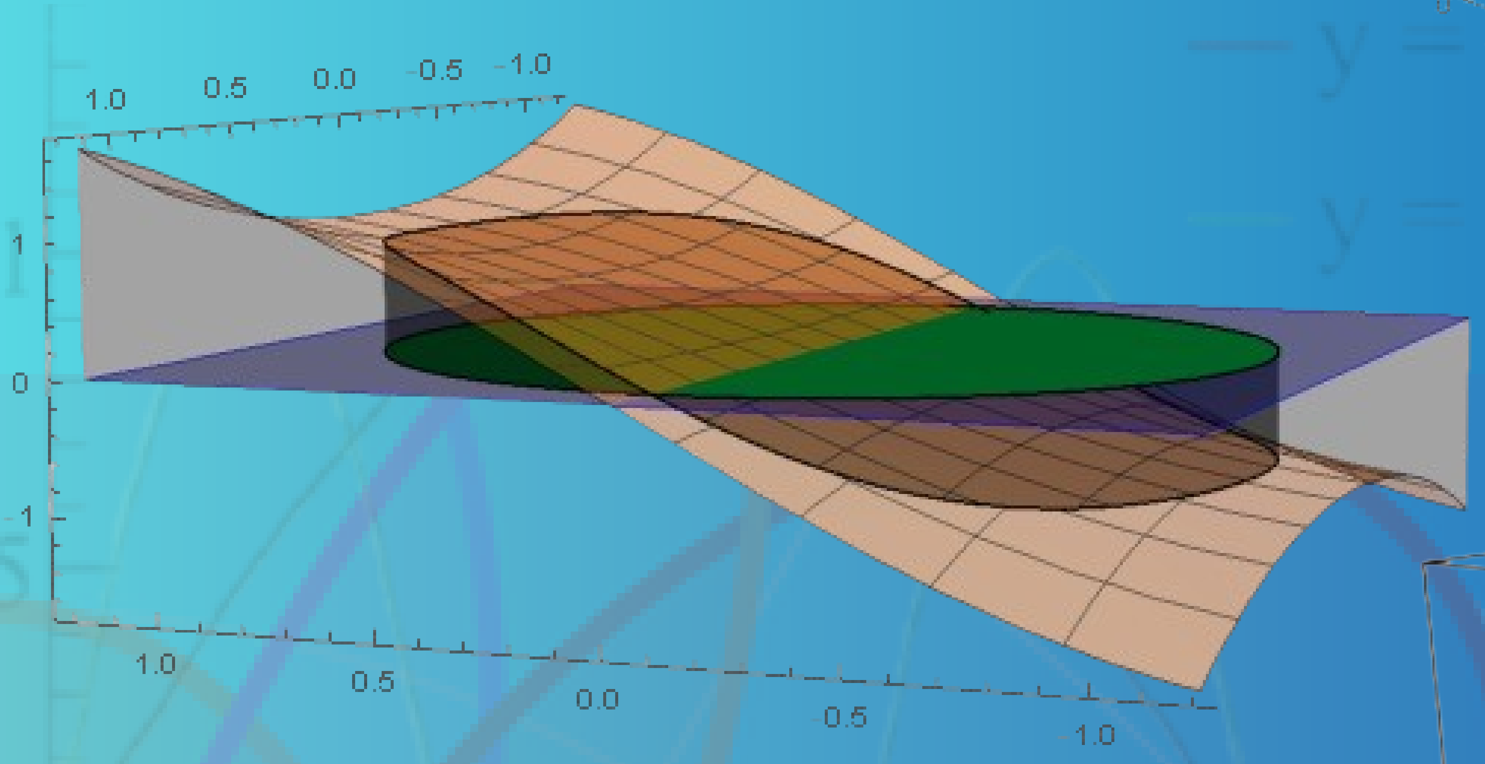
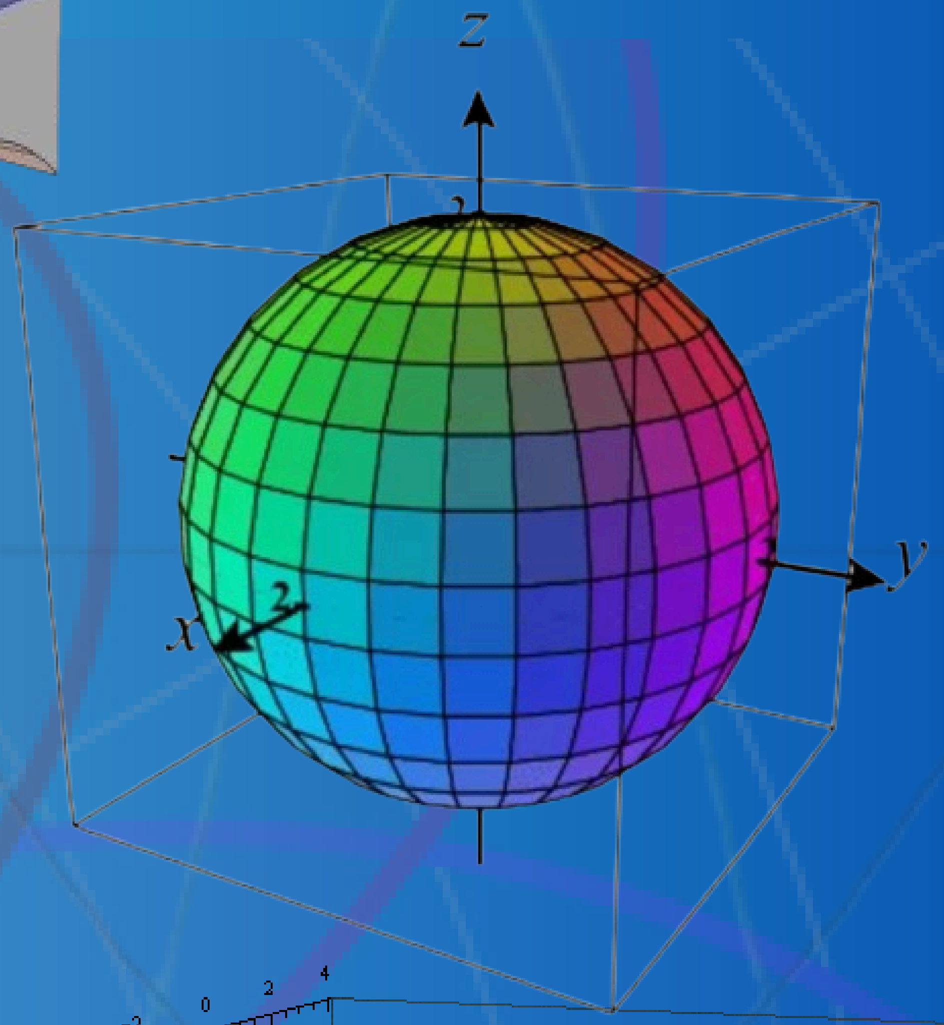
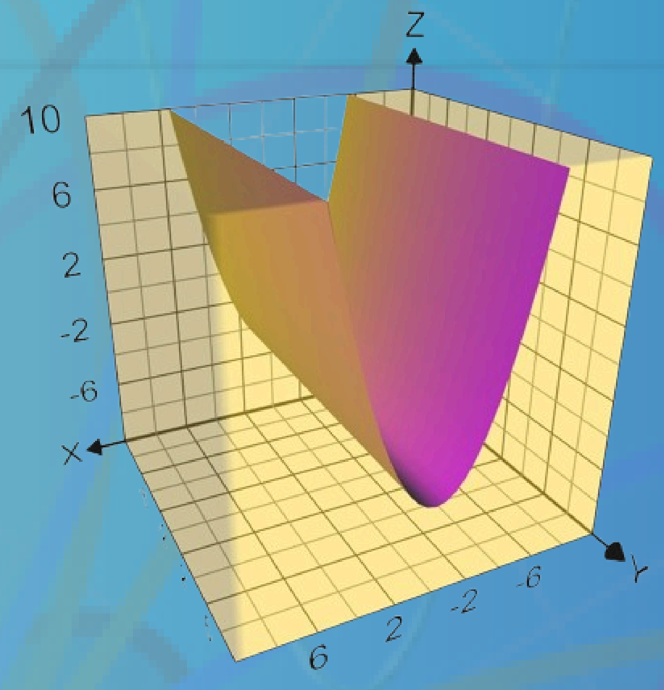
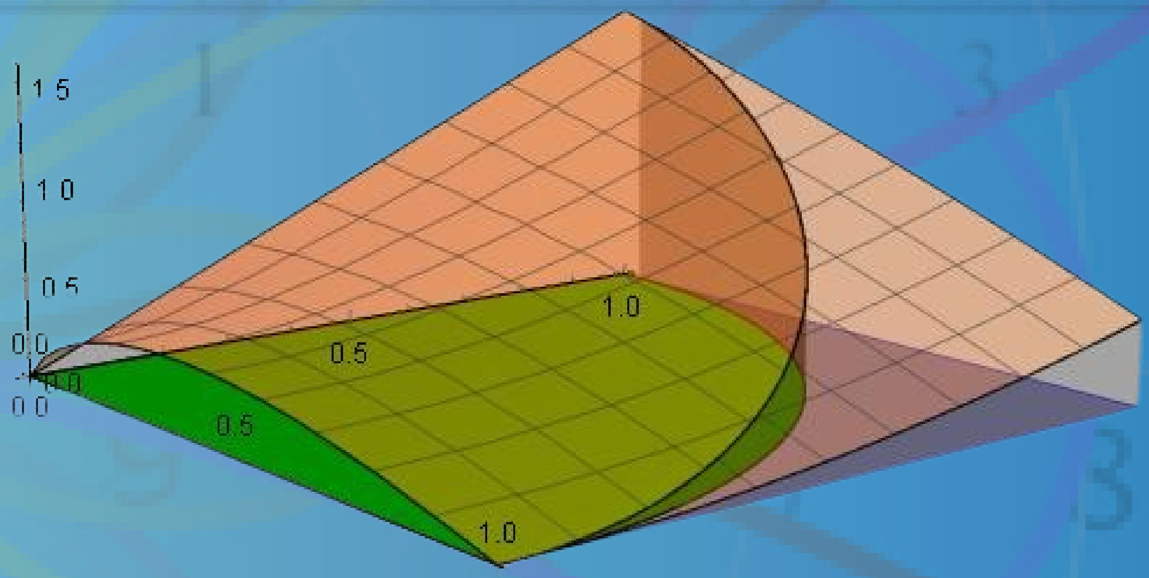
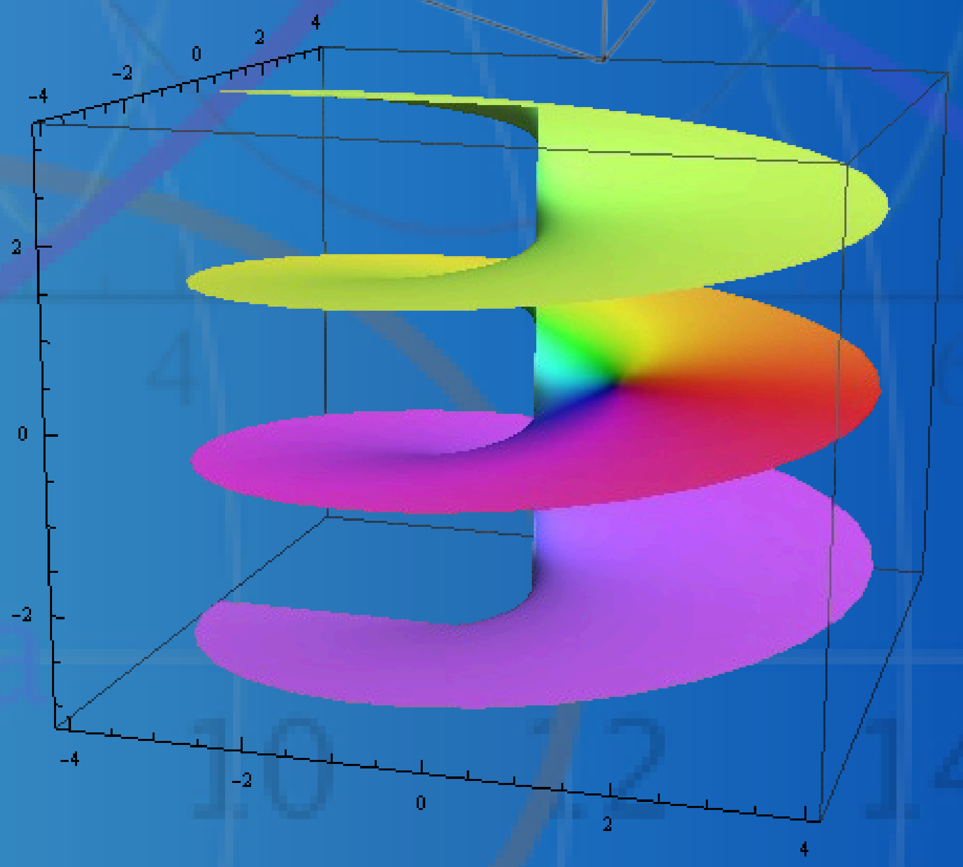
DEVELOPMENT OF MATHEMATICS PRACTICAL LABORATORY MANUAL AS PER THE ITEP SYLLABUS

2024-25

PAC Code 23.16

Part - I

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-I

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

Algebra and Trigonometry

Practical No. 1

Aim

To understand and verify the relationships between the roots and coefficients of a polynomial equation using MATLAB.

Problem

For the given polynomial equation $x^3 - 6x^2 + 11x - 6 = 0$, find the roots of the polynomial using MATLAB and verify the relationships between the roots and coefficients based on polynomial theory.

Theory

For a cubic polynomial of the form:

$$ax^3 + bx^2 + cx + d = 0,$$

where a, b, c, and d are the coefficients, the relationships between the roots r_1, r_2, and r_3 and these coefficients are as follows:

(i) Sum of the roots:

$$r_1 + r_2 + r_3 = -\frac{\text{coefficient of } x^2}{\text{coefficient of } x^3} = -\frac{b}{a}.$$

(ii) Sum of the product of roots taken two at a time:

$$r_1r_2 + r_1r_3 + r_2r_3 = \frac{\text{coefficient of } x}{\text{coefficient of } x^3} = \frac{c}{a}.$$

(iii) Product of the roots:

$$r_1r_2r_3 = -\frac{\text{constant term}}{\text{coefficient of } x^3} = -\frac{d}{a}.$$

Algorithm

Step 1: Define the coefficients of the cubic polynomial $ax^3 + bx^2 + cx + d = 0$ in MATLAB by creating a vector `coeff` containing the values of the coefficients;
Step 2: Calculate the roots of the polynomial using MATLAB's `roots` function. The roots are computed from the coefficients and stored in a variable `roots_of_poly`;
Step 3: Compute the sum of the roots using MATLAB's `sum` function and store the result in the variable `sum_of_roots` This value should be equal to $-\frac{b}{a}$;
Step 4: Compute the product of the roots using MATLAB's `prod` function and store the result in the variable `product_of_roots`. This value should be equal to $-\frac{d}{a}$;
Step 5: Compute the sum of the products of the roots taken two at a time using manual multiplication and addition, and store the result in `sum_twoproduct_roots`. This value should be equal to $\frac{c}{a}$;
Step 6: Display the roots of the polynomial using the `disp` command;
Step 7: Display the relationships between the roots and coefficients using the `disp` command, verifying that the following conditions hold:

- The sum of the roots is equal to $-\frac{b}{a}$.
- The sum of the products of the roots taken two at a time is equal to $\frac{c}{a}$.
- The product of the roots is equal to $-\frac{d}{a}$.

Program

```
% Define coefficients of the polynomial x^3 - 6x^2 + 11x - 6 = 0
coeff = [1 -6 11 -6];  % Coefficients: x^3 - 6x^2 + 11x - 6

% Step 1: Calculate roots
roots_of_poly = roots(coeff);

% Step 2: Compute relationships between roots and coefficients
sum_of_roots = sum(roots_of_poly);                 % Sum of roots
product_of_roots = prod(roots_of_poly);            % Product of roots
sum_twoproduct_roots = roots_of_poly(1)*roots_of_poly(2) + ...
roots_of_poly(1)*roots_of_poly(3) + ...
roots_of_poly(2)*roots_of_poly(3); % Sum of products taken two at a time

% Display roots
disp('Roots of the polynomial:');
disp(roots_of_poly);

% Display relationship between roots and coefficients of the polynomial
disp('Relationships between roots and coefficients:');

disp(['Sum of roots: ', num2str(sum_of_roots), ', which is the same as ' ...
'(-coefficient of x^2) / (coefficient of x^3) = ', num2str(-coeff(2)/coeff(1))])

disp([‘Sum of the product of roots taken two at a time: ', num2str
(sum_twoproduct_roots), ...
', which is the same as (coefficient of x) / (coefficient of x^3) = ',
 num2str(coeff(3)/coeff(1))]);

disp([‘Product of roots: ', num2str(product_of_roots), ', which is the same as '
'(-constant term) / (coefficient of x^3) = ', num2str(-coeff(4)/coeff(1))]);
```

Output

```
Roots of the polynomial:
3.0000
2.0000
1.0000

Relationships between roots and coefficients:
Sum of roots= 6, and (-coefficient of x^2) / (coefficient of x^3) = 6
Sum of the product of roots taken two at a time= 11,
and (coefficient of x) / (coefficient of x^3) = 11
Product of roots= 6, and (-constant term) / (coefficient of x^3) = 6
```

Conclusion

The MATLAB program verifies the relationships between the roots and coefficients of a cubic polynomial.

Exercise Problem:

For the given polynomial equation $x^4 + 4x^2 + 5x + 2 = 0$, find the roots of the polynomial using MATLAB and verify the relationships between the roots and coefficients based on polynomial theory.

Practical No. 2

Aim

To construct a polynomial from given roots and evaluate the polynomial at specified points using MATLAB. Additionally, plot the polynomial and locate the values of $p(x)$ at any arbitrary value of x.

Problem

Given the polynomial $P(x)$ of degree 4 with roots i, 1, and 2, find the polynomial using MATLAB. Compute $P(2)$, $P(-2)$, and $P(5)$, and plot the graph of $P(x)$, locating the values of $P(x)$ at $x = 2$, $x = -2$, and $x = 5$ on the plot.

Theory

A polynomial of degree 4 with roots r_1, r_2, r_3, r_4 can be written as:

$$P(x) = a(x - r_1)(x - r_2)(x - r_3)(x - r_4)$$

where a is a constant that can be chosen for normalization. If the roots are i, 1, and 2, then the polynomial can be expressed as:

$$P(x) = a(x - i)(x + i)(x - 1)(x - 2)$$

Since, the roots include i and $-i$, the term $(x - i)(x + i) = x^2 + 1$ can be simplified. So, the polynomial becomes:

$$P(x) = a(x^2 + 1)(x - 1)(x - 2).$$

For this problem, a can be assumed to be 1 for simplicity.

Algorithm

Step 1: Define the roots 1, 2, i, and $-i$ in MATLAB;
Step 2: Compute the polynomial coefficients using MATLAB's `poly` function;
Step 3: Define the polynomial $P(x)$ using `polyval`;
Step 4: Compute the values $P(2)$, $P(-2)$, and $P(5)$ using `polyval`;
Step 5: Plot the polynomial using `plot`;
Step 6: Mark the points at $x = 2$, $x = -2$, and $x = 5$ with filled markers;
Step 7: Display the polynomial equation and values at $x = 2$, $x = -2$, and $x = 5$.

Program

```
% Define the roots of the polynomial
roots_given = [1, 2, 1i, -1i];

% Generate the polynomial coefficients from the roots
coeffs_from_roots = poly(roots_given);

% Display the polynomial coefficients
disp(‘Polynomial coefficients from given roots:’);
disp(coeffs_from_roots);

% Define the polynomial function using the coefficients
p = @(x) polyval(coeffs_from_roots, x);

% Calculate the values of p(x) at x = 2, x = -2, x = 5
p_at_2 = p(2);
p_at_minus_2 = p(-2);
p_at_5 = p(5);

% Display the values at these points
disp(‘p(2) =’);
disp(p_at_2);
disp(‘p(-2) =’);
disp(p_at_minus_2);
disp(‘p(5) =’);
disp(p_at_5);

% Plot the polynomial over a specified range
x_vals = linspace(-3, 6, 500);  % Generate 500 points for a smooth curve
y_vals = p(x_vals);  % Calculate polynomial values at these points

figure;  % Open a new figure window
plot(x_vals, y_vals, ’b-’, ’LineWidth’, 1);


% Plot the polynomial in blue with thinner line
hold on;

% Mark the points on the graph at x = 2, x = -2, x = 5 with filled markers
plot(2, p_at_2, ’go’, ‘MarkerFaceColor’, ‘g’);  % Point at x = 2 with
green filled circle
plot(-2, p_at_minus_2, ’ro’, ‘MarkerFaceColor’, ‘r’);  % Point at x =-2
with red filled circle
plot(5, p_at_5, ‘mo’, ’MarkerFaceColor’, ‘m’);  % Point at x = 5 with magenta
```

```
filled circle

% Add a legend for the points
legend('p(x)', 'p(2)', 'p(-2)', 'p(5)', 'Location', 'best');

% Label the axes and add title
xlabel('x');
ylabel('p(x)');
title('Plot of the Polynomial p(x)');

% Add grid and set axis limits for better visualization
grid on;
axis([-3 6 -1 350]);  % Adjust axis limits for better visualization
hold off;
```

Output

```
The polynomial equation is:
p(x) = 1x^3 + -3x^2 + 3x + -3

p(2) =
0

p(-2) =
60

p(5) =
312
```

Conclusion

The MATLAB program successfully constructs the polynomial from the given roots, evaluates the polynomial at specific points, and plots the graph of the polynomial with the marked points.

Exercise Problem:

Given that $p(x)$ is a polynomial of degree 3 with roots 3, 1, and -1, find the polynomial expression for $p(x)$ and also compute the value of $p(5)$ using MATLAB.

Practical No. 3

Aim

To transform a polynomial equation into another equation whose roots are the roots of the given polynomial equation multiplied by a constant k, and find the roots of both polynomials using MATLAB.

Problem

Write a MATLAB program that takes a polynomial $p(x) = x^3 - 6x^2 + 11x - 6$, transforms it into a new polynomial $q(y)$ such that the roots of $q(y)$ are the roots of $p(x)$ multiplied by a constant $k = 2$, and finds the roots of both polynomials.

Theory

Given a polynomial equation $p(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 = 0$, we wish to create a transformed polynomial $q(y)$, whose roots will be k times the roots of the original equation $p(x)$. If x is the root of $p(x)$ and y is the root of $q(y)$, then we have $y = kx$. Substituting $x = \frac{y}{k}$ with $k \neq 0$ into $p(x)$, we get:

$$p\left(\frac{y}{k}\right) = a_n\left(\frac{y}{k}\right)^n + a_{n-1}\left(\frac{y}{k}\right)^{n-1} + \cdots + a_1\left(\frac{y}{k}\right) + a_0 = 0.$$

By multiplying the entire equation by k^n, we obtain the following.

$$q(y) = a_n y^n + a_{n-1}ky^{n-1} + \cdots + a_1 k^{n-1}y + a_0 k^n = 0.$$

This new polynomial $q(y)$ has roots $y = k \cdot x$, or equivalently, the roots of the original polynomial multiplied by k.

Algorithm

Step 1: Define the polynomial $p(x) = x^3 - 6x^2 + 11x - 6$ as a vector of coefficients;
Step 2: Specify the constant k by which we want to multiply the roots;
Step 3: Calculate the coefficients of the transformed polynomial $q(y)$;
Step 4: Find the roots of $p(x)$ and $q(y)$ using MATLABs 'roots' function;
Step 5: Display the transformed polynomial and the roots of both polynomials.

Program

```
% Define the original polynomial p(x) = x^3 - 6x^2 + 11x - 6
p_coefficients = [1, -6, 11, -6];
```

```
% Define the constant k
k = 2;

% Initialize an array to hold the transformed polynomial coefficients
q_coefficients = zeros(1, length(p_coefficients));

% Loop through each coefficient and multiply by the appropriate power of k
for i = 1:length(p_coefficients)
power = i-1; % Degree of the term
q_coefficients(i) = p_coefficients(i) * k^power;
end

% Define a symbolic variable y for displaying the polynomial
syms y;
% Convert the coefficients to a symbolic polynomial
q_polynomial = poly2sym(q_coefficients, y);

% Find the roots of the original polynomial p(x)
roots_p = roots(p_coefficients);

% Find the roots of the transformed polynomial q(y)
roots_q = roots(q_coefficients);

% Display the transformed polynomial
disp('The transformed polynomial q(y) is:');
disp(q_polynomial);

% Display the roots of both polynomials
disp('Roots of the original polynomial p(x):');
disp(roots_p);
disp('Roots of the transformed polynomial q(y):');
disp(roots_q);
```

Output

```
The transformed polynomial q(y) is:
y^3 - 12*y^2 + 44*y - 48

Roots of the original polynomial p(x):
   3.0000
   2.0000
   1.0000
```

```
 Roots of the transformed polynomial q(y):
    6.0000
    4.0000
    2.0000
```

Conclusion

This MATLAB code successfully transforms the polynomial $p(x) = x^3 - 6x^2 + 11x - 6$ into a new polynomial $q(y)$ whose roots are the roots of $p(x)$ multiplied by $k = 2$. The roots of both the original and transformed polynomials are calculated and displayed, verifying the transformation.

Exercise Problem

Transform the polynomial $p(x) = x^3 - 4x^2 + 5x - 1$ into another polynomial whose roots are the roots of the original polynomial multiplied by $k = 3$. Find and compare the roots of both polynomials.

Practical No. 4

Aim

To transform a general polynomial equation into another equation whose roots are the roots of the given polynomial equation multiplied by a constant k, and find the roots of both polynomials using MATLAB.

Problem

Write a MATLAB program that allows the user to input any polynomial equation and a constant k. The program should transform the polynomial into a new equation whose roots are k times the roots of the original polynomial. The program should then find and display the roots of both the original and the transformed polynomials.

Theory

The transformation process is based on the substitution $y = kx$, where x x is a root of the original polynomial, and y is a root of the transformed polynomial. By substituting $x = \frac{y}{k}$ with $k \neq 0$ into the original polynomial $p(x)$, we derive the transformed polynomial $q(y)$.

Algorithm

Step 1: Prompt the user to input the coefficients of $p(x)$ by using the command: `p_coefficients = input('Coefficients: ');`
Step 2: Prompt the user to input the constant k by using the command: `k = input('Enter the value of constant k: ');`
Step 3: Initialize an array to store the coefficients of the transformed polynomial $q(y)$ by using the command: `q_coefficients = zeros(1, length(p_coefficients));`
Step 4: Loop through each coefficient of $p(x)$ and multiply it by the appropriate power of k by using the command:

```
for i = 1:length(p_coefficients)
    power = i-1; % Degree of the term
    q_coefficients(i) = p_coefficients(i) * k^power;
end
```

Step 5: Define a symbolic variable y and convert the coefficients to a symbolic polynomial by using the command: `syms y; q_polynomial = poly2sym(q_coefficients, y);`
Step 6: Find the roots of the original polynomial $p(x)$ by using the command: `roots_p = roots(p_coefficients);`
Step 7: Find the roots of the transformed polynomial $q(y)$ by using the command: `roots_q = roots(q_coefficients);`
Step 8: Display the transformed polynomial and the roots of both polynomials by using the command:

```
disp(q_polynomial);
disp(roots_p);
disp(roots_q);
```

Program

```
% User inputs
disp('Enter the coefficients of the polynomial p(x) (highest to lowest degree):');
disp('Example: For p(x) = 2x^3 + 3x^2 - 5x + 4, enter [2 3 -5 4]');
p_coefficients = input('Coefficients: ');

k = input('Enter the value of constant k: ');

% Initialize an array to hold the transformed polynomial coefficients
q_coefficients = zeros(1, length(p_coefficients));

% Loop through each coefficient and multiply by the appropriate power of k
for i = 1:length(p_coefficients)
    power =  i-1; % Degree of the term
    q_coefficients(i) = p_coefficients(i) * k^power;
end
```

```
% Define a symbolic variable y for displaying the polynomial
syms y;
% Convert the coefficients to a symbolic polynomial
q_polynomial = poly2sym(q_coefficients, y);

% Find the roots of the original polynomial p(x)
roots_p = roots(p_coefficients);

% Find the roots of the transformed polynomial q(y)
roots_q = roots(q_coefficients);

% Display the transformed polynomial
disp('The transformed polynomial q(y) is:');
disp(q_polynomial);

% Display the roots of both polynomials
disp('Roots of the original polynomial p(x):');
disp(roots_p);
disp('Roots of the transformed polynomial q(y):');
disp(roots_q);
```

Output:

```
Coefficients: [1,1,-5,3]
Enter the value of constant k: 2
The transformed polynomial q(y) is:
y^3 + 2*y^2 - 20*y + 24

Roots of the original polynomial p(x):
  -3.0000 + 0.0000i
   1.0000 + 0.0000i
   1.0000 - 0.0000i

Roots of the transformed polynomial q(y):
  -6.0000 + 0.0000i
   2.0000 + 0.0000i
   2.0000 - 0.0000i
```

Conclusion

This MATLAB code allows users to transform any polynomial equation into a new polynomial whose roots are multiplied by a specified constant k. The program successfully calculates and displays the roots of both the original and transformed polynomials.

Practical No. 5

Aim

To transform a polynomial equation into another equation whose roots are the same in magnitude but opposite in sign of the roots of the given polynomial equation, and find the roots of both polynomials using MATLAB.

Problem

Write a MATLAB program that takes a polynomial $p(x) = x^3 - 6x^2 + 11x - 6$, transforms it into a new polynomial $q(y)$ such that the roots of $q(y)$ are the same in magnitude but opposite in sign of the roots of $p(x)$, and finds the roots of both polynomials.

Theory

Given a polynomial equation $p(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 = 0$, we wish to create a transformed polynomial $q(y)$, whose roots will be the negatives of the roots of the original equation $p(x)$. If x is the root of $p(x)$ and y is the root of $q(y)$, then we have $y = -x$.

To find the transformed polynomial, we substitute $x = -y$ into the original polynomial $p(x)$:

$$p(-y) = a_n(-y)^n + a_{n-1}(-y)^{n-1} + \cdots + a_1(-y) + a_0 = 0.$$

This gives the new polynomial $q(y)$ as:

$$q(y) = a_n(-y)^n + a_{n-1}(-y)^{n-1} + \cdots + a_1(-y) + a_0 = 0.$$

This polynomial $q(y)$ has roots that are the negatives of the roots of the original polynomial.

Algorithm

Step 1: Define the polynomial $p(x) = x^3 - 6x^2 + 11x - 6$ as a vector of coefficients;
Step 2: Substitute $x = -y$ in the polynomial $p(x)$ to get the coefficients of the transformed polynomial;
Step 3: Find the roots of $p(x)$ and $q(y)$ using MATLABs ‘roots‘ function;
Step 4: Display the transformed polynomial and the roots of both polynomials.

Program

```
% Define the original polynomial p(x) = x^3 - 6x^2 + 11x - 6
   p_coefficients = [1, -6, 11, -6];
```

```
% Initialize an array to hold the transformed polynomial coefficients
q_coefficients = zeros(1, length(p_coefficients));

% Loop through each coefficient and substitute x = -y
for i = 1:length(p_coefficients)
power = length(p_coefficients) - i; % Degree of the term
q_coefficients(i) = p_coefficients(i) * (-1)^power;
end

% Define a symbolic variable y for displaying the polynomial
syms y;
% Convert the coefficients to a symbolic polynomial
q_polynomial = poly2sym(q_coefficients, y);

% Find the roots of the original polynomial p(x)
roots_p = roots(p_coefficients);

% Find the roots of the transformed polynomial q(y)
roots_q = roots(q_coefficients);

% Display the transformed polynomial
disp('The transformed polynomial q(y) is:');
disp(q_polynomial);

% Display the roots of both polynomials
disp('Roots of the original polynomial p(x):');
disp(roots_p);
disp('Roots of the transformed polynomial q(y):');
disp(roots_q);
```

Output

```
  The transformed polynomial q(y) is:
- y^3 - 6*y^2 - 11*y - 6

Roots of the original polynomial p(x):
    3.0000
    2.0000
    1.0000

Roots of the transformed polynomial q(y):
   -3.0000
   -2.0000
   -1.0000
```

Conclusion

This MATLAB code successfully transforms the polynomial $p(x) = x^3 - 6x^2 + 11x - 6$ into a new polynomial $q(y)$ whose roots are the negatives of the roots of $p(x)$. The roots of both the original and transformed polynomials are calculated and displayed, verifying the correctness of the transformation.

Exercise Problem

Transform the polynomial $p(x) = x^3 - 4x^2 + 5x - 1$ into another polynomial whose roots are the negatives of the roots of the original polynomial. Find and compare the roots of both polynomials.

Practical No. 6

Aim

To transform a polynomial equation into another equation whose roots are the reciprocals of the roots of the given polynomial equation, and find the roots of both polynomials using MATLAB.

Problem

Write a MATLAB program that takes a polynomial $p(x) = x^3 - 6x^2 + 11x - 6$, transforms it into a new polynomial $q(y)$ such that the roots of $q(y)$ are the reciprocals of the roots of $p(x)$, and finds the roots of both polynomials.

Theory

Let the given polynomial $p(x)$ be:

$$p(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 = 0.$$

We want to find a new polynomial $q(y)$, where the roots of $q(y)$ are the reciprocals of the roots of $p(x)$. Let $x \neq 0$ be a root of $p(x)$, and let y be a root of $q(y)$. If $y = \frac{1}{x}$, then $x = \frac{1}{y}$.

Substituting $x = \frac{1}{y}$ into the original polynomial $p(x)$, we get:

$$p\left(\frac{1}{y}\right) = a_n\left(\frac{1}{y}\right)^n + a_{n-1}\left(\frac{1}{y}\right)^{n-1} + \cdots + a_1\left(\frac{1}{y}\right) + a_0 = 0.$$

Multiplying through by y^n (the highest power of y) to eliminate denominators, we obtain:

$$a_n + a_{n-1}y + a_{n-2}y^2 + \cdots + a_1y^{n-1} + a_0y^n = 0.$$

Rewriting in standard polynomial form, we have:

$$q(y) = a_0y^n + a_1y^{n-1} + \cdots + a_{n-1}y + a_n = 0.$$

Thus, the coefficients of $q(y)$ are obtained by reversing the order of the coefficients of $p(x)$.

Algorithm

Step 1: Define the polynomial $p(x) = x^3 - 6x^2 + 11x - 6$ as a vector of coefficients;
Step 2: Reverse the order of the coefficients of $p(x)$ to get $q(y)$;
Step 3: Find the roots of $p(x)$ and $q(y)$ using MATLABs ‘roots‘ function;
Step 4: Display the transformed polynomial and the roots of both polynomials.

Program

```
% Define the original polynomial p(x) = x^3 - 6x^2 + 11x - 6
p_coefficients = [1, -6, 11, -6];

% Reverse the order of the coefficients to get q(y)
q_coefficients = fliplr(p_coefficients);

% Define a symbolic variable y for displaying the polynomial
syms y;
% Convert the coefficients to a symbolic polynomial
q_polynomial = poly2sym(q_coefficients, y);

% Find the roots of the original polynomial p(x)
roots_p = roots(p_coefficients);

% Find the roots of the transformed polynomial q(y)
roots_q = roots(q_coefficients);

% Display the transformed polynomial
disp(‘The transformed polynomial q(y) is:’);
disp(q_polynomial);
```

```
% Display the roots of both polynomials
disp('Roots of the original polynomial p(x):');
disp(roots_p);
disp('Roots of the transformed polynomial q(y):');
disp(roots_q);
```

Output

```
The transformed polynomial q(y) is:
y^3 - 11*y^2 + 6*y - 1

Roots of the original polynomial p(x):
1.0000
2.0000
3.0000

Roots of the transformed polynomial q(y):
1.0000
0.5000
0.3333
```

Conclusion

This MATLAB code successfully transforms the polynomial $p(x) = x^3 - 6x^2 + 11x - 6$ into a new polynomial $q(y)$ whose roots are the reciprocals of the roots of $p(x)$. The roots of both the original and transformed polynomials are calculated and displayed, verifying the correctness of the transformation.

Exercise Problem

Transform the polynomial $p(x) = x^3 - 4x^2 + 5x - 1$ into another polynomial whose roots are the reciprocals of the roots of the original polynomial. Find and compare the roots of both polynomials.

Practical No. 7

Aim

To verify De-Moivre's Theorem for a complex number in polar form and apply it to calculate powers and roots.

Problem

Verify De-Moivres Theorem for the complex number $z = 1 + i$ for $n = 5$. Specifically, calculate z^5 using De Moivres Theorem and compare it with the result obtained from direct computation using MATLAB.

Theory

De-Moivres Theorem states that for any complex number in polar form $z = r(\cos\theta + i\sin\theta)$ and any integer n, we have:

$$z^n = r^n\left(\cos(n\theta) + i\sin(n\theta)\right).$$

This theorem is useful for computing higher powers and roots of complex numbers.

Algorithm

Step 1: Define the complex number $z = 1 + i$;
Step 2: Convert z to polar form $z = r(\cos\theta + i\sin\theta)$, where $r = |z|$ and $\theta = \arg(z)$. Compute r and θ using the commands `abs(z)` and `angle(z)` respectively;
Step 3: For a given power n, calculate r^n and $n\theta$ and use De Moivre's Theorem to compute z^n;
Step 4: Calculate z^5 directly using Matlab's power function;
Step 5: Use the command `round` in MATLAB to round off both the real and imaginary parts of the complex number to three decimal places;
Step 6: Display the result using command `disp`.

Program

```
% Define the complex number
z = 1 + 1i;

% Step 1: Convert to polar form
r = abs(z);                % Modulus of z
theta = angle(z);          % Argument of z

% Step 2: Apply De Moivre's Theorem for n = 5
n = 5;
r_n = r^n;                 % Raise modulus to the power of n
theta_n = n * theta;       % Multiply argument by n

% Step 3: Calculate z^5 using De Moivre's Theorem
z_power_demov = r_n * (cos(theta_n) + 1i * sin(theta_n));
```

```
% Step 4: Calculate z^5 directly to verify
z_power_direct = z^n;
% Round off upto three decimal places
z_power_demov_round=round(z_power_demov,3);
z_power_direct_round=round(z_power_direct,3);
% Display the results
disp('The value of z^5 using De Moivre''s Theorem is:');
disp(z_power_demov_round);
disp('The value of z^5 using direct calculation is:');
disp(z_power_direct_round);
```

Output

```
The value of z^5 using De Moivre's Theorem is:
-4 + 4i
The value of z^5 using direct calculation is:
-4 + 4i
```

Conclusion

De Moivres Theorem applied to calculate z^5 for $z = 1 + i$, and the results from both the theorem and direct computation in MATLAB matched, validating the theorem's accuracy.

Exercise Problem

Verify De Moivres Theorem for the complex number $z = -1 + i$ for $n = 6$. Specifically, calculate z^6 using De Moivres Theorem and compare it with the result obtained from direct computation with help of MATLAB.

Practical No. 8

Aim

To use De Moivres Theorem to find the n-th roots of a complex number.

Problem

Find the fourth roots of $z = 16$ using De Moivres Theorem.

Theory

De Moivres Theorem states that for a complex number $z = r(\cos\theta + i\sin\theta)$, the n-th roots are given by:

$$z_k = r^{1/n}\left(\cos\left(\frac{\theta+2k\pi}{n}\right) + i\sin\left(\frac{\theta+2k\pi}{n}\right)\right).$$

where $k = 0, 1, \ldots, n-1$. This theorem is useful for finding the roots of complex numbers in polar form.

Algorithm

Step 1: Define $z = 16$ using `z = 16;`.
Step 2: Calculate modulus r with `r = abs(z);` and argument θ with `theta = angle(z);`.
Step 3: Set $n = 4$ using `n = 4;`.
Step 4: For $k = 0, 1, 2, 3$, adjust angle with `theta_k = (theta + 2 * k * pi) / n;` and calculate each root using $root\_k = r^{(1/n)} * (cos(theta\_k) + 1i * sin(theta\_k))$;.
Step 5: Round each root to three decimals with `root_k_round = round(root_k, 3);`.
Step 6: Display each root using `disp([‘Root ’, num2str(k+1),‘:  ’,num2str(root_k_round)`

Program

```
% Define the complex number (can be any complex number)
z = 16;  % Example: z = 1 + i

% Step 1: Convert to polar form
r = abs(z);                % Modulus of z
theta = angle(z);          % Argument (angle) of z

% Step 2: Set the value of n for n-th roots (change n as needed)
n = 4;  % Example: for 4th roots

% Step 3: Calculate each n-th root
disp([‘The ’, num2str(n), ’-th roots of z are:’]);
for k = 0:n-1
theta_k = (theta + 2 * k * pi) / n;   % Adjust argument for each root
root_k = r^(1/n) * (cos(theta_k) + 1i * sin(theta_k));  % Calculate root
root_k_round = round(root_k, 3);       % Round to 3 decimal places
disp([‘Root ’, num2str(k+1), ‘: ’, num2str(root_k_round)]);  % Display the root
end
```

Output

```
The 4-th roots of z are:
Root 1: 2
Root 2: 0+2i
Root 3: -2
Root 4: 0-2i
```

Conclusion

In this practical, we used De Moivre's Theorem to find the fourth roots of the complex number $z = 2$. The four fourth roots were successfully calculated using the theorem.

Exercise Problem

Using De Moivres Theorem, find the fifth roots of $z = 32\left(\cos\frac{\pi}{6} + i\sin\frac{\pi}{6}\right)$.

Practical No. 9

Aim

To compute the values of the direct circular functions $\sin z$, $\cos z$, and $\tan z$ for a given complex number $z = x + yi$ and analyze their behavior.

Problem

Compute the values of the circular functions $\sin z$, $\cos z$, and $\tan z$ for $z = 1 + 2i$ and $z = -1 + 3i$ using MATLAB. Then, compare these results with the values obtained from the theoretical formulas for $\sin z$, $\cos z$, and $\tan z$ where $z = x + yi$.

Theory

For a complex number $z = x + yi$, where x and y are real numbers, the circular functions are given by:

$$\sin z = \sin(x + yi) = \sin(x)\cosh(y) + i\cos(x)\sinh(y)$$

$$\cos z = \cos(x + yi) = \cos(x)\cosh(y) - i\sin(x)\sinh(y)$$

Algorithm

Step 1: Define the complex numbers $z = 1 + 2i$ and $z = -1 + 3i$;
Step 2: Compute $\sin z$, $\cos z$, and $\tan z$ using both MATLABs built-in functions and the theoretical formulas;
Step 3: Write the theoretical formulas of $\sin z$, $\cos z$, $\tan z$;
Step 4: Display the results using the `disp` command.

Program

```
 % Define the complex numbers
z1 = 1 + 2i;
z2 = -1 + 3i;

% Direct calculation using MATLAB’s built-in functions
sin_z1_direct = sin(z1);
cos_z1_direct = cos(z1);
tan_z1_direct = tan(z1);

sin_z2_direct = sin(z2);
cos_z2_direct = cos(z2);
tan_z2_direct = tan(z2);

% Calculation by formula (theoretical)
x1 = real(z1); y1 = imag(z1);
x2 = real(z2); y2 = imag(z2);

% Theoretical sin, cos, and tan for z1
sin_z1_formula = sin(x1) * cosh(y1) + 1i * cos(x1) * sinh(y1);
cos_z1_formula = cos(x1) * cosh(y1) - 1i * sin(x1) * sinh(y1);
tan_z1_formula = sin_z1_formula / cos_z1_formula;

% Theoretical sin, cos, and tan for z2
sin_z2_formula = sin(x2) * cosh(y2) + 1i * cos(x2) * sinh(y2);
cos_z2_formula = cos(x2) * cosh(y2) - 1i * sin(x2) * sinh(y2);
tan_z2_formula = sin_z2_formula / cos_z2_formula;

% Display results for z1
disp([‘For z = ’, num2str(z1), ’:’]);
disp([‘sin(z) (Direct) = ’, num2str(sin_z1_direct)]);
disp([‘sin(z) (By formula) = ’, num2str(sin_z1_formula)]);
disp([‘cos(z) (Direct) = ’, num2str(cos_z1_direct)]);
disp([‘cos(z) (By formula) = ’, num2str(cos_z1_formula)]);
disp([‘tan(z) (Direct) = ’, num2str(tan_z1_direct)]);
disp([‘tan(z) (By formula) = ’, num2str(tan_z1_formula)]);

% Display results for z2
disp([‘For z = ’, num2str(z2), ’:’]);
disp([‘sin(z) (Direct) = ’, num2str(sin_z2_direct)]);
disp([‘sin(z) (By formula) = ’, num2str(sin_z2_formula)]);
disp([‘cos(z) (Direct) = ’, num2str(cos_z2_direct)]);
disp([‘cos(z) (By formula) = ’, num2str(cos_z2_formula)]);
disp([‘tan(z) (Direct) = ’, num2str(tan_z2_direct)]);
disp([‘tan(z) (By formula) = ’, num2str(tan_z2_formula)]);
```

Output

```
For z = 1+2i:
sin(z) (Direct) = 3.1658+1.9596i
sin(z) (By formula) = 3.1658+1.9596i
cos(z) (Direct) = 2.0327-3.0519i
cos(z) (By formula) = 2.0327-3.0519i
tan(z) (Direct) = 0.033813+1.0148i
tan(z) (By formula) = 0.033813+1.0148i

For z = -1+3i:
sin(z) (Direct) = -8.47165+5.41268i
sin(z) (By formula) = -8.47165+5.41268i
cos(z) (Direct) = 5.43958+8.42975i
cos(z) (By formula) = 5.43958+8.42975i
tan(z) (Direct) = -0.0045171+1.0021i
tan(z) (By formula) = -0.0045171+1.0021i
```

Conclusion

This practical demonstrated how to compute the values of circular functions $\sin z$, $\cos z$, and $\tan z$ for complex numbers using MATLAB.

Exercise Problem

Compute the values of $\sin z$, $\cos z$, and $\tan z$ for complex numbers $z = 3 + 4i$, $z = -2 + i$, and $z = 0.5 - 1.5i$.

Practical No. 10

Aim

To plot the values of the direct circular function $\sin z$, $\cos z$, and $\tan z$ for a range of complex numbers z using MATLAB.

Problem

Plot $\sin z$ for a grid of complex numbers $z = x + yi$, where x and y vary from -2 to 2. Create separate surface plots for the real and imaginary parts of $\sin z$ to observe how they behave over this range of complex numbers.

Theory

In MATLAB, we can evaluate and plot the real and imaginary part of the complex functions like $\sin z$, $\cos z$, and $\tan z$ over a grid of complex numbers.

Algorithm

Step 1: Set the range for x and y from -2 to 2 with a step size of 0.1:

$$x = -2 : 0.1 : 2; \quad y = -2 : 0.1 : 2;$$

Step 2: Use `meshgrid` to create a grid for x and y, then form complex numbers $z = x + iy$:

$$[X, Y] = \texttt{meshgrid}(x, y); \quad Z = X + iY;$$

Step 3: Calculate $\sin z$ using `sin` function:

$$\text{sin\_Z} = \texttt{sin}(Z);$$

Step 4: Plot the real and imaginary part of $\sin z$ using command `surf`.

Program

```
% Define the range for real and imaginary parts
x = -2:0.1:2;
y = -2:0.1:2;
% Create a mesh grid of complex numbers
[X, Y] = meshgrid(x, y);
Z = X + 1i * Y;
% Calculate sin(Z)
```

```
sin_Z = sin(Z);
% Plot real part of sin(Z)
figure;
surf(X, Y, real(sin_Z));
title('Real Part of sin(z)');
xlabel('Real part of z');
ylabel('Imaginary part of z');
zlabel('Real part of sin(z)');
colormap jet;
colorbar;
shading interp;
% Plot imaginary part of sin(Z)
figure;
surf(X, Y, imag(sin_Z));
title('Imaginary Part of sin(z)');
xlabel('Real part of z');
ylabel('Imaginary part of z');
zlabel('Imaginary part of sin(z)');
colormap jet;
colorbar;
shading interp;
```

Output

The surface plots of the real and imaginary parts of $\sin z$ are shown below:

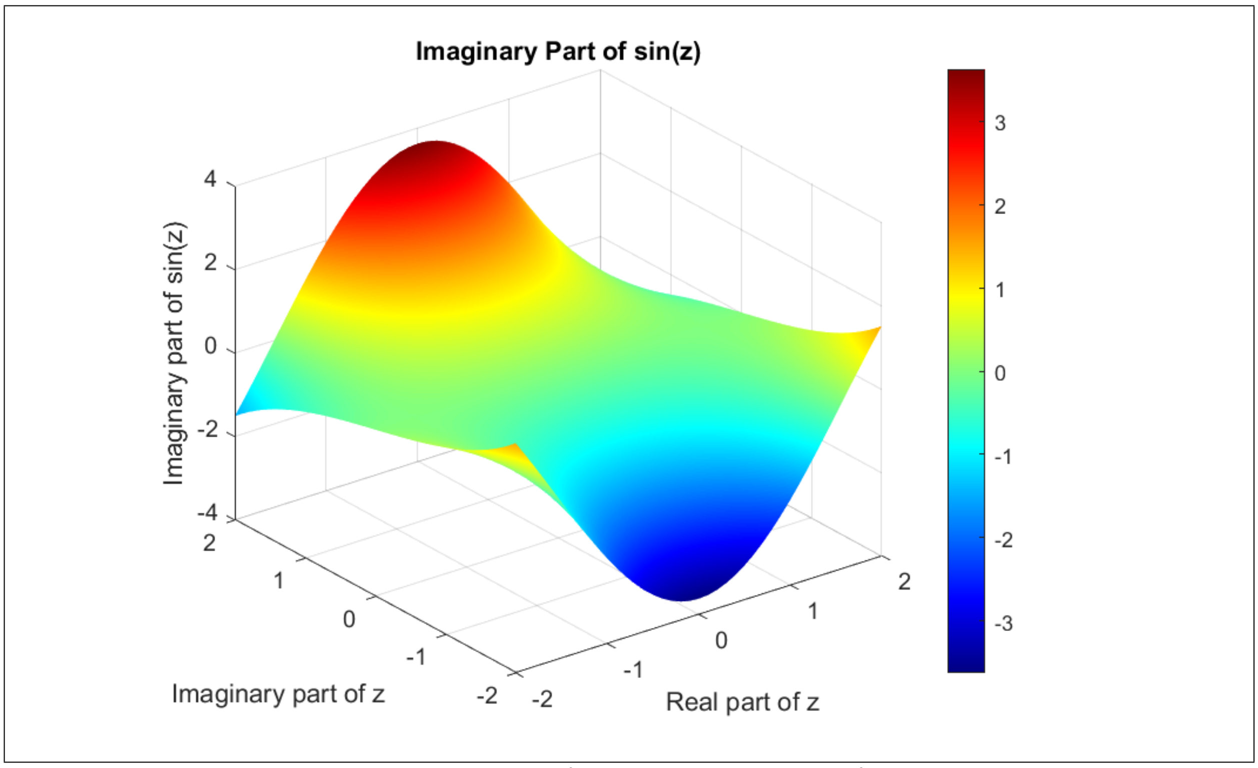


Figure 1.1: Imaginary part of $\sin z$ over the grid of complex numbers

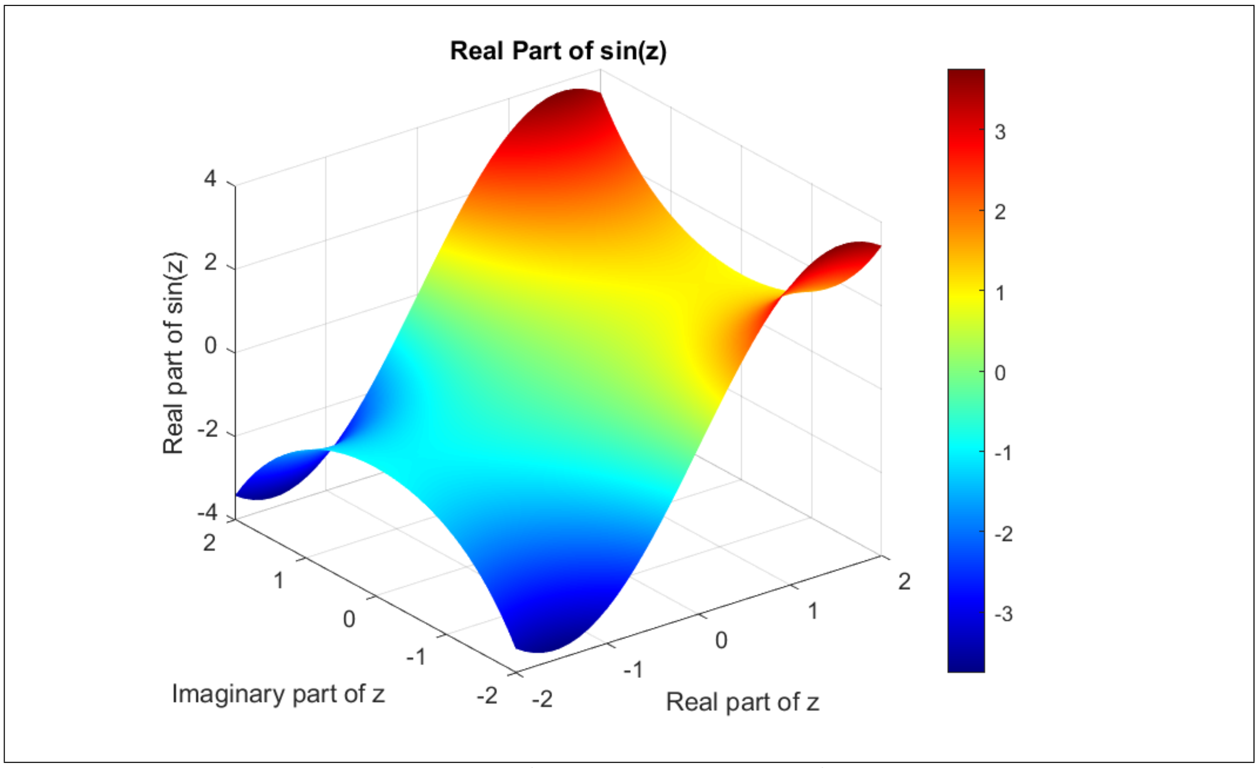


Figure 1.2: Real part of $\sin z$ over the grid of complex numbers

Conclusion

With the help of this program, we can plot surface plots of the real and imaginary parts of $\sin z$, which illustrate the behavior of the sine function over the defined range of complex values.

Exercise Problem

Plot the the real and imaginary parts of $\cos z$ for z in the same range.

Practical No. 11

Aim

To explore the relationship between complex exponential and circular functions $\sin z$ and $\cos z$ using Euler's formula. Compute the values of $\sin z$ and $\cos z$ for various complex numbers and verify them using Euler's identity.

Problem

Given a complex number $z = x + yi$, use Euler's formula to express $\sin z$ and $\cos z$ as complex exponentials:

$$e^{iz} = \cos z + i \sin z.$$

Compute $\sin z$ and $\cos z$ for the values $z = 2 + 3i$ and $z = 1 - 2i$ using both Eulers formula and MATLAB's built-in functions, and verify their consistency.

Theory

Euler's formula states that for any real number θ:

$$e^{i\theta} = \cos(\theta) + i \sin(\theta).$$

For complex numbers, this formula allows us to compute $\sin z$ and $\cos z$ for $z = x + yi$ as follows:

$$\cos z = \frac{e^{iz} + e^{-iz}}{2}, \quad \sin z = \frac{e^{iz} - e^{-iz}}{2i}.$$

Algorithm

Step 1: Define the complex numbers $z = 2 + 3i$ and $z = 1 - 2i$;
Step 2: Calculate $\sin z$ and $\cos z$ using Euler's formula:

$$\cos z = \frac{e^{iz} + e^{-iz}}{2}, \quad \sin z = \frac{e^{iz} - e^{-iz}}{2i};$$

Step 3: Calculate $\sin z$ and $\cos z$ using MATLABs functions, `sin(z)` and `cos(z)`;
Step 4: Compare the results to verify if the values from Eulers formula and MATLAB's function match.

Program

```
% Define the complex numbers
  clear all;
  close all;
 z1 = 2 + 3i;
 z2 = 1 - 2i;

 % Calculate sin(z) and cos(z) using Euler's formula
 sin_z1_euler = (exp(1i*z1) - exp(-1i*z1)) / (2i);
 cos_z1_euler = (exp(1i*z1) + exp(-1i*z1)) / 2;

 sin_z2_euler = (exp(1i*z2) - exp(-1i*z2)) / (2i);
 cos_z2_euler = (exp(1i*z2) + exp(-1i*z2)) / 2;

 % Calculate sin(z) and cos(z) using MATLAB's functions
 sin_z1_matlab = sin(z1);
 cos_z1_matlab = cos(z1);

 sin_z2_matlab = sin(z2);
 cos_z2_matlab = cos(z2);

 % Display results
 disp([‘For z = ’, num2str(z1), ’:’]);
 disp([‘sin(z) using Eulers formula = ’, num2str(sin_z1_euler)]);
 disp([‘sin(z) using direct MATLAB command = ’, num2str(sin_z1_matlab)]);
 disp([‘cos(z) using Eulers formula = ’, num2str(cos_z1_euler)]);
 disp([‘cos(z) using direct MATLAB command = ’, num2str(cos_z1_matlab)]);

 disp([‘For z = ’, num2str(z2), ’:’]);
 disp([‘sin(z) using Eulers formula = ’, num2str(sin_z2_euler)]);
 disp([‘sin(z) using direct MATLAB command = ’, num2str(sin_z2_matlab)]);
 disp([‘cos(z)using Eulers formula = ’, num2str(cos_z2_euler)]);
 disp([‘cos(z) using direct MATLAB command = ’, num2str(cos_z2_matlab)]);
```

Output

```
 For z = 2+3i:
sin(z) using Eulers formula = 9.1545-4.16891i
sin(z) using direct MATLAB command = 9.1545-4.16891i
cos(z) using Eulers formula = -4.18963-9.10923i
cos(z) using direct MATLAB command = -4.18963-9.10923i
For z = 1-2i:
sin(z) using Eulers formula = 3.1658-1.9596i
sin(z) using direct MATLAB command = 3.1658-1.9596i
cos(z)using Eulers formula = 2.0327+3.0519i
cos(z) using direct MATLAB command = 2.0327+3.0519i
```

Conclusion

This practical demonstrated the link between circular functions and complex exponentials using Eulers formula. The values of $\sin z$ and $\cos z$ obtained from Eulers formula matched MATLAB's built-in function outputs, confirming the relationship between exponential and circular functions in the complex domain.

Exercise Problem

For each complex number $z = 3+4i$ and $z = -2+3i$, calculate $\sin z$ and $\cos z$ using Eulers formula and MATLAB functions, and verify the results.

Practical No. 12

Aim

To compute specific mathematical expressions involving the real and imaginary parts of complex trigonometric functions using MATLAB.

Problem

Given $\tan(\alpha + i\beta) = x + iy$, where α and β are real numbers, and x and y are the real and imaginary parts of $\tan(\alpha + i\beta)$, compute the value of $x^2 + y^2 + 2x\cot(2\alpha)$ using MATLAB with $\alpha = \frac{\pi}{4}$, and $\beta = 1$.

Theory

By solving the expresion theoretically $\tan(\alpha+i\beta) = x+iy$, we will get $x^2+y^2+2x\cot(2\alpha) = 1$.

Algorithm

Step 1: Define the valurs of α and β;
Step 2: Calculate $\tan(\alpha + i\beta)$ by using MATLAB's built-in `tan` function;
Step 3: Seperate the real and imaginary parts by Using MATLABs `real` and `imag` functions to get x and y, the real and imaginary parts of the complex tangent function;
Step 4: Compute the desired expression $x^2 + y^2 + 2x\cot(2\alpha)$;
Step 5: Display the calculated expression by using `disp` command.

Program

```
% Define the real and imaginary parts of the complex number (alpha, beta)
alpha = pi/4;  % example value for alpha
beta = 1;      % example value for beta

% Calculate the tangent of (alpha + i*beta)
z = tan(alpha + 1i*beta);

% Extract the real and imaginary parts of tan(alpha + i*beta)
x = real(z);
y = imag(z);

% Compute the desired expression: x^2 + y^2 + 2x * cot(2*alpha)
result = x^2 + y^2 + 2*x * cot(2*alpha);

% Display the result
disp(['The value of x^2 + y^2 + 2x * cot(2*alpha) is: ', num2str(result)]);
```

Expected Output

```
The value of x^2 + y^2 + 2x * cot(2*alpha) is: 1
```

Conclusion

The code provide the correct value of the expression.

Exercise Problem

Given $\sin(\phi + i\psi) = a + ib$, where ϕ and ψ are real numbers, and a and b are the real and imaginary parts of $\sin(\phi + i\psi)$, compute the value of $a^2 + b^2 + 2a\cos(2\phi)$ using MATLAB with $\phi = \frac{\pi}{6}$ and $\psi = 1$.

Practical No. 13

Aim

To compute the hyperbolic functions $\sinh z$, $\cosh z$, and $\tanh z$ for a given complex number $z = x + iy$.

Problem

Compute the values of the hyperbolic functions $\sinh z$, $\cosh z$, and $\tanh z$ for $z = 2+2i$ and $z = -5+3i$ using MATLAB.

Theory

The hyperbolic functions for a complex number $z = a + ib$ are defined as follows:

$$\sinh(z) = \frac{e^z - e^{-z}}{2};$$

$$\cosh(z) = \frac{e^z + e^{-z}}{2};$$

$$\tanh(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^z - e^{-z}}{e^z + e^{-z}}.$$

Algorithm

Step1 : Define the complex numbers $z_1 = 2 + 2i$ and $z_2 = -5 + 3i$;
Step 2: Compute $\sinh z$, $\cosh z$, and $\tanh z$ using MATLAB's built-in functions with the commands `sinh(z)`, `cosh(z)`, and `tanh(z)` respectively;
Step 3: Display the results using the `disp` command.

Program

```
% Define the complex numbers
z1 = 2 + 2i;
z2 = -5 + 3i;

% Calculate and display for z1
fprintf('For z = 2 + 2i:\n');
sinh_z1 = sinh(z1);
cosh_z1 = cosh(z1);
tanh_z1 = tanh(z1);
```

```
% Display results for z1
disp([‘sinh(z): ’, num2str(real(sinh_z1)), ‘ + ’, num2str(imag(sinh_z1)), ’i’
disp([‘cosh(z): ’, num2str(real(cosh_z1)), ‘ + ’, num2str(imag(cosh_z1)), ’i’
disp([‘tanh(z): ’, num2str(real(tanh_z1)), ‘ + ’, num2str(imag(tanh_z1)), ’i’

% Calculate and display for z2
fprintf(’\nFor z = -5 + 3i:\n’);
sinh_z2 = sinh(z2);
cosh_z2 = cosh(z2);
tanh_z2 = tanh(z2);

% Display results for z2
disp([‘sinh(z): ’, num2str(real(sinh_z2)), ‘ + ’, num2str(imag(sinh_z2)), ’i’
disp([‘cosh(z): ’, num2str(real(cosh_z2)), ‘ + ’, num2str(imag(cosh_z2)), ’i’
disp([‘tanh(z): ’, num2str(real(tanh_z2)), ‘ + ’, num2str(imag(tanh_z2)), ’i’
```

Output

```
For z = 2 + 2i:
sinh(z): -1.5093 + 3.421i
cosh(z): -1.5656 + 3.2979i
tanh(z): 1.0238 + -0.028393i

For z = -5 + 3i:
sinh(z): 73.4606 + 10.4725i
cosh(z): -73.4673 + -10.4716i
tanh(z): -0.99991 + -2.5369e-05i
```

Conclusion

With this MATLAB code, we can compute the hyperbolic function values for a given complex number.

Exercise Problem

Compute the value of the function $\sinh z + 8\cosh z - 2\tanh^2 z$ for $z = i$ and $z = -8 + 3i$ using MATLAB.

Practical No. 14

Aim

To plot the real and imaginary parts of the hyperbolic sine function for a complex number z using MATLAB.

Problem

Write a MATLAB program to plot the real and imaginary parts of the hyperbolic sine function $\sinh(z)$ for a complex variable $z = x + iy$, where x and y are real numbers. Plot the real and imaginary parts separately for x in the range $[-10, 10]$ and y in the range $[-10, 10]$.

Theory

The hyperbolic sine function for a complex number $z = x + iy$ (where i is the imaginary unit) is defined as:

$$\sinh(z) = \frac{e^z - e^{-z}}{2}.$$

Substituting $z = x + iy$, we get:

$$\sinh(x + iy) = \frac{e^{x+iy} - e^{-x-iy}}{2}.$$

Using the properties of exponential and Euler's formula, we can break it down into its real and imaginary parts:

$$\sinh(x + iy) = \sinh(x)\cos(y) + i\cosh(x)\sin(y).$$

Algorithm

Step 1: Define the range for x and y values for which the graph will be plotted;
Step 2: Compute the real and imaginary parts of $\sinh(x + iy)$ using the above formulas;
Step 3: Plot the real and imaginary parts separately using MATLAB's 'surf' command;
Step 4: Add appropriate labels, titles, and grid to the graph for better visualization.

Program

```
% Define the range for x and y values
[x, y] = meshgrid(linspace(-10, 10, 400), linspace(-10, 10, 400));

% Calculate the real and imaginary parts of sinh(x + iy)
real_part = sinh(x) .* cos(y);    % Real part: sinh(x) * cos(y)
```

```
imag_part = cosh(x) .* sin(y);    % Imaginary part: cosh(x) * sin(y)

% Plot the real part of sinh(x + iy)
figure;
surf(x, y, real_part);   % Create a 3D surface plot
colormap jet;            % Set the colormap to 'jet' for a colorful plot
shading interp;          % Apply interpolation shading for smooth surface
title('Real Part of sinh(x + iy)', 'FontSize', 14);
xlabel('x', 'FontSize', 12);
ylabel('y', 'FontSize', 12);
zlabel('Real Part', 'FontSize', 12);
colorbar;                % Display color bar to show value scale
grid on;                 % Enable grid for better visibility
lighting gouraud;        % Apply Gouraud lighting for smoother shading
view(3);                 % Set the default 3D view angle

% Plot the imaginary part of sinh(x + iy)
figure;
surf(x, y, imag_part);   % Create a 3D surface plot
colormap hot;            % Set the colormap to 'hot' for a different color theme
shading interp;          % Apply interpolation shading for smooth surface
title('Imaginary Part of sinh(x + iy)', 'FontSize', 14);
xlabel('x', 'FontSize', 12);
ylabel('y', 'FontSize', 12);
zlabel('Imaginary Part', 'FontSize', 12);
colorbar;                % Display color bar to show value scale
grid on;                 % Enable grid for better visibility
lighting gouraud;        % Apply Gouraud lighting for smoother shading
view(3);                 % Set the default 3D view angle
```

Output

The output will consist of two 3D surface plots which are given by Figure 1.3, 1.4

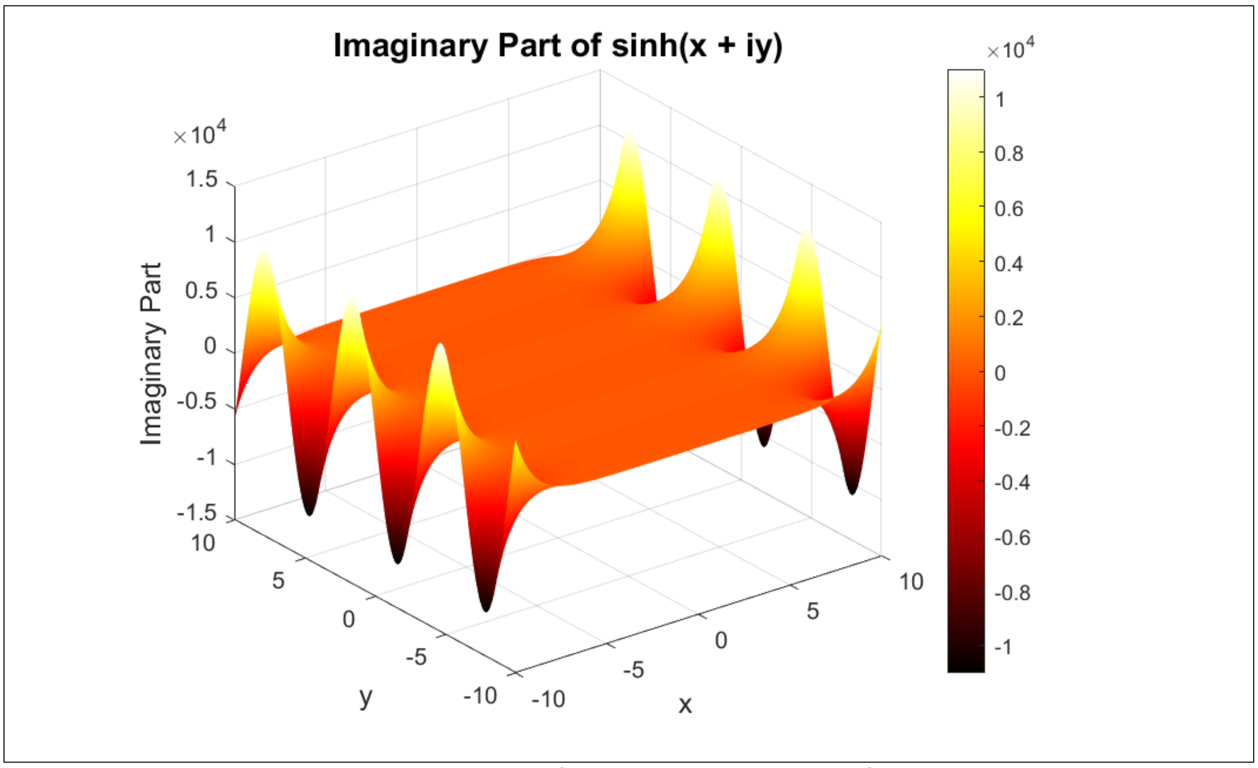


Figure 1.3: Imaginary part of $\sinh z$ over the grid of complex numbers

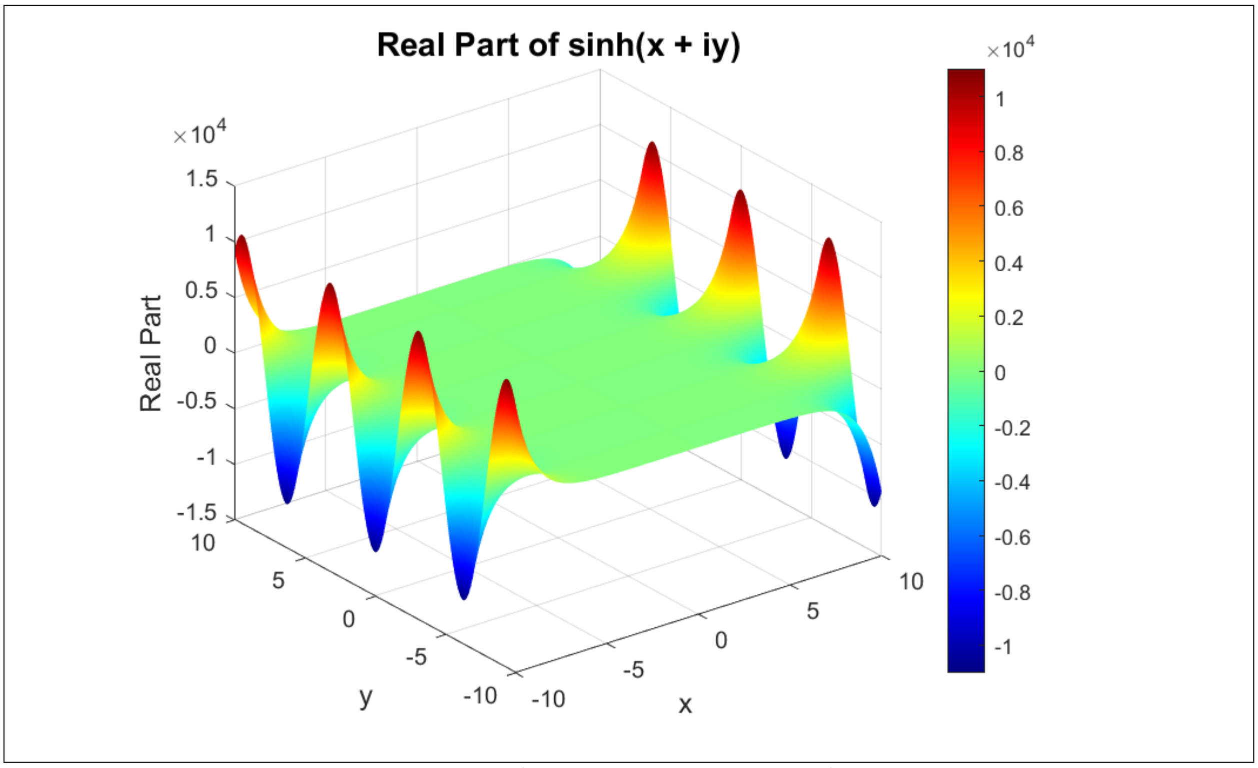


Figure 1.4: Real part of $\sinh z$ over the grid of complex numbers

Conclusion

This MATLAB code successfully computes and plots the real and imaginary parts of the hyperbolic sine function for a complex number $z = x + iy$ over specified ranges of x and y. The plots provide a clear visualization of how the real and imaginary components of $\sinh(z)$ behave in the complex plane, showcasing their dependence on both the real and imaginary parts of z.

Exercise Problem

Modify the program to plot the real and imaginary parts of the hyperbolic cosine function $\cosh(z)$ for the same ranges of x and y. Compare the plots of $\sinh(z)$ and $\cosh(z)$, and describe their similarities and differences.

Practical No. 15

Aim

To compute the inverse circular functions $\sin^{-1} z$, $\cos^{-1} z$, and $\tan^{-1} z$ for a given complex number $z = x + iy$.

Problem

Compute the values of the inverse circular functions $\sin^{-1} z$, $\cos^{-1} z$, and $\tan^{-1} z$ for $z = 2 + 2i$ and $z = -5 + 3i$ using MATLAB.

Theory

For a complex number $z = x+iy$, the inverse trigonometric functions are defined as follows:

- If $\sin(\alpha + i\beta) = x + iy$, then $\alpha + i\beta$ is called the inverse sine of z, denoted $\sin^{-1} z$.
- If $\cos(\alpha + i\beta) = x + iy$, then $\alpha + i\beta$ is called the inverse cosine of z, denoted $\cos^{-1} z$.
- If $\tan(\alpha+i\beta) = x+iy$, then $\alpha+i\beta$ is called the inverse tangent of z, denoted $\tan^{-1} z$.

These inverse functions return the complex angle $\alpha + i\beta$ whose sine, cosine, or tangent equals the given complex number $z = x + iy$.

Algorithm

Step1: Define the complex numbers $z_1 = 2 + 2i$ and $z_2 = -5 + 3i$;
Step2: Compute $\sin^{-1} z$, $\cos^{-1} z$, and $\tan^{-1} z$ using MATLABs built-in functions;
Step3: Display the results using the `disp` command.

Program

```
 clear all;
close all;
% Define the complex numbers
z1 = 2 + 2i;
z2 = -5 + 3i;

% Calculate and display for z1
disp('For z = 2 + 2i:');

asin_z1 = asin(z1);
acos_z1 = acos(z1);
atan_z1 = atan(z1);

% Display results for z1
disp([‘sin^-1(z)$: ',num2str(asin_z1)]);
disp([‘cos^-1(z): ', num2str(acos_z1)]);
disp([‘tan^-1(z): ', num2str(atan_z1)]);

% Calculate and display for z2
disp(‘For z = -5 + 3i:');
asin_z2 = asin(z2);
acos_z2 = acos(z2);
atan_z2 = atan(z2);

% Display results for z2
disp([‘sin^-1(z): ',num2str(asin_z2)]);
disp([‘cos^-1(z): ', num2str(acos_z2)]);
disp([‘tan^{-1}(z): ', num2str(atan_z2)]);
```

Output

```
For z = 2 + 2i:
For z = 2 + 2i:
sin^-1(z)$: 0.75425+1.7343i
cos^-1(z): 0.81655-1.7343i
tan^-1(z): 1.3112+0.23888i
For z = -5 + 3i:
sin^-1(z): -1.0238+2.4529i
cos^-1(z): 2.5946-2.4529i
tan^-1(z): -1.4237+0.086569i
```

Conclusion

With this MATLAB code, we can compute the values of the inverse circular functions for a given complex number.

Exercise Problem

Compute the value of the function $5\sin^{-1} z + 18\cos^{-1} z - 2\tan^{-1} z$ for $z = i$ and $z = -8+3i$ using MATLAB.

Practical No. 16

Aim

To compute the inverse hyperbolic functions $\sinh^{-1} z$, $\cosh^{-1} z$, and $\tanh^{-1} z$ for a given complex number $z = x + iy$ and verify their results using logarithmic expressions.

Problem

Write a matlab code to compute the values of the inverse hyperbolic functions $\sinh^{-1} z$, $\cosh^{-1} z$, and $\tanh^{-1} z$ for any given value of z. Verify the results by comparing them with the logarithmic definitions:

$$\sinh^{-1}(z) = \ln(z + \sqrt{z^2+1}),$$

$$\cosh^{-1}(z) = \ln(z + \sqrt{z^2-1}),$$

$$\tanh^{-1}(z) = \frac{1}{2}\ln\left(\frac{1+z}{1-z}\right).$$

Theory

For a complex number $z = x + iy$, the inverse hyperbolic functions are defined as follows:

- If $\sinh(\alpha + i\beta) = x + iy$, then $\alpha + i\beta$ is called the inverse hyperbolic sine of z, denoted $\sinh^{-1} z$.
- If $\cosh(\alpha + i\beta) = x + iy$, then $\alpha + i\beta$ is called the inverse hyperbolic cosine of z, denoted $\cosh^{-1} z$.
- If $\tanh(\alpha + i\beta) = x + iy$, then $\alpha + i\beta$ is called the inverse hyperbolic tangent of z, denoted $\tanh^{-1} z$.

The inverse hyperbolic functions can also be expressed in logarithmic forms:

$$\sinh^{-1}(z) = \ln(z + \sqrt{z^2+1}),$$

$$\cosh^{-1}(z) = \ln(z + \sqrt{z^2-1}),$$

$$\tanh^{-1}(z) = \frac{1}{2}\ln\left(\frac{1+z}{1-z}\right).$$

Algorithm

Step1: Define the complex number z using the `input` command;
Step2: Compute $\sinh^{-1} z$, $\cosh^{-1} z$, and $\tanh^{-1} z$ using MATLABs built-in functions;
Step3: Verify the results by calculating each inverse function using their logarithmic definitions;
Step4: Display the results and verification outcomes using the `disp` command.

Program

```
% Step 1: Prompt user for complex number input
z = input('Enter a complex number in the form x + yi: ');

% Step 2: Compute inverse hyperbolic functions using MATLABs built-in functions
asinh_z = asinh(z);   % Inverse hyperbolic sine
acosh_z = acosh(z);   % Inverse hyperbolic cosine
atanh_z = atanh(z);   % Inverse hyperbolic tangent

% Step 3: Display the computed results
disp(‘Computed Inverse Hyperbolic Function Values for z:’);
disp([‘asinh(z): ’, num2str(asinh_z)]);
disp([‘acosh(z): ’, num2str(acosh_z)]);
disp([‘atanh(z): ’, num2str(atanh_z)]);

% Step 4: Verification using logarithmic expressions
disp(‘Verification of Inverse Hyperbolic Functions using Logarithmic
Expressions:’);
disp([‘asinh(z) calculated as ln(z + sqrt(z^2 + 1)): ’, num2str(log(z + sqrt(z^2
+ 1)))]);
disp([‘acosh(z) calculated as ln(z + sqrt(z^2 - 1)): ’, num2str(log(z + sqrt(z^z
- 1)))]);
disp([‘atanh(z) calculated as 0.5 * ln((1 + z) / (1 - z)): ’, num2str(0.5 * log
 ((1 + z) / (1 - z)))]);
```

Output

```
Enter a complex number in the form x + yi: 2+5i
Computed Inverse Hyperbolic Function Values for z:
asinh(z): 2.3705+1.1842i
acosh(z): 2.383+1.1961i
atanh(z): 0.067066+1.3993i
Verification of Inverse Hyperbolic Functions using Logarithmic Expressions:
asinh(z) calculated as ln(z + sqrt(z^2 + 1)): 2.3705+1.1842i
acosh(z) calculated as ln(z + sqrt(z^2 - 1)): 2.383+1.1961i
atanh(z) calculated as 0.5 * ln((1 + z) / (1 - z)): 0.067066+1.3993i
```

Conclusion

This MATLAB code successfully computes the values of the inverse hyperbolic functions $\sinh^{-1} z$, $\cosh^{-1} z$, and $\tanh^{-1} z$ for a given complex number z. The results are verified using their logarithmic expressions, confirming the accuracy of MATLABs built-in functions.

Exercise Problem

Compute the value of the function

$$5\sinh^{-1} z + 18\cosh^{-1} z - 2\tanh^{-1} z$$

for $z = 9 + 5i$ and $z = -8 + 3i$ using MATLAB.

Practical No. 17

Aim

To compute the logarithm of a complex number $z = x + iy$ and verify the logarithmic formula $\ln z = \ln|z| + i\arg(z)$ using MATLAB.

Problem

Write a MATLAB program that takes a complex number as input from the user, calculates the natural logarithm $\ln z$, base-10 logarithm $\log_{10} z$, and base-2 logarithm $\log_2 z$ for the given number, and verifies that $\ln z = \ln|z| + i\arg(z)$ holds true.

Theory

For a complex number $z = x + iy$, the logarithmic functions are defined as:

- The natural logarithm $\ln z$ is given by:

$$\ln z = \ln |z| + i \arg(z),$$

  where $|z| = \sqrt{x^2 + y^2}$ and $\arg(z) = \tan^{-1}(y/x)$.

- The base-10 logarithm $\log_{10} z$ is computed as:

$$\log_{10} z = \frac{\ln z}{\ln 10}.$$

- The base-2 logarithm $\log_2 z$ is computed as:

$$\log_2 z = \frac{\ln z}{\ln 2}.$$

Verification involves calculating $\ln |z| + i \arg(z)$ and comparing it with MATLABs built-in $\ln z$ for accuracy.

Algorithm

Step 1: Prompt the user to input a complex number $z = x + iy$;
Step 2: Calculate $\ln z$, $\log_{10} z$, and $\log_2 z$;
Step 3: Verify $\ln z = \ln |z| + i \arg(z)$ by computing each side and comparing;
Step 4: Display all results.

Program

```
% Prompt user for complex number input
z = input('Enter a complex number in the form x + yi: ');

% Calculate logarithmic values
ln_z = log(z);            % Natural logarithm
log10_z = log10(z);        % Base-10 logarithm
log2_z = log(z) / log(2); % Base-2 logarithm

% Display logarithmic results
disp(‘Computed Logarithmic Values:’);
disp([‘ln(z): ’, num2str(ln_z)]);
disp([‘log10(z): ’, num2str(log10_z)]);
disp([‘log2(z): ’, num2str(log2_z)]);
```

```
% Verification of logarithmic formula
modulus_z = abs(z);            % Compute |z|
argument_z = angle(z);         % Compute arg(z)
ln_z_formula = log(modulus_z) + 1i * argument_z; % Formula-based calculation

% Display verification results
disp('Verification of ln(z) = ln|z| + i * arg(z):');
disp(['ln(z) by MATLAB inbuilt function: ', num2str(ln_z)]);
disp(['ln(z) using formula: ', num2str(ln_z_formula)]);
if abs(ln_z - ln_z_formula) < 1e-10
disp('Verification successful: Both methods agree.');
else
disp('Verification failed: Results do not match.');
end
```

Output

```
Enter a complex number in the form x + yi: 4+3i
Computed Logarithmic Values:
ln(z): 1.6094+0.6435i
log10(z): 0.69897+0.27947i
log2(z): 2.3219+0.92838i
Verification of ln(z) = ln|z| + i * arg(z):
ln(z) by MATLAB inbuilt function: 1.6094+0.6435i
ln(z) using formula: 1.6094+0.6435i
Verification successful: Both methods agree.
```

Conclusion

This MATLAB code successfully computes the natural and base-specific logarithms for any given complex number and verifies that $\ln z = \ln|z| + i\arg(z)$ holds, supporting the theoretical formulation.

Exercise Problem

Compute $2\ln z + 3\log_{10} z - \log_2 z$ for $z = -3 + 7i$ and $z = 4 - 5i$ using MATLAB.

Practical No. 18

Aim

To approximate the value of π using the Gregory series for different values of N, and plot the error in approximation as a function of N using MATLAB.

Problem

Write a MATLAB program to calculate an approximation of π using the Gregory series:

$$\pi \approx 4 \sum_{k=0}^{N} \frac{(-1)^k}{2k+1}$$

for a given N.

Specifically, compute the approximation for N ranging from 10000 to 150000 with a step size of 1000. Plot a graph showing the error (the absolute difference between the approximation and MATLAB's built-in value of π) as a function of N.

Theory

The Gregory series is an infinite series that approximates π as:

$$\pi = 4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots\right)$$

The accuracy of this approximation improves as N increases, but convergence is slow. Plotting the error as a function of N reveals the convergence pattern and shows how the approximation stabilizes as N grows.

Algorithm

Step 1: Define N values (starting from 10000 and incrementing by 100 up to the maximum N):

```
N_values = 10000:1000:max_N;
```

Step 2: Initialize an empty array to store the errors:

```
errors = zeros(size(N_values));
```

Step 3: For each N in the defined range calculate the Gregory series approximation of π using the formula.
Step 4: Calculate the absolute error with MATLABs built-in π:

```
errors(i) = abs(pi - pi_approx);
```

Step 5: Plot the error as a function of N using a logarithmic scale.

Program

```
% Define a range of N values (from 100 up to a desired maximum in steps of 100)
max_N = 150000; % Set the maximum value of N
N_values = 10000:100:max_N;
errors = zeros(size(N_values));

% Calculate Gregory series approximation and error for each N
for i = 1:length(N_values)
N = N_values(i);
pi_approx = 0;
for k = 0:N
pi_approx = pi_approx + ((-1)^k) / (2*k + 1);
end
pi_approx = 4 * pi_approx; % Multiply by 4 as per series definition
errors(i) = abs(pi - pi_approx); % Absolute error with MATLABs pi
end

% Plot the error as a function of N
figure;
semilogy(N_values, errors, '-*'); % semilogy for better visualization
xlabel('Number of Terms (N)');
ylabel('Absolute Error');
title('Error in Gregory Series Approximation of \pi');
grid on;
```

Output

After running the program, the plot will display how the absolute error decreases with increasing N.

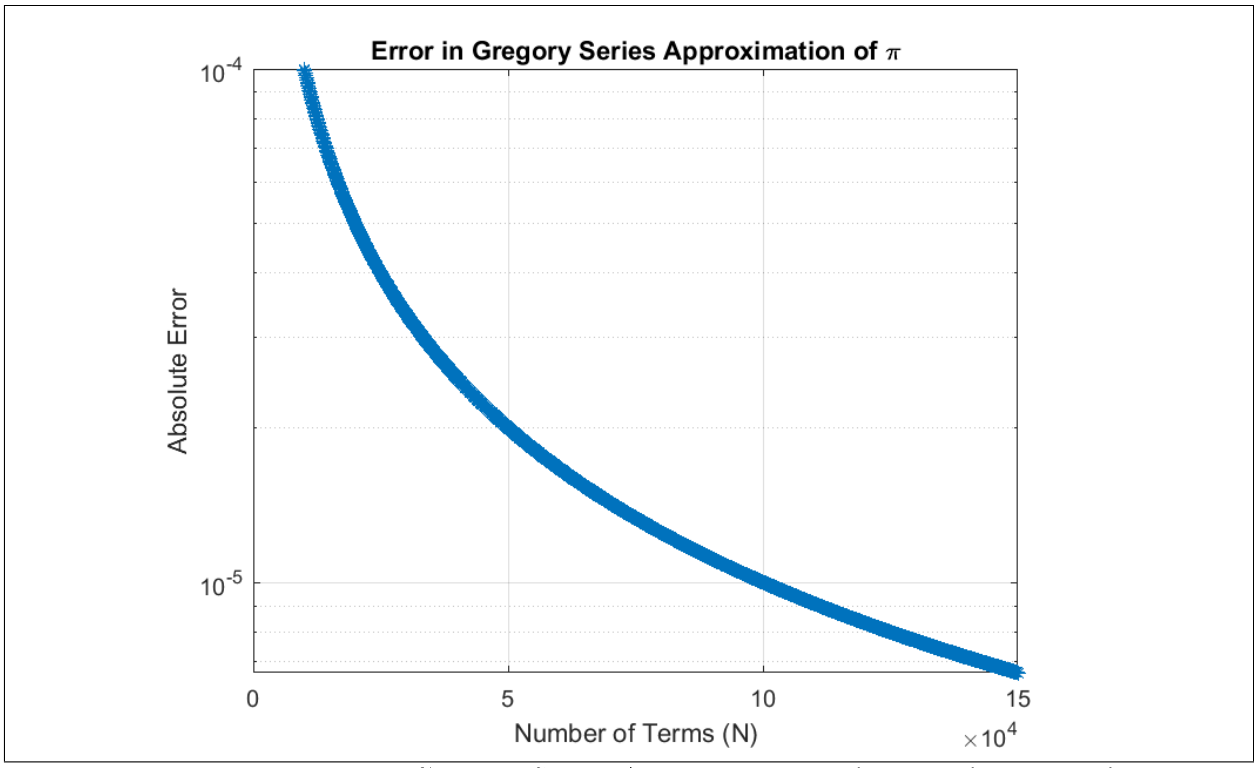


Figure 1.5: Error in Gregory Series Approximation of π as a function of N

Conclusion

The plot reveals that the Gregory series slowly converges to π as N increases. Large values of N are required to achieve a low error, illustrating the slow convergence of the series.

Exercise Problem

Using the program, find the error in the approximation of π for $N = 10000$.

Matrices

Practical No. 19

Aim

To write a program to find the determinant of a given matrix.

Problem

Write a program to compute the determinant of the matrix

$$A = \begin{pmatrix} 2 & 3 & 1 \\ 4 & 1 & 2 \\ 3 & 2 & 5 \end{pmatrix}.$$

Theory

The determinant of a matrix is a scalar value that can be computed from the elements of a square matrix. It is used in many areas of linear algebra, including the solution of system of linear equations, analysis of linear transformation, and more.

Algorithm

Step1: Define the matrix A;
Step2: Compute the determinant using the command `det(A)`;
Step3: Output the determinant using the command `disp`.

Program

```
clc
clear
% Step 1: Define the matrix A
A = [2, 3, 1; 4, 1, 2; 3, 2, 5];

% Step 2: Compute the determinant of A
det_A = det(A);

% Step 3: Output the determinant
disp('Determinant of matrix A:');
disp(det_A);
```

Output

```
Determinant of matrix A:
-29
```

Conclusion

The program correctly computes the determinant of the given matrix.

Practical No. 20

Aim

To write a program to find the inverse of the given matrix.

Problem

Write a program to compute the inverse of a matrix

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 5 & 6 & 0 \end{pmatrix}.$$

Theory

The inverse of a matrix A is another matrix A^{-1} such that

$$AA^{-1} = A^{-1}A = I$$

where, I is the identity matrix. Not all matrices have an inverse, a matrix must be square and have a non-zero determinant.

Algorithm

Step1: Define the matrix A;
Step2: Compute the inverse of A using the command `inv(A)`;
Step3: Output the inverse using the command `disp`.

Program

```
clc
clear
% Step 1: Define the matrix
A = [1, 2, 3; 0, 1, 4; 5, 6, 0];

% Step 2: Compute the inverse
inv_A = inv(A);

% Step 3: Output the inverse
disp('Inverse of matrix A:');
disp(inv_A);
```

Output

```
Inverse of matrix A:
   -24    18     5
    20   -15    -4
    -5     4     1
```

Conclusion

The program correctly computes the inverse of the given matrix.

Practical No. 21

Aim

To write a program to find the rank of a given matrix.

Problem

Write a program to compute the rank of the matrix

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}.$$

Theory

The rank of a matrix is the dimension of the vector space generated by its rows or columns. It can be computed using the command `rank`.

Algorithm

Step1: Define the matrix A;
Step2: Compute the rank using the command `rank;`
Step3: Output the rank using the command `disp`.

Program

```
clc
clear
% Step 1: Define the matrix
A = [1, 2, 3; 4, 5, 6; 7, 8, 9];

% Step 2: Compute the rank
rank_A = rank(A);

% Step 3: Output the rank
disp('Rank of matrix A:');
disp(rank_A);
```

Output

```
Rank of matrix A:
2
```

Conclusion

The program correctly computes the rank of the given matrix.

Practical No. 22

Aim

To write a program to find the eigenvalues and eigenvectors of a given matrix.

Problem

Write a program to compute the eigenvalues and eigenvectors of matrix

$$A = \begin{pmatrix} 4 & -2 \\ 1 & 1 \end{pmatrix}.$$

Theory

Eigenvector of a square matrix A is a non-zero vector v for any scalar λ such that :

$$A\mathbf{v} = \lambda\mathbf{v}$$

where, λ is the eigenvalue corresponding to the eigenvector $\mathbf{v}$.

Algorithm

Step1: Define the matrix A;
Step2: Compute the eigenvectors and eigenvalues using the command `eig(A)`;
Step3: Output the eigenvalues and eigenvectors using command `disp`.

Program

```
clc
clear
% Step 1: Define the matrix A
A = [4, -2; 1, 1];

% Step 2: Compute eigenvalues and eigenvectors of A
[eigenvectors, eigenvalues] = eig(A);

% Step 3: Output the eigenvalues and eigenvectors
disp('Eigenvalues:');
disp(diag(eigenvalues));
disp('Eigenvectors:');
disp(eigenvectors);
```

Output

```
Eigenvalues:
     3
     2

Eigenvectors:
    0.8944    0.7071
    0.4472   -0.7071
```

Conclusion

The program correctly computes the eigenvalues and eigenvectors of the given matrix.

Practical No. 23

Aim

To write a program to calculate the matrix multiplication.

Problem

Write a MATLAB program to verify the given matrices are compatible for multiplication. If these are compatible then find their multiplication

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \; and \; B = \begin{pmatrix} 2 & 0 \\ 1 & 2 \end{pmatrix}.$$

Theory

Matrix multiplication is a binary operation that produces a matrix from two matrices. If A is an $m \times n$ matrix and B is an $n \times p$ matrix, the product AB is an $m \times p$ matrix where each element is computed as:

$$(AB)_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

Algorithm

Step1: Define matrices A and B;
Step2: Verify the dimensions of both matrices multiplication using command `size(A)`;
Step3: Compute the product matrix $C = AB$;
Step4: Output the result using the command `disp`.

Program

```
clc
clear
% Step 1: Define the matrices
A = [1, 2; 3, 4];
B = [2, 0; 1, 2];

% Step 2: Verify dimensions
[m, n] = size(A);
[p, q] = size(B);
if n ~= p
disp('Matrices are not compatible for multiplication');
end
% Step 3: Compute the product
C = A * B;
% Step 4: Output the result
disp('Matrix A:');
disp(A);
disp('Matrix B:');
disp(B);
disp('Product Matrix C = AB:');
disp(C);
```

Output

```
Matrix A:
     1     2
     3     4

Matrix B:
     2     0
     1     2

Product Matrix C = AB:
     4     4
```

```
    10      8
```

Conclusion

The program correctly performs matrix multiplication and outputs the resulting matrix.

Practical No. 24

Aim

To determine the consistency and solve a system of linear equations using rank method.

Problem

Write a program to determine the consistency and solve the system of linear equations given by

$$\begin{aligned} x + 2y + z &= 10, \\ 2x + 4y + 2z &= 20, \\ x + y + z &= 10. \end{aligned}$$

Theory

The rank of a matrix is the maximum number of linearly independent row or column vectors in the matrix. For a system of linear equations $Ax = b$, the system is consistent if and only if the rank of the matrix A is equal to the rank of the augmented matrix $[A|b]$. Also, if the rank is equal to the number of variables, the system has a unique solution. If the rank is less than the number of variables, the system has infinitely many solutions.

Algorithm

Step1: Define the coefficient matrix A and the right-hand side vector b;
Step2: Calculate the rank of A and the rank of the augmented matrix $[A|b]$;
Step3: Compare the ranks to determine the consistency of the system;
Step4: If the system is consistent and the ranks are equal to the number of variables, solve the system for a unique solution;
Step5: If the system is consistent but the ranks are less than the number of variables, find the general solution;

Step6: Output the solution(s) using the command `disp`.

Program

```
clc
clear

% Step 1: Define the coefficient matrix and right-hand side vector
A = [1, 2, 1; 1, 1, 1; 1, 1, 1];
b = [10; 10; 10];

% Step 2: Calculate the rank of the matrix A
rank_A = rank(A);

% Step 3: Calculate the rank of the augmented matrix [A|b]
rank_Ab = rank([A b]);

% Step 4: Number of variables (columns of A)
num_vars = size(A, 2);

% Step 5: Compare the ranks to determine the consistency of the system
if rank_A == rank_Ab
    if rank_A == num_vars
        disp('The system is consistent and has a unique solution.');
        % Solve the system
        x = inv(A) * b;
        disp('Solution:');
        disp(x);
    else
        % Step 6: Find the augmented matrix and its row reduced echelon form
        Ab = [A b];
        A_r = rref(Ab);
        % Logical index for rows where all entries are zero
        idx = all(A_r == 0, 2);
        % Remove rows with all zeros
        A_clean = A_r(~idx, 1:end-1);
        b = A_r(~idx, end);
        % Particular solution
        x_p = linsolve(A_clean, b);
        % Find the null space (general solution)
        N = null(A, 'r');
        x_general = x_p + N;
        disp('The system is consistent and has infinitely many solutions.');
        % Find the linear combination of the vectors in N
        c = sym('c', [size(N, 2) 1]);
        L = zeros(size(x_p));
        for i = 1:size(N, 2)
            L = L + c(i) * N(:, i);
        end
        % General solutions of the system
        general_solutions = x_p + L;
        disp('The general solutions of the system is:');
        disp(general_solutions);
    end
else
    disp('The system is inconsistent and has no solution.');
end
```

Output

```
The system is consistent and has infinitely many solutions.
The general solutions of the system is:
10 - c1
     0
    c1
```

Here, the $c1$ and $c2$ are arbitrary coefficients.

Conclusion

The program correctly determines the consistency of the system and provides the solution. If the system is consistent and the ranks are equal to the number of variables, it finds the unique solution. If the system is consistent but the ranks are less than the number of variables, it provides the general solution. If the system is inconsistent, it indicates that there is no solution.

Practical No. 25

Aim

To write a MATLAB program to solve a homogeneous system of linear equations using inbuilt commands.

Problem

Write a program to solve a homogeneous system of linear equations, $AX = 0$, given by

$$\begin{aligned} x + 2y + 3z &= 0, \\ 4x + 5y + 6z &= 0, \\ 7x + 8y + 9z &= 0. \end{aligned}$$

Theory

A homogeneous system of linear equations is of the form $AX = 0$, where A is a matrix of coefficients and X is the vector of unknowns. The solution to this system can be found by determining the null space of A.

Algorithm

Step1: Define the coefficient matrix A;
Step2: Calculate the rank of A using command `rank(A)`;
Step3: Find the number of variables by finding the number of columns in A;
Step4: Check whether system has trivial solution or infinite solutions;
Step5: Output the solution using the command `disp`.

Program

```
clc
clear
% Step 1: Example matrix with a unique solution
A = [1, 2, 3; 4, 5, 6; 7, 8, 9];

% Step 2: Calculate the rank of the matrix A
matrix_rank = rank(A);

% Step 3: Number of variables (columns of A)
num_vars = size(A, 2);

% Step 4:
if matrix_rank == num_vars
    disp('The system has a unique (trivial) solution');
else
     disp('The system has infinitely many solutions.');

    % Find the null space (general solution)
    N = null(A, ‘r’);

    % Step 6
    %%% to find the linear combination of the vectors in N
        c = sym(‘c’, [size(N, 2) 1]);
        L=zeros(size(1));
     for i=1:size(N, 2)
        L = L+c(i)*N(:, i);
     end
     % To find the general solutions of the system
        general_solutions=L;
        disp(‘The general solutions of the system is:’)
        disp(general_solutions)
end
```

Output

```
The system has infinitely many solutions.
The general solutions of the system is:
   c1
-2*c1
   c1
```

Conclusion

The program correctly solves the homogeneous system of linear equations using the built-in symbolic command `null` and outputs the solution vector in symbolic form.

Practical No. 26

Aim

To write a program to check if a matrix is symmetric, skew-symmetric, hermitian, or skew-hermitian.

Problem

Write a program to determine if a given square matrix is symmetric, skew-symmetric, hermitian, or skew-hermitian. The program should output the type of the matrix.

Theory

- A matrix A is symmetric if $A = A^T$.
- A matrix A is skew-symmetric if $A = -A^T$.
- A matrix A is Hermitian if $A = A^H$, where A^H is the conjugate transpose of A.
- A matrix A is skew-Hermitian if $A = -A^H$, where A^H is the conjugate transpose of A.

Algorithm

Step1: Define the matrix A;
Step2: Compute the transpose A^T and the conjugate of transpose as A^H;
Step3: Check the conditions for symmetry, skew-symmetry, Hermitian, and skew-Hermitian;
Step4: Output the result using the command `disp`.

Program

```
clc
clear
% Step 1: Define the matrix
A = [1, 2+1i, 3; 2-1i, 5, 6; 3, 6, 9];

% Step 2: Compute the transpose and conjugate transpose
AT = A.'; % command for transpose
AH = A';  % command for transpose and conjugate

% Step 3: Check the conditions
is_symmetric = isequal(A, AT);
is_skew_symmetric = isequal(A, -AT);
is_Hermitian = isequal(A, AH);
is_skew_Hermitian = isequal(A, -AH);

% Step 4: Output the result
if is_symmetric
    disp('The matrix is symmetric.');
elseif is_skew_symmetric
    disp('The matrix is skew-symmetric.');
elseif is_Hermitian
    disp('The matrix is Hermitian.');
elseif is_skew_Hermitian
```

```
    disp('The matrix is skew-Hermitian.');
else
    disp('The matrix does not fall into any of the specified categories.');
end
```

Output

```
The matrix is Hermitian.
```

Conclusion

The program correctly checks if the given matrix is symmetric, skew-symmetric, Hermitian, or skew-Hermitian, and outputs the correct results.

Practical No. 27

Aim

To write a program to find the characteristic polynomial of a given matrix using symbolic form.

Problem

Write a program to compute the characteristic polynomial of the matrix

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & -1 & 2 \\ 0 & 0 & 3 \end{pmatrix}.$$

Theory

The characteristic polynomial of a matrix A is a polynomial which is invariant under matrix similarity and has the eigenvalues of the matrix as roots. It is defined as

$$p(\lambda) = \det(A - \lambda I)$$

where, λ is a scalar and I is the identity matrix of the same dimension of A.

Algorithm

Step 1: Define the matrix A;
Step2: Create a symbolic variable x;
Step3: Find the characteristic polynomial coefficients using the command `poly`;
Step4: Create the characteristic polynomial as a symbolic expression using the command `poly2sym`;
Step5: Output the characteristic polynomial using the command `disp`.

Program

```
clc
clear
% Step 1: Define the matrix
A = [1, 2, 3; 0, -1, 2; 0, 0, 3];

% Step 2: Create a symbolic variable
syms x;

% Step 3: Find the characteristic polynomial coefficients
char_poly_coeffs = poly(A);

% Step 4: Create the characteristic polynomial as a symbolic expression
char_poly = poly2sym(char_poly_coeffs, x);

% Step 5: Output the characteristic polynomial
disp('Characteristic polynomial:');
disp(char_poly);
```

Output

```
Characteristic polynomial:
x^3 - 3*x^2 - x + 3
```

Conclusion

The program correctly computes the characteristic polynomial of the given matrix in symbolic form.

Practical No. 28

Aim

To write a program to find the roots of a given polynomial.

Problem

Write a program to compute the roots of the polynomial

$$p(x) = x^3 - 3x^2 - x + 3.$$

Theory

The roots of a polynomial $p(x)$ are the values of x for which $p(x) = 0$. These roots can be found using various numerical and analytical methods. In MATLAB, the roots of a polynomial can be computed using the `roots` function.

Algorithm

Step1: Define the coefficient vector of polynomial $p(x)$;
Step2: Find the roots using the command `roots`;
Step3: Output the roots using the command `disp`.

Program

```
clc
clear
% Step 1: Define the polynomial
p = [1, -3, -1, 3];

% Step 2: Find the roots of the polynomial
roots_p = roots(p);

% Step 3: Output the roots
disp('Roots of the polynomial:');
disp(roots_p);
```

Output

```
Roots of the polynomial:
    3.0000
```

```
   -1.0000
    1.0000
```

Conclusion

The program correctly computes the roots of the given polynomial.

Practical No. 29

Aim

To write a program to find a polynomial from its given roots.

Problem

Write a program to compute the polynomial whose roots are $1, 2, 3$.

Theory

A polynomial can be uniquely determined by its roots. The `poly` function in MATLAB generates the coefficients of a polynomial from its roots. This polynomial can then be expressed as a symbolic expression for better readability in MATLAB.

Algorithm

Step1: Define the roots of the polynomial;
Step2: Compute the polynomial from its roots using the command `poly`;
Step3: Output the polynomial coefficients using the command `disp`;
Step4: Create the polynomial as a symbolic expression for better readability using the command `poly2sym`;
Step5: Output the symbolic polynomial using the command `disp`.

Program

```
clc
clear
% Step 1: Define the roots of the polynomial
roots_p = [1 2 3];
```

```
% Step 2: Create the polynomial from its roots
p = poly(roots_p);

% Step 3: Display the polynomial coefficients
disp('Polynomial coefficients:');
disp(p);

% Step 4: Create the polynomial as a symbolic expression
syms x;
poly_expr = poly2sym(p, x);

% Step 5: Output the symbolic polynomial
disp('Polynomial:');
disp(poly_expr);
```

Output:

```
Polynomial coefficients:
     1    -6    11    -6

Polynomial:
x^3 - 6*x^2 + 11*x - 6
```

Conclusion:

The program successfully computes the polynomial from the given roots and displays it both as coefficients and as a symbolic expression by demonstrating the process of polynomial reconstruction from its roots.

Practical No. 30

Aim

To write a program to define and plot functions in MATLAB.

Problem

Write a program to define the function $f(x) = x^2 + \sin(x)$ and plot it over the interval $[-10, 10]$.

Theory

In MATLAB, functions can be defined using the anonymous functions, and the `fplot` function can be used to plot the function over a specified interval.

Algorithm

Step1: Define the function $f(x)$ using an anonymous function;
Step2: Plot the function using the `fplot` command;
Step3: Label the x-axis using the `xlabel` command;
Step4: Label the y-axis using the `ylabel` command;
Step5: Title the plot using the `title` command.

Program

```
clc
clear
% Step 1: Define the function
syms x

f =  x^2 + sin(x);

% Step 2: Plot the function
fplot(f, [-10, 10]);

% Step 3: Label the x-axis
xlabel('x');

% Step 4: Label the y-axis
ylabel('f(x)');

% Step 5: Title the plot
title('Plot of the function f(x) = x^2 + sin(x)');
```

Conclusion:

The program successfully defines and plots the function $f(x) = x^2 + \sin(x)$ over the interval $[-10, 10]$. This demonstrates the use of `fplot` function in MATLAB for visualizing mathematical functions.

Practical No. 31

Aim

To write a program to classify the type of discontinuity of a function at a given point.

Problem

Write a program to determine and classify the type of discontinuity of the function $f(x) = \frac{1}{x}$ at the point $x = 0$.

Theory

Discontinuities in functions can be classified into different types:

- **Removable Discontinuity:** If the left-hand limit and right-hand limit at a point are equal but not equal to the function value at that point.
- **Infinite Discontinuity:** If the function approaches infinity from one or both sides of the point.
- **Jump Discontinuity:** If the left-hand limit and right-hand limit at a point are not equal.

In MATLAB, the `limit` function can be used to compute the left-hand and right-hand limits.

Algorithm:

Step1: Define the function $f(x)$ using an anonymous function;
Step2: Specify the point of interest c;
Step3: Compute the left-hand limit at c using the command `limit`;
Step4: Compute the right-hand limit at c using the command `limit`;
Step5: Compute the function value at c;
Step6: Classify the type of discontinuity based on the computed limits and function value;
Step7: Output the result using the command `disp`.

Program

```
clc
clear
% Step 1: Define the function
f = @(x) 1 / x;
```

```
% Step 2: Specify the point of interest
c = 1;

% Step 3: Compute the left-hand limit
syms x;
limit_left = limit(f(x), x, c, ‘left’);

% Step 4: Compute the right-hand limit
limit_right = limit(f(x), x, c, ‘right’);

% Step 5: Compute the function value at c
f_c = f(c);

% Step 6: Classify the type of discontinuity
if limit_left == limit_right && limit_left == f_c
    disp([‘Function is continuous at x = ’, num2str(c)]);
elseif limit_left == limit_right
    disp([‘Function has a removable discontinuity at x = ’, num2str(c)]);
elseif isinf(limit_left) || isinf(limit_right)
    disp([‘Function has an infinite discontinuity at x = ’, num2str(c)]);
else
    disp([‘Function has a jump discontinuity at x = ’, num2str(c)]);
end
```

Output

```
Function has an infinite discontinuity at x = 0
```

Conclusion:

The program correctly classifies the type of discontinuity of the function $f(x) = \frac{1}{x}$ at the point $x = 0$ as an infinite discontinuity. This demonstrates the use of symbolic computation in MATLAB for analyzing and classifying discontinuities in functions.

Practical No. 32

Aim

To write a program to check the differentiability of a function at a given point.

Problem

Write a program to determine and check the differentiability of the function

$$f(x) = \frac{1}{x}$$

at the point $x = 1$.

Theory

A function $f(x)$ is differentiable at a point c if the left-hand derivative and right-hand derivative at c are equal. The derivative of $f(x)$ at c can be computed using symbolic differentiation in MATLAB.

Algorithm:

Step1: Define the function $f(x)$ using an anonymous function;
Step2: Specify the point of interest c;
Step3: Compute the left-hand derivative at c using the `diff` function;
Step4: Compute the right-hand derivative at c using the `diff` function;
Step5: Check if the left-hand and right-hand derivatives are equal;
Step6: Output the result using the command `disp`.

Program

```
clc
clear
% Step 1: Define the function
f = @(x) 1 / x;

% Step 2: Specify the point of interest
c = 1;

% Step 3: Compute the left-hand derivative
```

```
syms x;
left_derivative = limit((f(x) - f(c)) / (x - c), x, c, ‘left’);

% Step 4: Compute the right-hand derivative
right_derivative = limit((f(x) - f(c)) / (x - c), x, c, ‘right’);

% Step 5: Check if the left-hand and right-hand derivatives are equal
if left_derivative == right_derivative
    disp([‘Function is differentiable at x = ’, num2str(c)]);
    disp([‘The derivative at x = ’, num2str(c), ‘ is ’, char(left_derivative)]);
else
    disp([‘Function is not differentiable at x = ’, num2str(c)]);
end
```

Output

```
Function is differentiable at x = 1
The derivative at x = 1 is -1
```

Conclusion

The program correctly checks the differentiability of the function $f(x) = \frac{1}{x}$ at the point $x = 1$ and shows that the function is differentiable at that point.

Practical No. 33

Aim

To write a program to compute the symbolic derivative of a differentiable function.

Problem

Write a program to compute the single order and double order derivative of the function

$$f(x) = x^2 + \sin(x).$$

Theory

The derivative of a function $f(x)$ at a point $x = c$ is defined as the limit

$$f'(c) = \lim_{h \to 0} \frac{f(c+h) - f(c)}{h}$$

If this limit exists, the function is said to be differentiable at that point, and the value of the limit is the derivative. In MATLAB, the `diff` function can be used to compute the derivatives symbolically.

Algorithm

Step1: Define the function $f(x)$;
Step2: Compute the symbolic derivative of $f(x)$ using the command `diff(f,x)`;
Step3: Compute the symbolic double derivative of $f(x)$ using the command `diff(f,x,2)`;
Step4: Output the result using the command `disp`.

Program

```
clc
clear
% Step 1: Define the function
syms x;
f = x^2 + sin(x);

% Step 2: Compute the symbolic first order derivative
f_derivative = diff(f, x);

% Step 3: Compute the symbolic second order derivative
f_double_derivative = diff(f, x, 2);

% Step 4: Output the result
disp('The derivative of f(x):');
disp(f_derivative)
disp('f_double_derivative of f(x):');
disp(f_double_derivative)
```

Output

```
The derivative of f(x):
2*x + cos(x)
```

```
f_double_derivative of f(x):
2 - sin(x)
```

Conclusion

The program successfully computes the first and double order derivative of the function $f(x) = x^2 + \sin(x)$.

Practical No. 34

Aim

To write a program to find the n^{th} roots of a complex number using De Moivre's Theorem.

Problem

Write a program to compute the n^{th} root of the complex number

$$z = 8 + 6i$$

for $n = 3$.

Theory

De Moivre's theorem states that for a complex number $z = re^{i\theta}$, the n^{th} roots are given by

$$z^{1/n} = r^{1/n}\left(\cos\left(\frac{\theta + 2k\pi}{n}\right) + i\sin\left(\frac{\theta + 2k\pi}{n}\right)\right)$$

for $k = 0, 1, \ldots, n-1$. Where, r is the modulus of z and θ is the argument of z. In MATLAB, we can use the `abs` and `angle` functions to compute the modulus and argument of a complex number, respectively.

Algorithm

Step1: Define the complex number z;
Step2: Compute the modulus r of z using the command `abs`;
Step3: Compute the argument θ of z using the command `angle`;
Step4: Compute the n^{th} roots using De Moivre's Theorem;
Step5: Output the results using the command `disp`.

Program

```
clc
clear
% Step 1: Define the complex number
z = 8 + 6i;

% Step 2: Compute the modulus
r = abs(z);

% Step 3: Compute the argument
theta = angle(z);

% Step 4: Compute the nth roots using De Moivre's Theorem
n = 3;
roots = zeros(n, 1);
for k = 0:n-1
    roots(k+1) = r^(1/n) * (cos((theta + 2*k*pi)/n) +....
                  i * sin((theta + 2*k*pi)/n));
end

% Step 5: Output the results
disp('The 3rd roots of the complex number 8 + 6i are:');
disp(roots);
```

Output

```
The 3rd roots of the complex number 8 + 6i are:
   2.1051 + 0.4586i
  -1.4497 + 1.5937i
  -0.6554 - 2.0523i
```

Conclusion

The program successfully computes the 3rd roots of the complex number $8 + 6i$ using De Moivre's theorem. This demonstrates the use of complex number operations in MATLAB to find the roots of complex numbers.

Practical No. 35

Aim

To write a program to find the Taylor series expansion of a given function about a specified point.

Problem

Write a program to compute the Taylor series expansion of the function

$$f(x) = \sin(x)$$

about the point $x = 2$ up to the 6^{th} order terms.

Theory

The Taylor series expansion of a function $f(x)$ about a point $x = a$ is given by

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots$$

In MATLAB, the `taylor` function can be used to compute the Taylor series expansion of a function symbolically.

Algorithm

Step1: Define the function $f(x)$ symbolically;
Step2: Specify the point a at which the Taylor series is to be expanded;
Step3: Compute the Taylor series expansion using the command `taylor`;
Step4: Output the result using the command `disp`.

Program

```
clc
clear
% Ensure the Symbolic Math Toolbox is available
syms x;

% Step 1: Define the function
f = sin(x);

% Step 2: Specify the point of expansion
a = 2;
```

```
% Step 3: Compute the Taylor series expansion
taylor_series = taylor(f, x, ‘ExpansionPoint’, a, ‘Order’, 4);

% Step 4: Output the result
disp(‘The Taylor series expansion is:’);
disp(taylor_series);
```

Output

```
The Taylor series expansion is:
sin(2) - (sin(2)*(x - 2)^2)/2 + cos(2)*(x - 2) - (cos(2)*(x - 2)^3)/6
```

Conclusion

The program successfully computes the Taylor series expansion of the function $\sin(x)$ about the point $x = 2$ up to the 6^{th} order terms using the symbolic computation in MATLAB. This demonstrates the use of symbolic computation to find the Taylor series of functions about any specified point.

Practical No. 36

Aim

To write a program to verify the Rolle's theorem for a given function on a specified interval.

Problem

Write a program to check if the function

$$f(x) = x^3 - 3x + 2$$

satisfies the conditions of Rolle's theorem on the interval $[0, 2]$.

Theory

Rolle's theorem states that if a function f is continuous on the closed interval $[a, b]$, differentiable on the open interval (a, b), and $f(a) = f(b)$, then there exists at least one number c in (a, b) such that $f'(c) = 0$.

Algorithm

Step1: Define the function $f(x)$ symbolically;
Step2: Specify the interval $[a, b]$;
Step3: Check if $f(a) = f(b)$;
Step4: Compute the derivative of $f(x)$;
Step5: Find the roots of the derivative within the interval (a, b);
Step6: Output the result using the command `disp`.

Program

```
clc
clear
syms x;
% Step 1: Define the continuous and differentiable function
f = x^3 - 3*x + 2;

% Step 2: Specify the interval
a = 0;
b = sqrt(3);

% Step 3: Check if f(a) = f(b)
fa = subs(f, x, a);
fb = subs(f, x, b);
if fa == fb
    % Step 4: Compute the derivative
    f_prime = diff(f, x);
    % Step 5: Find the roots of the derivative within the interval
    critical_points = solve(f_prime == 0, x);
    critical_points = double(critical_points);
    c = critical_points(critical_points > a & critical_points < b);
    % Step 6: Output the result
    if ~isempty(c)
        disp('Rolle's Theorem is satisfied.');
        disp(['There exists at least one c in interval such that f'(c) = 0.']);
        disp(['The value of c is: ', num2str(c')]);
    end
else
    disp('Rolle''s Theorem is not applicable since f(a) does not equal f(b).');
end
```

Output

```
Rolle's Theorem is satisfied.
There exists at least one number c in (0, 1.7321) such that f'(c) = 0.
The value of c is: 1
```

Conclusion

The program successfully verifies Rolle's Theorem for the function x^3-3x+2 on the interval $[0,2]$which demonstrates that there exists at least one point c in $(0,2)$ where the derivative of the function is zero.

Practical No. 37

Aim

To write a program to verify Bolzano's Intermediate Value theorem for a given function.

Problem

Write a program to verify Bolzano's Intermediate Value theorem for the function $f(x) = x^3 - 6x^2 + 11x - 6$ on the interval $[1.5, 2.5]$. Also, check if there is a root in this interval.

Theory

Bolzano's Intermediate Value theorem states that if $f(x)$ is a continuous function on the interval $[a,b]$ and $f(a)$ and $f(b)$ have opposite signs, then there exists at least one $c \in (a,b)$ such that $f(c) = 0$. This theorem is used to prove the existence of roots of a continuous functions within a given interval.

Algorithm

Step1: Define the function $f(x)$;
Step2: Define the interval $[a,b]$;
Step3: Check the signs of $f(a)$ and $f(b)$;
Step4: If $f(a)$ and $f(b)$ have opposite signs, conclude that there exist atleast one root in $[a,b]$;
Step5: Use a root-finding inbuilt function (e.g., `fzero`) to find a root in the interval;
Step6: Output the result using the command `disp`.

Program

```
clc
clear
% Step 1: Define the function
f = @(x) x^3 - 6*x^2 + 11*x - 6;

% Step 2: Define the interval [a, b]
a = 1.5;
b = 2.5;

% Step 3: Check the signs of f(a) and f(b)
fa = f(a);
fb = f(b);

% Step 4: Check if f(a) and f(b) have opposite signs
if fa * fb < 0
  disp('There exists at least one root in the given interval by Bolzano IVT.');
  % Step 5: Use fzero function to find a root in the interval
    root = fzero(f, [a, b]);
    root=round(root, 6);
    disp('Root found(approximated):');
    disp(root);
else
    disp('BolzanoIVT does not guarantee a root in the given interval.');
end
```

Output

```
There is at least one root in the given interval by Bolzano's IVT.
Root found:
2
```

Conclusion

The program correctly verifies Bolzano's Intermediate Value theorem and finds a root within the given interval. The theorem confirms the existence of at least one root in the interval $[1.5, 2.5]$ for the given function.

Practical No. 38

Aim

To write a program to verify Lagrange's Mean Value theorem for a given function.

Problem

Write a program to verify Lagrange's Mean Value theorem for the function $f(x) = x^3 - 3x + 1$ on the interval $[0, 2]$. Also, check if there exists a point $c \in (0, 2)$ such that $f'(c) = \frac{f(b)-f(a)}{b-a}$.

Theory

Lagrange's Mean Value theorem states that if a function f is continuous on the closed interval $[a, b]$ and differentiable on the open interval (a, b), then there exists at least one $c \in (a, b)$ such that

$$f'(c) = \frac{f(b) - f(a)}{b - a}.$$

This theorem provides a formalized version of the intuitive idea that a continuous and differentiable curve will have at least one tangent that is parallel to the secant line connecting the endpoints.

Algorithm

Step1: Define the function $f(x)$ and calculate the derivative $f'(x)$;
Step2: Define the interval $[a, b]$;
Step3: Calculate the slope of the secant line $\frac{f(b)-f(a)}{b-a}$;
Step4: Define the target function $g(c) = f'(c) - \frac{f(b)-f(a)}{b-a}$;
Step5: Use a root-finding inbuilt function (e.g., `fzero`) to find a root of the $g(c)$ in the interval (a, b);
Step6: Output the result using the command `disp`.

Program

```
clc
clear
syms x
% Step 1: Define a continuous function on [a,b] and differentiable function on (a,b)
f = @(x) x^3 - 3*x + 1;
df = diff(sym(f),x);    % we used sym to change f in symbolic
df= matlabFunction(df); % We used matlabFunction to change in matlab handles

% Step 2: Define the interval [a, b]
```

```
a = 0;
b = 2;

% Step 3: Calculate the slope of the secant line
slope_secant = (f(b) - f(a)) / (b - a);

% Step 4: Define the target function g(c)
g = @(c) df(c) - slope_secant;
% Step 5: Use fzero to find a root of g(c) in the interval (a, b)
c = fzero(g, [a, b]);
% Step 6: Output the result
if ~isempty(c)
disp('There exists a point c in the given interval such that f'(c) satisfies
LMVT.');
disp('Point c found:');
disp(c);
end
```

Output

```
There exists a point c in the given interval such that f'(c) satisfies LMVT.
Point c found:
    1.1547
```

Conclusion

The program correctly verifies Lagrange's Mean Value theorem and finds a point within the given interval that satisfies the theorem. The theorem confirms that there exists at least one point $c \in (0, 2)$ for the given function.

Practical No. 39

Aim

To write a program to determine the maximum number of positive and negative roots of a polynomial using Descartes' rule of sign.

Problem

Write a code to find the maximum number of positive and negative roots of polynomial

$$P(x) = x^3 - 6x^2 + 11x - 6.$$

Theory

Descartes' rule of signs states that:

1. The number of positive real roots of a polynomial is either equal to the number of sign changes between consecutive nonzero coefficients or less than it by an even number.

2. The number of negative real roots of a polynomial is either equal to the number of sign changes between consecutive nonzero coefficients after substituting x with $-x$ or less than it by an even number.

Algorithm

Step1: Define the coefficient list of polynomial $P(x)$ and $P(-x)$;
Step2: Remove zero coefficients from the polynomial;
Step3: Compute the maximum number of positive roots;
Step4: Compute the maximum number of negative roots;
Step5: Display the results.

Program

```
clc
clear

% Step 1: Define the polynomial coefficients for P(x) and P(-x)
% Example polynomial: P(x) = x^3 - 6x^2 + 11x - 6 (roots are 1, 2, 3)
coeffs = [1, -6, 11, -6];
neg_coeffs = coeffs .* (-1).^(length(coeffs)-1:-1:0);

% Step 2: Remove zero coefficients
coeffs = coeffs(coeffs ~= 0);
neg_coeffs = neg_coeffs(neg_coeffs ~= 0);

% Step 3: Compute the number of sign changes for positive roots
number = 0;
for i = 1:length(coeffs)-1
    if sign(coeffs(i)) ~= sign(coeffs(i+1))
        number = number + 1;
    end
end
max_positive_roots = number;

% Step 4: Compute the number of sign changes for negative roots
number1 = 0;
for i = 1:length(neg_coeffs)-1
```

```
    if sign(neg_coeffs(i)) ~= sign(neg_coeffs(i+1))
        number1 = number1 + 1;
    end
end
max_negative_roots = number1;

% Step 5: Display the results
disp(['Maximum number of positive roots: ', num2str(max_positive_roots)]);
disp(['Maximum number of negative roots: ', num2str(max_negative_roots)]);
```

Output

```
Maximum number of positive roots: 3
Maximum number of negative roots: 0
```

Conclusion

The program successfully determines the maximum number of positive and negative roots of a polynomial using Descartes' rule of signs.

Practical No. 40

Aim

To write a program to find the radius of convergence of a given power series using the ratio test.

Problem

Write a code to find the radius of convergence of the power series $\sum_{n=0}^{\infty} \frac{x^n}{n!}$.

Theory

The radius of convergence of a power series $\sum a_n x^n$ is found using the ratio test, which is defined as: if

$$\lim_{n\to\infty} \left| \frac{a_{n+1}}{a_n} \right| = L, \; then,$$

the radius of convergence R is given by:

$$R = \frac{1}{L}$$

Algorithm

Step1: Define the coefficients of the power series;
Step2: Compute the limit of the ratio of successive coefficients;
Step3: Calculate the radius of convergence;
Step4: Display the result.

Program

```
clc
clear

% Step 1: Define the coefficients of the power series
% Example: Coefficients for the series sum(a_n * x^n) where a_n = 1/n!
coeffs = @(n) 1 ./ factorial(n);

% Step 2: Compute the limit of the ratio of successive coefficients
syms n;
ratio = abs(coeffs(n+1) / coeffs(n));
L = limit(ratio, n, Inf);

% Step 3: Calculate the radius of convergence
R = 1 / L;

% Step 4: Display the result
disp('Radius of convergence:');
disp(R);
```

Output

```
Radius of convergence:
Inf
```

Conclusion

The program correctly computes the radius of convergence of the given power series using the ratio test.

Practical No. 41

Aim

To write a program to determine the convergence of a given sequence using the limit.

Problem

Write a code to check whether the sequence

$$a_n = 1/n$$

is convergent or not.

Theory

A sequence $\{a_n\}$ converges to a limit L if for every $\epsilon > 0$, there exists a positive integer N such that for all $n > N$, $|a_n - L| < \epsilon$. In MATLAB, this can be checked by computing the limit of the sequence as n approaches to infinity.

Algorithm

Step1: Define the sequence;
Step2: Compute the limit of the sequence as n approaches to infinity;
Step3: Check if the limit exists;
Ste 4: Display the result.
Program:

```
clc
clear

% Step 1: Define the sequence
% Example: Sequence a_n = 1/n
syms n;
sequence = 1 / n;

% Step 2: Compute the limit of the sequence as n approaches infinity
L = limit(sequence, n, Inf);

% Step 3: Check if the limit exists
if isfinite(L)
    result = ‘The sequence is convergent.’;
else
    result = ‘The sequence is divergent.’;
end
```

```
% Step 4: Display the result
disp('Limit of the sequence:');
disp(L);
disp(result);
```

Output

```
Limit of the sequence:
0
The sequence is convergent.
```

Conclusion

The program correctly determines the limit of the given sequence and checks whether the sequence is convergent or divergent. In this example, the sequence $\{1/n\}$ converges to 0 as n approaches to infinity.

Practical No. 42

Aim

To write a program to determine the convergence or divergence of a given series.

Problem

Write a program to check whether the given series is convergent or divergent. Also, find the sum of the series if it is convergent.

Theory

A series $\sum_{n=1}^{\infty} a_n$ converges if the sequence of partial sums $S_N = \sum_{n=1}^{N} a_n$ converges as N approaches infinity. In MATLAB, this can be checked by computing the limit of the partial sums.

Algorithm

Step1: Define the series;
Step2: Compute the sum of the series;

Step3: Check if the sum exists;
Step4: Display the result.

Program

```
clc
clear

% Step 1: Define the series
% Example: Series a_n = 1/n^2
syms n;
series = 1 / n^2;

% Step 2: Compute the partial sum of the series
sum = symsum(series, n, 1, Inf);

% Step 3 and 4: Check if the sum exists
if isfinite(sum)
    disp('Sum of the series is:');
    disp(sum);
else
    disp('The series is divergent.');
end
```

Output

```
Sum of the series is:
pi^2/6
```

Conclusion

The program correctly determines the sum of the series and checks whether the series is convergent or divergent. In this example, the series $\sum_{n=1}^{\infty} \frac{1}{n^2}$ converges to $\frac{\pi^2}{6}$, indicating that the series is convergent.

Practical No. 43

Aim

To calculate the definite integral of a given function over a specified interval.

Problem

Calculate the integral of the function $f(x) = x^2$ over the interval $[0, 1]$.

Theory

The definite integral of a function $f(x)$ over an interval $[a, b]$ is given by

$$\int_a^b f(x)\,dx$$

Algorithm

Step1: Define the function $f(x)$;
Step2: Specify the limits of integration $[a, b]$;
Step3: Calculate the integral using the appropriate numerical method or symbolic integration.

Program

```
clc
clear

% Step 1: Define the function
f = @(x) x.^2;

% Step 2: Specify the limits of integration
a = 0;
b = 1;

% Step 3: Calculate the integral
integral_value = integral(f, a, b);

% Step 4: Display the result
disp([‘Value of the integral of f(x) = x^2 from ’, num2str(a), ‘to’,
num2str(b),‘:’]);
disp(integral_value);
```

Output

```
Value of the integral of f(x) = x^2 from 0 to 1:
0.3333
```

Conclusion

The program correctly calculates the definite integral of the function $f(x) = x^2$ over the interval $[0, 1]$, providing the numerical value of approximately 0.3333.

Chapter 2

Single and Multi Variate Calculus

Practical No. 1

Aim

To write a program to evaluate the limit of multivariable function using MATLAB.

Problem

Determine whether the following limit exist? :

$$\lim_{(x,y)\to(0,0)} \frac{x^2 - y^2}{x^2 + y^2}.$$

Theory

The limit of a multivariable function as $(x, y) \to (0, 0)$ exists if the function approaches the same value along any path leading to the point $(0, 0)$. In this case, we will evaluate the limit of the function $\frac{x^2-y^2}{x^2+y^2}$ along several paths and see if the limit is the same along each path. To check the limit, we will consider:
1. The path along the x-axis ($y = 0$).
2. The path along the y-axis ($x = 0$).
3. The path along $y = x$ (the line $y = x$).

Algorithm

Step 1: Define the function $f(x, y) = \frac{x^2-y^2}{x^2+y^2}$;
Step 2: Evaluate the limit along the x-axis ($y = 0$);
Step 3: Evaluate the limit along the y-axis ($x = 0$);
Step 4: Evaluate the limit along the path $y = x$;
Step 5: Compare the results of the different paths to determine if the limit exists.

MATLAB Program

```
% Define symbolic variables
syms x y;

% Define the function f(x, y)
f = (x^2 - y^2) / (x^2 + y^2);

% Evaluate the limit along the x-axis (y = 0)
limit_x_axis = limit(f, y, 0);

% Evaluate the limit along the y-axis (x = 0)
limit_y_axis = limit(f, x, 0);

% Evaluate the limit along the path y = x
limit_line_y_equals_x = limit(f, y, x);

% Display the results
disp([‘Limit along the x-axis: ’, num2str(limit_x_axis)]);
disp([‘Limit along the y-axis: ’, num2str(limit_y_axis)]);
disp([‘Limit along the path y = x: ’, num2str(limit_line_y_equals_x)]);
```

Output

```
Limit along the x-axis: 1
Limit along the y-axis: -1
Limit along the path y = x: 0
```

General Matlab code for finding limit of any function

```
% Step 1: Define variables x and y
syms x y;

% Step 2: Request the user to input the function f(x, y)
f_input = input(‘Enter the function f(x, y): ’, ’s’);
f = str2func([‘@(x, y) ’ f_input]); % Convert to an anonymous function

% Step 3: Request the user to input the limit point (a, b)
a = input(‘Enter the limit point for x (a): ’);
b = input(‘Enter the limit point for y (b): ’);

% Step 4: limit calculations along different paths
try
    % Define the function f(x, y)
    f_sym = str2sym(f_input);

    % Substitute different paths into the function
    f_path1 = subs(f_sym, y, x);     % Path 1: y = x
    f_path2 = subs(f_sym, y, -x);    % Path 2: y = -x
    f_path3 = subs(f_sym, y, b);     % Path 3: y = b (constant y)

    % Calculate limits along these paths
    limit_path1 = limit(f_path1, x, a);  % Limit as x -> a along y = x
    limit_path2 = limit(f_path2, x, a);  % Limit as x -> a along y = -x
    limit_path3 = limit(f_path3, x, a);  % Limit as x -> a along y = b

    % Display the results of limits
    fprintf('Limit along y = x: %s\n', char(limit_path1));
    fprintf('Limit along y = -x: %s\n', char(limit_path2));
    fprintf('Limit along y = %f: %s\n', b, char(limit_path3));

catch
    % Handle any errors in limit calculation
    warning('Error with symbolic limit calculation. Proceeding with numerical
    evaluation.');
end

% Step 5: Numerical evaluation and handling of undefined values
% Define a grid around the limit point for numerical evaluation
grid_range = linspace(a - 1, a + 1, 100);  % Define a small range around the poin
(a, b)
[x_grid, y_grid] = meshgrid(grid_range, grid_range);

% Safe numerical evaluation of the function, with small epsilon to avoid division
zero epsilon = 1e-6;
try
    f_values = f(x_grid, y_grid);  % Evaluate the function on the grid
catch
    warning('Numerical evaluation encountered issues, attempting to
    handle undefined values.');
    f_values = NaN(size(x_grid));  % Set as NaN if there is an evaluation error
end

% Avoid division by zero or undefined values (replace Inf and NaN)
f_values(isinf(f_values) | isnan(f_values)) = NaN;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Step 6: Plotting the surface plot
```

```
figure;
surf(x_grid, y_grid, f_values, 'EdgeColor', 'none'); % 3D surface plot
title([‘Surface Plot of f(x, y) near (' num2str(a) ', ' num2str(b) ')']);
xlabel(‘x');
ylabel(‘y');
zlabel(‘f(x, y)');
colorbar;
shading interp;

% Step 7: Plotting the contour plot
figure;
contourf(x_grid, y_grid, f_values, 20); % 20 contour levels for detail
title([‘Contour Plot of f(x, y) near (' num2str(a) ', ' num2str(b) ')']);
xlabel(‘x');
ylabel(‘y');
colorbar;
```

Conclusion

The limit of the function $\frac{x^2-y^2}{x^2+y^2}$ as $(x,y) \to (0,0)$ does not exist because the limit differs along different paths. Specifically, the limit is 1 along the x-axis, -1 along the y-axis, and 0 along the path $y = x$.

Exercise Problem

Determine whether the following limit exist?

$$\lim_{(x,y)\to(0,0)} \frac{x^3 - y^3}{x^3 + y^3}.$$

Practical No. 2

Aim

To write a program to evaluate the continuity of multivariable function using MATLAB.

Problem

Check the continuity of the function

$$f(x,y) = \sin(x) + \cos(y).$$

Theory

A function $f(x,y)$ is said to be continuous at a point (x_0, y_0) if the following conditions are satisfied:
1. The function $f(x,y)$ is defined at (x_0, y_0).
2. The limit of the function exists at (x_0, y_0).
3. The value of the function at (x_0, y_0) is equal to the limit of the function as (x,y) approaches (x_0, y_0).

Algorithm

Step 1: Define the function $f(x,y) = \sin(x) + \cos(y)$;
Step 2: Evaluate the function at an arbitrary point (x_0, y_0);
Step 3: Evaluate the limit of $f(x,y)$ as (x,y) approaches (x_0, y_0);
Step 4: Check if the function value at (x_0, y_0) matches the limit, confirming continuity.

MATLAB Program

```
% Define symbolic variables
syms x y;

% Define the function f(x, y)
f = sin(x) + cos(y);

% Check the continuity by evaluating the function at an arbitrary point (e.g.,(0,
f_val = subs(f, {x, y}, {0, 0});

% Display the function value at the point (0, 0)
disp([‘f(0, 0) = ’, num2str(f_val)]);
```

Output

```
f(0, 0) = 1
```

General Matlab code for finding continuity of any function

```
% Step 1: Define the function
user_func = input(‘Enter the function f(x, y) in terms of x and y as a string: ’,
’s’);
f = str2func([‘@(x, y) ’, user_func]);  % Convert string to function handle
```

```
% Step 2: Input the point of interest (a, b)
a = input('Enter the value of x (a): ');
b = input('Enter the value of y (b): ');

% Step 3: Evaluate function value at (a, b), checking for division by zero
try
    f_value = f(a, b);
    fprintf('f(%f, %f) = %f\n', a, b, f_value);
catch
    fprintf('The function f(x, y) is undefined at (%f, %f) due to division by zero.
    \n', a, b);
    f_value = NaN;  % Mark as undefined
end

% Step 4: Evaluate the limit along the x-axis (y = b), checking for division by zero
try
    x_axis_limit = f(a, b);  % Limit when y = b (same as evaluating f(a, b))
    fprintf('Limit along the x-axis (f(%f, %f)) = %f\n', a, b, x_axis_limit);
catch
    fprintf('The limit along the x-axis is undefined at (%f, %f) due to division by
    zero.\n', a, b);
    x_axis_limit = NaN;  % Mark as undefined
end

% Step 5: Evaluate the limit along the y-axis (x = a), checking for division by zero
try
    y_axis_limit = f(a, b);  % Limit when x = a (same as evaluating f(a, b))
    fprintf('Limit along the y-axis (f(%f, %f)) = %f\n', a, b, y_axis_limit);
catch
    fprintf('The limit along the y-axis is undefined at (%f, %f) due to division by
    zero.\n', a, b);
    y_axis_limit = NaN;  % Mark as undefined
end

% Step 6: Check continuity only if values are not NaN
if ~isnan(f_value) && f_value == x_axis_limit && f_value == y_axis_limit
    fprintf('The function f(x,y) is continuous at (%f, %f).\n', a, b);
else
    fprintf('The function f(x,y) is not continuous at (%f, %f).\n', a, b);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Step 7: Graphical Representation
% Create a meshgrid for x and y values
```

```
x_vals = linspace(-pi, pi, 100); % x range, can be adjusted
y_vals = linspace(-pi, pi, 100); % y range, can be adjusted
[X, Y] = meshgrid(x_vals, y_vals); % Create grid

% Evaluate the function on the grid with error handling
Z = NaN(size(X));  % Initialize Z with NaN values for undefined points

for i = 1:numel(X)
    try
        Z(i) = f(X(i), Y(i));
    catch
        Z(i) = NaN;  % Keep NaN where the function is undefined
    end
end

% Create the surface plot, using NaN handling to skip undefined points
figure;
surf(X, Y, Z, 'EdgeColor', 'none'); % 3D surface plot
xlabel('x'); % Label x-axis
ylabel('y'); % Label y-axis
zlabel('f(x, y)'); % Label z-axis
title('Surface plot of user-defined f(x, y) with handling for undefined points');
% Title
colorbar; % Show color scale
grid on; % Turn on grid
```

Conclusion

The function $f(x, y) = \sin(x) + \cos(y)$ is continuous at every point in its domain, as both $\sin(x)$ and $\cos(y)$ are continuous functions. Therefore, $f(x, y)$ is continuous everywhere.

Exercise Problem

Check the continuity of the function $f(x, y) = e^x \cos(y)$.

Practical No. 3

Aim

Write a program to compute the first partial derivatives of the function using MATLAB.

Problem

Given the function

$$f(x,y) = x^2y + 3xy^2,$$

find the following partial derivatives:

$$f_x(x,y) \quad \text{and} \quad f_y(x,y).$$

Theory

To compute the partial derivatives of the function $f(x,y) = x^2y + 3xy^2$, we need to apply the following steps:
1. The first partial derivative with respect to x involves differentiating $f(x,y)$ with respect to x, treating y as a constant.
2. The first partial derivative with respect to y involves differentiating $f(x,y)$ with respect to y, treating x as a constant.

Algorithm

Step 1: Define the function $f(x,y) = x^2y + 3xy^2$;
Step 2: Compute the partial derivative $f_x(x,y)$ with respect to x;
Step 3: Compute the partial derivative $f_y(x,y)$ with respect to y;
Step 4: Display the results.

MATLAB Program

```
% Define the symbolic variables
syms x y;

% Define the function f(x, y)
f = x^2 * y + 3 * x * y^2;

% Compute the partial derivatives with respect to x and y
f_x = diff(f, x);
f_y = diff(f, y);

% Display the results
disp([‘f_x(x, y) = ’, char(f_x)]);
disp([‘f_y(x, y) = ’, char(f_y)]);
```

Output

```
f_x(x, y) = 2*x*y + 3*y^2
```

```
f_y(x, y) = x^2 + 6*x*y
```

Conclusion

The first partial derivatives of the function $f(x,y) = x^2y + 3xy^2$ are:

$$f_x(x,y) = 2xy + 3y^2$$

and

$$f_y(x,y) = x^2 + 6xy.$$

Exercise Problem

Given the function $f(x,y) = x^2+4xy+2y^2$, find the partial derivatives $f_x(x,y)$ and $f_y(x,y)$.

Practical No. 4

Aim

Write a program to compute the first partial derivatives using MATLAB.

Problem

Given the function:

$$f(x,y) = \ln(x^2 + y^2),$$

find the following partial derivatives at the point $(x,y) = (3,4)$:

$$f_x(x,y) \quad \text{and} \quad f_y(x,y).$$

Theory

To compute the partial derivatives of the function $f(x,y) = \ln(x^2 + y^2)$, we need to apply the chain rule:
1. The first partial derivative with respect to x involves differentiating $\ln(x^2 + y^2)$ with respect to x, treating y as constant.
2. The first partial derivative with respect to y involves differentiating $\ln(x^2 + y^2)$ with respect to y, treating x as constant.
3. After computing the partial derivatives, evaluate them at the point $(3,4)$.

Algorithm

Step 1: Define the function $f(x, y) = \ln(x^2 + y^2)$;
Step 2: Compute the partial derivative $f_x(x, y)$ with respect to x;
Step 3: Compute the partial derivative $f_y(x, y)$ with respect to y;
Step 4: Evaluate both partial derivatives at the point $(x, y) = (3, 4)$;
Step 5: Display the results.

MATLAB Program

```
% Define the symbolic variables
syms x y;

% Define the function f(x, y)
f = log(x^2 + y^2);

% Compute the partial derivatives with respect to x and y
f_x = diff(f, x);
f_y = diff(f, y);

% Evaluate the partial derivatives at the point (x, y) = (3, 4)
f_x_val = subs(f_x, {x, y}, {3, 4});
f_y_val = subs(f_y, {x, y}, {3, 4});

% Display the results
disp([‘f_x(3, 4) = ’, num2str(f_x_val)]);
disp([‘f_y(3, 4) = ’, num2str(f_y_val)]);
```

Output

```
f_x(3, 4) = 0.24
f_y(3, 4) = 0.32
```

Conclusion

The first partial derivatives of the function $f(x, y) = \ln(x^2 + y^2)$ at the point $(x, y) = (3, 4)$ are computed as follows:

$$f_x(3, 4) = 0.24 \quad \text{and} \quad f_y(3, 4) = 0.32.$$

Exercise Problem

Given the function $f(x, y) = \ln(x^2 + y^2)$, find the partial derivatives f_x and f_y at the point $(x, y) = (2, 3)$.

Practical No. 5

Aim

Write a program to compute the higher-order partial derivatives using MATLAB.

Problem

Given the function

$$f(x, y) = \sin(xy) + x^2y^3,$$

find the following higher-order partial derivatives:

$$f_{xx}, \quad f_{yy}, \quad f_{xy}, \quad f_{xxy}.$$

Theory

The higher-order partial derivatives involve taking successive derivatives of the function with respect to the specified variables. To find the higher-order partial derivatives, we apply the following steps:
1. f_x is the first partial derivative of f with respect to x.
2. f_{xx} is the second partial derivative of f with respect to x.
3. f_y is the first partial derivative of f with respect to y.
4. f_{yy} is the second partial derivative of f with respect to y.
5. Mixed derivatives such as f_{xy} and f_{xxy} are computed by taking partial derivatives with respect to both variables in the specified order.

Algorithm

Step 1: Define the function $f(x, y) = \sin(xy) + x^2y^3$;
Step 2: Compute the first and second partial derivatives f_x, f_y, f_{xx}, f_{yy}, f_{xy}, and f_{xxy};
Step 3: Display the results.

MATLAB Program

```
% Define the symbolic variables
syms x y;

% Define the function f(x, y)
f = sin(x*y) + x^2 * y^3;

% Compute the higher-order partial derivatives
f_x = diff(f, x);
```

```
f_xx = diff(f_x, x);
f_y = diff(f, y);
f_yy = diff(f_y, y);
f_xy = diff(f_x, y);
f_xxy = diff(f_xx, y);

% Display the results
disp([‘f_x = ’, char(f_x)]);
disp([‘f_xx = ’, char(f_xx)]);
disp([‘f_y = ’, char(f_y)]);
disp([‘f_yy = ’, char(f_yy)]);
disp([‘f_xy = ’, char(f_xy)]);
disp([‘f_xxy = ’, char(f_xxy)]);
```

Output

```
f_x = y*cos(x*y) + 2*x*y^3
f_xx = -y^2*sin(x*y) + 2*y^3
f_y = x*cos(x*y) + 3*x^2*y^2
f_yy = -x^2*sin(x*y) + 6*x^2*y
f_xy = cos(x*y) - x*y*sin(x*y) + 6*x*y^2
f_xxy = -x*sin(x*y) - x^2*y*cos(x*y) + 12*x*y
```

Conclusion

The higher-order partial derivatives of the function $f(x, y) = \sin(xy) + x^2y^3$ are computed as follows:

$$f_x = y\cos(xy) + 2xy^3, \quad f_{xx} = -y^2\sin(xy) + 2y^3, \quad f_y = x\cos(xy) + 3x^2y^2,$$

$$f_{yy} = -x^2\sin(xy)+6x^2y, \quad f_{xy} = \cos(xy)-xy\sin(xy)+6xy^2, \quad f_{xxy} = -x\sin(xy)-x^2y\cos(xy)+12xy.$$

Exercise Problem

Find the higher-order partial derivatives $f_{xx}, f_{yy}, f_{xy}, f_{xxy}$ for the function $f(x, y) = e^{xy} + x^3y^2$.

Practical No. 6

Aim

Write a program to find the equation of the tangent plane to the surface using MATLAB.

Problem

Given the surface

$$z = x^2 + y^2,$$

find the equation of the tangent plane at the point $(1, 1, 2)$.

Theory

The equation of the tangent plane to a surface $z = f(x, y)$ at the point (x_0, y_0, z_0) is given by:

$$z - z_0 = \frac{\partial f}{\partial x}(x_0, y_0)(x - x_0) + \frac{\partial f}{\partial y}(x_0, y_0)(y - y_0).$$

Here, $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ are the partial derivatives of $f(x, y)$ with respect to x and y, evaluated at the point (x_0, y_0).

Algorithm

Step 1: Define the surface function $f(x, y) = x^2 + y^2$;
Step 2: Compute the partial derivatives $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$;
Step 3: Evaluate the partial derivatives at the point $(1, 1)$;
Step 4: Use the formula for the tangent plane to write the equation of the tangent plane at $(1, 1, 2)$.

MATLAB Program

```
% Define the symbolic variables
syms x y;

% Define the surface function z = x^2 + y^2
z = x^2 + y^2;

% Compute the partial derivatives
dzdx = diff(z, x);
dzdy = diff(z, y);
```

```
% Evaluate the partial derivatives at the point (1, 1)
dzdx_at_point = double(subs(dzdx, {x, y}, {1, 1}));
dzdy_at_point = double(subs(dzdy, {x, y}, {1, 1}));

% Equation of the tangent plane
z_tangent = z - 2; % point (1, 1, 2)
tangent_plane_eq = dzdx_at_point * (x - 1) + dzdy_at_point * (y - 1) + 2;

% Display the result
disp([‘The equation of the tangent plane is: z = ’, char(tangent_plane_eq)]);
```

Output

```
The equation of the tangent plane is: z = 2*x + 2*y - 2
```

Conclusion

The equation of the tangent plane to the surface $z = x^2 + y^2$ at the point $(1, 1, 2)$ is $z = 2x + 2y - 2$.

Exercise Problem

Find the equation of the tangent plane to the surface $z = x^2 + 3y^2$ at the point $(2, 1, 7)$.

Practical No. 7

Aim

Write a Program to compute total differentiation using MATLAB.

Problem

Given the function:

$$f(x, y) = e^{xy},$$

where $x = \sin(t)$ and $y = \cos(t)$, find $\frac{df}{dt}$ at $t = \frac{\pi}{4}$.

Theory

The chain rule is used to differentiate functions of several variables. For a function $f(x, y)$ where $x = x(t)$ and $y = y(t)$, the total derivative with respect to t is given by:

$$\frac{df}{dt} = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}.$$

Here, we will calculate the partial derivatives of $f(x, y) = e^{xy}$ with respect to x and y, then apply the chain rule.

Algorithm

Step 1: Define the function $f(x, y) = e^{xy}$ and the relationships $x = \sin(t)$ and $y = \cos(t)$;
Step 2: Compute the partial derivatives $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$;
Step 3: Find $\frac{dx}{dt}$ and $\frac{dy}{dt}$;
Step 4: Apply the chain rule to compute $\frac{df}{dt}$;
Step 5: Evaluate $\frac{df}{dt}$ at $t = \frac{\pi}{4}$.

MATLAB Program

```
% Define the time variable t
syms t;

% Define x(t) and y(t)
x = sin(t);
y = cos(t);

% Define the function f(x, y) = exp(x * y)
f = exp(x * y);

% Compute the derivative of f with respect to t
dfdt = diff(f, t);

% Evaluate df/dt at t = pi/4
result = double(subs(dfdt, t, pi/4));

% Display the result
disp(['The value of df/dt at t = pi/4 is: ', num2str(result)]);
```

Output

```
The value of df/dt at t = pi/4 is: 0
```

Conclusion

The value of $\frac{df}{dt}$ at $t = \frac{\pi}{4}$ is 0.

Exercise Problem

Find $\frac{df}{dt}$ for the given function $f(x, y) = e^{x+y}$, where $x = \sin(t)$ and $y = \cos(t)$, at $t = \frac{\pi}{2}$.

Practical No. 8

Aim

Write a program to find the gradient of the function using MATLAB.

Problem

Given the function

$$f(x, y, z) = x^2 + y^2,$$

find the gradient of f at the point $(3, 4)$.

Theory

The gradient of a scalar function $f(x, y, z)$ is a vector that points in the direction of the greatest rate of increase of the function. It is defined as the vector of partial derivatives of f with respect to each of its variables as follows

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right).$$

For the given function $f(x, y, z) = x^2 + y^2$, the gradient will be a vector with components $\frac{\partial f}{\partial x}$, $\frac{\partial f}{\partial y}$ and $\frac{\partial f}{\partial z}$ will be zero, as the function does not depend on z.

Algorithm

Step 1: Define the function $f(x, y, z) = x^2 + y^2$;
Step 2: Compute the partial derivatives $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$;
Step 3: Evaluate the gradient at the point $(3, 4)$;
Step 4: Display the gradient vector.

MATLAB Program

```
% Define the point (3, 4) and function f(x, y, z) = x^2 + y^2
x0 = 3; y0 = 4; z0 = 0;

% Define the partial derivatives of f(x, y, z) = x^2 + y^2
dfdx = 2 * x0;
dfdy = 2 * y0;
dfdz = 0; % Since f does not depend on z

% Display the gradient
```

```
gradient_f = [dfdx, dfdy, dfdz];
disp(['The gradient of f at the point (3, 4) is: ', mat2str(gradient_f)]);
```

Output

```
The gradient of f at the point (3, 4) is: [6, 8, 0]
```

Conclusion

The gradient of the function $f(x, y, z) = x^2 + y^2$ at the point $(3, 4)$ is the vector $(6, 8, 0)$.

Exercise Problem

Find the gradient of the function $f(x, y) = x^2y + y^3$ at the point $(2, 1)$.

Practical No. 9

Aim

Write a program to calculate the directional derivative using MATLAB.

Problem

Given the function

$$f(x, y, z) = xy + z,$$

calculate the directional derivative of f at the point $(2, -1, 3)$ in the direction of $\vec{v} = (1, 2, -1)$.

Theory

The directional derivative of a function $f(x, y, z)$ at a point $P(x_0, y_0, z_0)$ in the direction of a vector $\vec{v} = (a, b, c)$ is given by:

$$D_{\vec{u}}f = \nabla f \cdot \vec{u},$$

where $\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}\right)$ is the gradient of f and $\vec{u}$ is the unit vector in the direction of $\vec{v}$.

To find $\vec{u}$, we normalize $\vec{v}$ as follows:

$$\vec{u} = \frac{\vec{v}}{|\vec{v}|},$$

where $|\vec{v}| = \sqrt{a^2 + b^2 + c^2}$.

Algorithm

Step 1: Define the function $f(x, y, z) = xy + z$;
Step 2: Compute the gradient $\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}\right)$;
Step 3: Evaluate ∇f at the point $(2, -1, 3)$;
Step 4: Normalize the direction vector $\vec{v} = (1, 2, -1)$ to obtain $\vec{u}$;
Step 5: Compute the directional derivative $D_{\vec{u}} f = \nabla f \cdot \vec{u}$.

MATLAB Program

```
% Define the point and direction vector
x0 = 2; y0 = -1; z0 = 3;
v = [1, 2, -1];

% Define the gradient components of f(x, y, z) = xy + z
grad_f = [-1, 2, 1];

% Calculate the unit vector in the direction of v
unit_v = v / norm(v);

% Calculate the directional derivative
directional_derivative = dot(grad_f, unit_v);

% Display the result
disp(['The value of the directional derivative is: ', num2str(directional_derivat
]);
```

Output

```
The value of the directional derivative is: 0.8165
```

Conclusion

The directional derivative of $f(x, y, z) = xy + z$ at the point $(2, -1, 3)$ in the direction of $\vec{v} = (1, 2, -1)$ is approximately 0.8165.

Exercise Problem

Calculate the directional derivative of $f(x, y, z) = x^2 + y^2 + z^2$ at the point $(1, 1, 1)$ in the direction of $\vec{v} = (2, -1, 2)$.

Practical No. 10

Aim

Write a program to find the equation of the normal line to the surface using MATLAB.

Problem

Given the surface

$$z = x^2 - y^2,$$

find the equation of the normal line to this surface at the point $(1, 1, 0)$.

Theory

The normal line to a surface at a given point is a line that is perpendicular to the tangent plane at that point. For a surface defined by $z = f(x, y)$, the normal vector to the surface at a point (x_0, y_0, z_0) is given by $\nabla f(x, y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, -1\right)$.

Once we find the normal vector, the equation of the normal line passing through (x_0, y_0, z_0) can be written in parametric form:

$$(x, y, z) = (x_0, y_0, z_0) + t \cdot (A, B, C),$$

where (A, B, C) is the normal vector and t is a parameter.

Algorithm

Step 1: Define the surface $z = x^2 - y^2$;
Step 2: Compute the partial derivatives $\frac{\partial z}{\partial x}$ and $\frac{\partial z}{\partial y}$;
Step 3: Evaluate $\frac{\partial z}{\partial x}$ and $\frac{\partial z}{\partial y}$ at the point $(1, 1)$;
Step 4: Form the normal vector $\vec{N} = \left(\frac{\partial z}{\partial x}, \frac{\partial z}{\partial y}, -1\right)$;
Step 5: Use the point $(1, 1, 0)$ and the normal vector to write the parametric equation of the normal line.

MATLAB Program

```
% Define the point and partial derivatives
x0 = 1; y0 = 1; z0 = 0;
dx = 2 * x0; % Partial derivative w.r.t x
dy = -2 * y0; % Partial derivative w.r.t y

% Define the normal vector
```

```
normal_vector = [dx, dy, -1];

% Define the parametric form of the normal line
syms t
x = x0 + normal_vector(1) * t;
y = y0 + normal_vector(2) * t;
z = z0 + normal_vector(3) * t;

% Display the parametric equations
disp([‘x(t) = ’, char(x)]);
disp([‘y(t) = ’, char(y)]);
disp([‘z(t) = ’, char(z)]);
```

Output

```
x(t) = 1 + 2*t
y(t) = 1 - 2*t
z(t) = -t
```

Conclusion

The equation of the normal line to the surface $z = x^2 - y^2$ at the point $(1, 1, 0)$ is given by:

$$x = 1 + 2t, \quad y = 1 - 2t, \quad z = -t.$$

Exercise Problem

Consider the surface $z = x^2 + y^2$. Find the equation of the normal line to this surface at the point $(1, 1, 2)$.

Practical No. 11

Aim

Write a program to find the critical points and classify them as maxima, minima, or saddle points using MATLAB.

Problem

Given the function

$$f(x, y) = x^2 - y^2$$

find the critical points and determine their nature.

Theory

This is a basic example of a two-variable function representing a hyperbolic paraboloid. This problem is used frequently in engineering, physics, and architecture, Optics and Imaging, Quantum Mechanics.

Algorithm

Step 1: Define the function $f(x, y) = x^2 - y^2$;
Step 2: Compute the partial derivatives using the `diff` command;
Step 3: Solve for critical points by setting $f_x = 0$ and $f_y = 0$ using the `solve` command;
Step 4: Compute the second partial derivatives using the `diff` command;
Step 5: Evaluate the Hessian determinant $D = f_{xx} \cdot f_{yy} - (f_{xy})^2$ at the critical points;
Step 6: Determine the nature of the critical points;
Step 7: Display the results of each using `fprintf` command to the output.

MATLAB Program

```
% Define the symbolic variables x and y
syms x y;

% Define the function f(x, y)
f = x^2 - y^2;

% Compute the partial derivatives
f_x = diff(f, x);
f_y = diff(f, y);

% Solve for critical points
critical_points = solve([f_x == 0, f_y == 0], [x, y]);

% Compute the second partial derivatives
f_xx = diff(f_x, x);
f_yy = diff(f_y, y);
f_xy = diff(f_x, y);

% Evaluate the Hessian determinant at the critical point
D = f_xx * f_yy - f_xy^2;

% Substitute critical points into D and classify
```

```
x_val = double(critical_points.x);
y_val = double(critical_points.y);
D_val = subs(D, {x, y}, {x_val, y_val});
f_xx_val = subs(f_xx, {x, y}, {x_val, y_val});

if D_val > 0 && f_xx_val > 0
    disp('The critical point is a local minimum.');
elseif D_val > 0 && f_xx_val < 0
    disp('The critical point is a local maximum.');
elseif D_val < 0
    disp('The critical point is a saddle point.');
else
    disp('The test is inconclusive.');
end
```

Output

```
Critical Point:
[0, 0]

Classification:
Saddle point
```

Conclusion

The program correctly computes the critical points and the critical point at $(0, 0)$ is a saddle point.

Exercise Problem

Given the function:

$$f(x, y) = x^2 + y^2 + 2x,$$

find the critical points and determine whether each is a maximum, minimum, or saddle point.

Practical No. 12

Aim

Write a program to verify Euler's theorem for homogeneous functions using MATLAB.

Problem

Given the function $f(x, y) = x^2 + y^2$, verify Euler's theorem for this homogeneous functions.

Theory

Euler's theorem states that if a function $z = f(x, y)$ is a homogeneous function of degree n, then it satisfies the equation:

$$x\frac{\partial z}{\partial x} + y\frac{\partial z}{\partial y} = nz \tag{2.1}$$

Explanation

A function $f(x, y)$ is called **homogeneous of degree** n if for any scalar λ:

$$f(\lambda x, \lambda y) = \lambda^n f(x, y) \tag{2.2}$$

Differentiating both sides with respect to λ and setting $\lambda = 1$, we obtain the above equation.

Algorithm

Step 1: Define symbolic variables x and y using MATLAB `syms` command.
Step 2: Define the function $f(x, y) = x^2 + y^2$ symbolically.
Step 3: Compute the second partial derivatives using the `diff` command.
Step 4: Apply Euler's theorem: The left-hand and right-hand side of Euler's theorem is computed
Step 5: The left-hand side and right-hand side of the equation are displayed to verify Euler's theorem using `fprintf` command to the output.

MATLAB Program

```
 % Define symbolic variables
syms x y;

% Define the function f(x, y)
f = x^2 + y^2;

% Compute the partial derivatives of f with respect to x and y
df_dx = diff(f, x);
df_dy = diff(f, y);

% Apply Euler's theorem: x*(df/dx) + y*(df/dy)
```

```
lhs = x*df_dx + y*df_dy;

% Calculate the right-hand side of Euler’s theorem: n*f(x, y)
rhs = 2*f;  % The degree of homogeneity is 2

% Display the results
disp(‘Left-hand side of Euler’’s theorem:’);
disp(lhs);
disp(‘Right-hand side of Euler’’s theorem:’);
disp(rhs);
```

Output

```
Left-hand side of Euler’s theorem:
2*x^2 + 2*y^2

Right-hand side of Euler’s theorem:
2*x^2 + 2*y^2
```

Conclusion

The program correctly verify the Euler’s theorem.

Exercise Problem

Given the function:

$$f(x, y) = 3x^2 + 4xy + y^2,$$

verify Euler’s theorem for this homogeneous functions.

Practical No. 13

Aim

Write a program to compute the second-order Taylor expansion using MATLAB.

Problem

Given the function $f(x, y) = x^3 + y^3$, compute the second-order Taylor expansion $f(x, y)$ around (1, 1).

Theory

The second-order Taylor expansion is widely used in optimization, numerical methods, and machine learning for approximating functions near a given point. In the context of functions like $f(x, y) = x^3 + y^3$, it helps in simplifying complex functions for easier analysis and computational efficiency, especially when dealing with multivariable functions. It is also applied in error analysis and for finding local minima or maxima in optimization problems.

Algorithm

Step 1: Define symbolic variables x and y using MATLAB `syms` command;
Step 2: Define the function $f(x, y) = x^3 + y^3$ symbolically;
Step 3: Compute the first and second partial derivatives using the `diff` command;
Step 4: Evaluate the function and its derivatives at the point (1, 1), Use the `subs` function to substitute into the function and its derivatives;
Step 5: Formulate the second-order Taylor expansion;
Step 6: Display the results of each using `fprintf` command to the output.

MATLAB Program

```
 syms x y;

% Define the function f(x, y) = x^3 + y^3
f = x^3 + y^3;

% Compute the first and second partial derivatives
df_dx = diff(f, x);
df_dy = diff(f, y);
d2f_dx2 = diff(df_dx, x);
d2f_dy2 = diff(df_dy, y);
d2f_dxdy = diff(df_dx, y);

% Evaluate the function and its derivatives at the point (1, 1)
f_1_1 = subs(f, {x, y}, {1, 1});
df_dx_1_1 = subs(df_dx, {x, y}, {1, 1});
df_dy_1_1 = subs(df_dy, {x, y}, {1, 1});
d2f_dx2_1_1 = subs(d2f_dx2, {x, y}, {1, 1});
d2f_dy2_1_1 = subs(d2f_dy2, {x, y}, {1, 1});
d2f_dxdy_1_1 = subs(d2f_dxdy, {x, y}, {1, 1});

% Display the Taylor series expansion
taylor_expansion = f_1_1 + df_dx_1_1*(x - 1) + df_dy_1_1*(y - 1) + ...
                   (1/2)*(d2f_dx2_1_1*(x - 1)^2 + d2f_dy2_1_1*(y - 1)^2);
```

```
disp('Second-order Taylor Expansion of f(x, y) around (1, 1):');
disp(taylor_expansion);
```

Output

```
Second-order Taylor Expansion of f(x, y) around (1, 1):
3*x + 3*y + 3*(x - 1)^2 + 3*(y - 1)^2 - 4
```

Conclusion

The second-order Taylor expansion of the function $f(x, y) = x^3 + y^3$ around the point $(1, 1)$ is:

$$f(x, y) \approx 2 + 3(x - 1) + 3(y - 1) + 3(x - 1)^2 + 3(y - 1)^2.$$

Exercise Problem

Given the function:

$$f(x, y) = x^2y + xy^2,$$

compute the second-order Taylor expansion $f(x, y)$ around (1, 1).

Practical No. 14

Aim

Write a program to compute the Jacobian using MATLAB.

Problem

Given the function $f(x, y) = (e^x, e^y)$, find the Jacobian matrix $J(f)$.

Theory

The Jacobian matrix for $f(x, y) = (e^x, e^y)$ is used to analyze how changes in input variables x and y affect the output. It has applications in optimization (e.g., gradient descent), robotics (e.g., kinematics), dynamical systems (e.g., stability analysis), and multivariable calculus (e.g., change of variables in integrals). In these contexts, it helps with tasks like computing gradients, transforming coordinates, and analyzing system behavior near equilibrium points.

Algorithm

Step 1: Define symbolic variables x and y using MATLAB `syms` command;
Step 2: Define the function $f_1 = e^x$ and $f_2 = e^y$ symbolically;
Step 3: Use the `jacobian` function to compute the Jacobian matrix, which contains the partial derivatives of each component of $\mathbf{f}(x, y)$ with respect to x and y;
Step 4: Evaluate the function and its derivatives at the point (1, 1), Use the `subs` function to substitute into the function and its derivatives;
Step 5: Formulate the second-order Taylor expansion;
Step 6: Display the Jacobian matrix using `fprintf` command to the output.

MATLAB Program

```
 syms x y

% Define the functions f1(x, y) and f2(x, y)
f1 = exp(x);   % f1(x, y) = e^x
f2 = exp(y);   % f2(x, y) = e^y

% Compute the Jacobian matrix (partial derivatives)
J = jacobian([f1, f2], [x, y]);

% Display the Jacobian matrix
disp('Jacobian matrix J(f) =');
disp(J);
```

Output

```
Jacobian matrix J(f) =
[exp(x),      0]
[     0, exp(y)]
```

Conclusion

The output Jacobian matrix is:

$$J(\mathbf{f}) = \begin{bmatrix} e^x & 0 \\ 0 & e^y \end{bmatrix}$$

Exercise Problem

Given the function $f(x, y) = (x + y, xy)$, find the Jacobian matrix $J(f)$.

Practical No. 15

Aim

Write a program to compute the limit of indeterminant form using MATLAB.

Problem

Find the limit

$$\lim_{x\to 1} \frac{x^3 - 1}{x - 1}.$$

Theory

This type of limit is used to compute the derivative of the function, which represents the rate of change of the function at that point. It also appears in real-world problems involving rate of change, such as in physics for velocity and acceleration calculations.

Algorithm

Step 1: Define symbolic variables x and y using MATLAB `syms` command;
Step 2: Define the function symbolically;
Step 3: Use `simplify()` command to cancel out common factors;
Step 4: Evaluate the limit Using `limit()` to compute the value of the limit at $x = 1$;
Step 5: Display the result using `fprintf` command to the output.

MATLAB Program

```
 syms x;

% Define the expression
f = (x^3 - 1) / (x - 1);

% Simplify the expression
simplified_expr = simplify(f);
```

```
% Evaluate the limit as x approaches 1
limit_value = limit(simplified_expr, x, 1);

% Display the result
disp('The limit is:');
disp(limit_value);
```

Output

```
The limit is
3
```

Conclusion

By simplifying the expression and evaluating the limit, we find that it equals 3, which corresponds to the derivative of x^3 at $x = 1$.

Exercise Problem

Find the limit

$$\lim_{x \to 1} \frac{x^2 - 1}{x + 1}.$$

Practical No. 16

Aim

Write a program to evaluate the integral by changing the order of integration using MATLAB.

Problem

Evaluate the following double integral by changing the order of integration:

$$I = \int_0^2 \int_0^{2-x} (x+y)\, dy\, dx.$$

Theory

This approach is particularly valuable in problems involving bounded regions, such as in heat transfer analysis or probability density functions. This problem demonstrates a technique useful in fields such as physics, engineering, and probability, where changing the order of integration simplifies complex integrals.

Algorithm

Step 1: Define symbolic variables x and y using MATLAB `syms` command;
Step 2: Define the function symbolically;
Step 3: The `outerintegral` calculates the integral;
Step 4: Use `integral` to evaluate the inner integral over x for a given y from 0 to $2-y$;
Step 5: Use `arrayfun` to apply the inner integral calculation over the range of integration;
Step 6: Display the result using `fprintf` command to the output.

MATLAB Program

```
 % Define the integrand as an anonymous function
f = @(x, y) x + y;

% Set up the outer integral over y from 0 to 2
outer_integral = integral(@(y) arrayfun(@(y_val) ...
    integral(@(x) f(x, y_val), 0, 2 - y_val), y), 0, 2);

% Display the result
fprintf('The value of the integral is: %.4f\n', outer_integral);
```

Output

```
The value of the integral is: 2.6667
```

Conclusion

Reversing the order of integration in this problem allowed us to evaluate the double integral more efficiently, yielding a solution of approximately 2.6667.

Exercise Problem

Evaluate the following double integral by changing the order of integration:

$$I = \int_0^1 \int_{2y}^2 x^2 \, dy \, dx.$$

Practical No. 17

Aim

Write a program for evaluating the double integral in polar form using MATLAB.

Problem

Evaluate the integral

$$\iint_R r \, dr \, d\theta,$$

where R is the region bounded by $r = 2$ and $0 \leq \theta \leq \frac{\pi}{2}$.

Theory

The integrand $f(r, \theta) = r$ is often used in polar coordinate systems, where r represents the radial distance from the origin. This type of function commonly appears in calculations of areas and masses of circular regions. Integrating r over a specified region helps find quantities like the area of a sector or circular region.

Algorithm

Step 1: Define symbolic variables x and y using MATLAB `syms` command;
Step 2: Define the function and limit symbolically;
Step 3: Define integrand $f(r, \theta) = r$;
Step 4: Using `integral` to evaluate the integral;
Step 5: Display the result using `fprintf` command to the output.

MATLAB Program

```
% Define the limits of r and theta
r_min = 0;
r_max = 2;
theta_min = 0;
theta_max = pi/2;

% Define the integrand as an anonymous function
integrand = @(r, theta) r;

% Perform the double integration using integral2 function
result = integral2(@(r, theta) integrand(r, theta), r_min, r_max, theta_min,
theta_max);

% Display the result
disp([‘The value of the integral is: ’, num2str(result)]);
```

Output

```
The value of the integral is: 3.1416
```

Conclusion

Reversing the order of integration in this problem allowed us to evaluate the double integral more efficiently, yielding a solution of approximately 3.1416.

Exercise Problem

Evaluate the integral:

$$\iint_R r\sin(\theta)\,dr\,d\theta,$$

where R is the region bounded by $1 \le r \le 4$ and $0 \le \theta \le \frac{\pi}{3}$.

Practical No. 18

Aim

Write a program to evaluate the triple integral over the given solid region using MATLAB.

Problem

Evaluate the integral

$$\iiint_V (x + y + z)\, dV,$$

where V is the region bounded by $0 \leq x \leq 1$, $0 \leq y \leq 2$, and $0 \leq z \leq 3$.

Theory

Triple integrals are used to calculate quantities like volume, mass, or charge within a 3-dimensional region. For the given problem, we need to find the value of the integral $\iiint_V (x+y+z)\, dV$ on the cuboid defined by the bounds $0 \leq x \leq 1$, $0 \leq y \leq 2$, and $0 \leq z \leq 3$.

Algorithm

Step 1: Define symbolic variables x, y, and z;
Step 2: Define the limits for the integration;
Step 3: Set up the integrand $f(x, y, z) = x + y + z$;
Step 4: Perform the triple integration using MATLABs `integral3` or `integral` function;
Step 5: Display the result using `disp` or `fprintf`.

MATLAB Program

```
% Define the limits of x, y, and z
x_min = 0; x_max = 1;
y_min = 0; y_max = 2;
z_min = 0; z_max = 3;

% Define the integrand as an anonymous function
integrand = @(x, y, z) (x + y + z);

% Perform the triple integration using integral3 function
result = integral3(integrand, x_min, x_max, y_min, y_max, z_min, z_max);

% Display the result
disp([‘The value of the integral is: ’, num2str(result)]);
```

Output

```
The value of the integral is: 18
```

Conclusion

By evaluating the triple integral over the specified cuboid region, we found that the value of the integral is 18, representing the total accumulated quantity $x+y+z$ within the given limits.

Exercise Problem

Evaluate the integral

$$\iiint_V z^2 \, dV,$$

where V is the region given by $x^2 + y^2 + z^2 \leq 1$ (a unit sphere).

Practical No. 19

Aim

Write a program to evaluate the volume of the solid inside the sphere and above the plane using triple integration in spherical coordinates using MATLAB.

Problem

Find the volume of the solid within the sphere $x^2 + y^2 + z^2 = 9$ and above the plane $z = 2$. Use spherical coordinates to set up and evaluate the triple integral.

Theory

In spherical coordinates, the volume element dV is given by $r^2 \sin(\theta)\, dr\, d\theta\, d\phi$. The given sphere has the equation $x^2+y^2+z^2 = 9$, which in spherical coordinates becomes $r = 3$. The plane $z = 2$ will define the upper bound for r, θ, and ϕ. This triple integral will calculate the volume of the region inside the sphere and above the plane.

Algorithm

Step 1: Define the symbolic variables r, θ, and ϕ;
Step 2: Set up the limits for r, θ, and ϕ;
Step 3: Set up the integrand for the volume in spherical coordinates;
Step 4: Perform the triple integration using MATLAB's `integral3` or `integral` function;
Step 5: Display the result using `disp` or `fprintf`.

MATLAB Program

```
% Define the limits of r, theta, and phi
r_min = 0; r_max = 3;
theta_min = 0; theta_max = pi;
phi_min = 0; phi_max = pi/3;  % Plane z=2 limits r for phi

% Define the integrand as an anonymous function
integrand = @(r, theta, phi) r.^2 .* sin(phi);

% Perform the triple integration using integral3 function
result = integral3(integrand, r_min, r_max, theta_min, theta_max, phi_min, phi_max);

% Display the result
disp([‘The volume of the solid is: ’, num2str(result)]);
```

Output

```
The volume of the solid is: 23.5627
```

Conclusion

The volume of the solid inside the sphere $x^2 + y^2 + z^2 = 9$ and above the plane $z = 2$ is approximately 23.5627 cubic units, computed using a triple integral in spherical coordinates.

Exercise Problem

Find the volume of the solid inside the region bounded by the paraboloid $z = x^2 + y^2$ and the plane $z = 4$. Use cylindrical coordinates to set up and evaluate the triple integral.

Practical No. 20

Aim

Write a program to evaluate the Beta function by it's integral definition using MATLAB.

Problem

Evaluate the Beta function

$$B(5,2) = \int_0^1 t^{5-1}(1-t)^{2-1}\,dt = \int_0^1 t^4(1-t)\,dt.$$

Theory

The Beta function $B(x, y)$ is defined as:

$$B(x,y) = \int_0^1 t^{x-1}(1-t)^{y-1}\,dt,$$

for $x, y > 0$. This integral can be evaluated by substitution or by using integration by parts. The Beta function is symmetric, meaning $B(x, y) = B(y, x)$, and has a relationship with the Gamma function given by $B(x,y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$.

Algorithm

Step 1: Define the values of $x = 5$ and $y = 2$;
Step 2: Set up the integrand $t^4(1 - t)$;
Step 3: Perform the integration over the interval $t \in [0, 1]$;
Step 4: Calculate the result using MATLABs `integral` function;
Step 5: Display the result using the `disp` command.

MATLAB Program

```
% Define the integrand as an anonymous function
integrand = @(t) t.^4 .* (1 - t);

% Perform the integration using integral function
result = integral(integrand, 0, 1);

% Display the result
disp(['The value of B(5, 2) is: ', num2str(result)]);
```

Output

```
The value of B(5, 2) is: 0.03333
```

Conclusion

The value of the Beta function $B(5, 2)$ is approximately 0.03333, as calculated using the integral definition. This confirms the evaluation of the Beta function through direct integration.

Exercise Problem

Evaluate the Beta function $B(3, 3)$ by computing the integral

$$B(3,3) = \int_0^1 t^{3-1}(1-t)^{3-1}\, dt = \int_0^1 t^2(1-t)^2\, dt.$$

Practical No. 21

Aim

Write a program to evaluate the Gamma function by its integral definition using MATLAB.

Problem

Compute the value of the Gamma function

$$\Gamma\left(\frac{3}{2}\right) = \int_0^\infty t^{\frac{3}{2}-1}e^{-t}\, dt = \int_0^\infty t^{\frac{1}{2}}e^{-t}\, dt.$$

Theory

The Gamma function $\Gamma(n)$ is defined for $n > 0$ by the integral

$$\Gamma(n) = \int_0^\infty t^{n-1}e^{-t}\, dt.$$

For non-integer values, the Gamma function does not directly correspond to factorials but can be related to them through recurrence and reflection formulas. Specifically, for half-integer values, the Gamma function has a relationship with the square root of π, such as $\Gamma\left(\frac{3}{2}\right) = \frac{1}{2}\sqrt{\pi}$.

Algorithm

Step 1: Define the value of $n = \frac{3}{2}$;
Step 2: Set up the integrand $t^{\frac{1}{2}}e^{-t}$;
Step 3: Perform the integration over the interval $t \in [0, \infty)$ using MATLABs `integral` function;
Step 4: Calculate the result and verify with the known value $\Gamma\left(\frac{3}{2}\right) = \frac{\sqrt{\pi}}{2}$;
Step 5: Display the result.

MATLAB Program

```
% Define the integrand as an anonymous function
integrand = @(t) t.^(1/2) .* exp(-t);

% Perform the integration over the interval [0, Inf] using integral function
result = integral(integrand, 0, Inf);

% Display the result
disp([‘The value of Gamma(3/2) is: ’, num2str(result)]);
```

Output

```
The value of Gamma(3/2) is: 0.88623
```

Conclusion

The value of the Gamma function $\Gamma\left(\frac{3}{2}\right)$ is approximately 0.88623, which is equivalent to $\frac{\sqrt{\pi}}{2}$. This confirms the known result of the Gamma function for half-integer values.

Exercise Problem

Calculate $\Gamma\left(\frac{5}{2}\right)$ using the recurrence relation:

$$\Gamma\left(\frac{5}{2}\right) = \frac{3}{2} \cdot \Gamma\left(\frac{3}{2}\right).$$

Use the value of $\Gamma\left(\frac{3}{2}\right) = 0.88623$ in your computation.

Practical No. 22

Aim

Write a program to evaluate the Gamma function by its integral definition using MATLAB.

Problem

Compute the value of the Gamma function

$$\Gamma\left(\frac{1}{2}\right) = \int_0^\infty t^{\frac{1}{2}-1}e^{-t}\,dt = \int_0^\infty t^{-\frac{1}{2}}e^{-t}\,dt.$$

Theory

The Gamma function $\Gamma(n)$ is defined by the integral

$$\Gamma(n) = \int_0^\infty t^{n-1}e^{-t}\,dt,$$

for $n > 0$. The value $\Gamma\left(\frac{1}{2}\right)$ is significant, as it has a special relationship with π, specifically:

$$\Gamma\left(\frac{1}{2}\right) = \sqrt{\pi}.$$

This result is useful in various mathematical and statistical applications.

Algorithm

Step 1: Define the value of $n = \frac{1}{2}$;
Step 2: Set up the integrand $t^{-\frac{1}{2}}e^{-t}$;
Step 3: Perform the integration over the interval $t \in [0, \infty)$ using MATLABs `integral` function;
Step 4: Calculate the result and verify it matches $\sqrt{\pi}$;
Step 5: Display the result.

MATLAB Program

```
% Define the integrand as an anonymous function
integrand = @(t) t.^(-1/2) .* exp(-t);

% Perform the integration over the interval [0, Inf] using integral function
result = integral(integrand, 0, Inf);

% Display the result
disp(['The value of Gamma(1/2) is: ', num2str(result)]);
```

Output

```
The value of Gamma(1/2) is: 1.7725
```

Conclusion

The computed value of $\Gamma\left(\frac{1}{2}\right)$ is approximately 1.7725, which is equal to $\sqrt{\pi}$. This confirms the known relationship of the Gamma function at half-integer values.

Exercise Problem

Evaluate $\Gamma\left(\frac{5}{2}\right)$.

Practical No. 23

Aim

Write a program to evaluate the integral by the Gamma function using MATLAB.

Problem

Compute the integral

$$\int_0^\infty x^{2.5}e^{-3x}\,dx.$$

Theory

The Gamma function $\Gamma(n)$ is defined as:

$$\Gamma(n) = \int_0^\infty t^{n-1}e^{-t}\,dt.$$

To match this form, we can perform a substitution. For an integral of the form $\int_0^\infty x^{n-1}e^{-ax}\,dx$ (with $a > 0$), we have

$$\int_0^\infty x^{n-1}e^{-ax}\,dx = \frac{\Gamma(n)}{a^n}.$$

Using this result, we can evaluate the given integral by setting $n = 3.5$ and $a = 3$.

Algorithm

Step 1: Identify the values of $n = 3.5$ and $a = 3$ from the integral;
Step 2: Use the formula $\int_0^\infty x^{n-1}e^{-ax}\,dx = \frac{\Gamma(n)}{a^n}$;
Step 3: Substitute $n = 3.5$ and $a = 3$ into the formula;
Step 4: Calculate $\Gamma(3.5)$ using known values or MATLAB;
Step 5: Compute the final result.

Solution

We apply the formula

$$\int_0^\infty x^{2.5}e^{-3x}\,dx = \frac{\Gamma(3.5)}{3^{3.5}}.$$

Using the known value $\Gamma(3.5) = \frac{3\sqrt{\pi}}{4}$, we find

$$\int_0^\infty x^{2.5}e^{-3x}\,dx = \frac{3\sqrt{\pi}}{4\times 3^{3.5}} = \frac{3\sqrt{\pi}}{4\times 15.5885} \approx 0.1061.$$

MATLAB Program

```
% Define constants
a = 3;
n = 3.5;

% Calculate Gamma(n)
gamma_n = gamma(n);

% Calculate the result
result = gamma_n / a^n;

% Display the result
disp([‘The value of the integral is: ’, num2str(result)]);
```

Output

```
The value of the integral is: 0.1061
```

Conclusion

The value of the integral $\int_0^\infty x^{2.5}e^{-3x}\,dx$ is approximately 0.1061, obtained by expressing it in terms of the Gamma function.

Exercise Problem

Evaluate the integral

$$\int_0^\infty x^{1.5} e^{-4x}\, dx.$$

Use the formula $\int_0^\infty x^{n-1} e^{-ax}\, dx = \frac{\Gamma(n)}{a^n}$ and $\Gamma(2.5) = \frac{3\sqrt{\pi}}{4}$.

Practical No. 24

Aim

Write a program to verify the relationship between the Beta and Gamma functions using MATLAB.

Problem

Show that

$$B(x, y) = \frac{\Gamma(x)\,\Gamma(y)}{\Gamma(x + y)}.$$

Theory

The Beta and Gamma functions are special functions widely used in calculus and mathematical analysis

- **Beta Function** $B(x, y)$:

$$B(x, y) = \int_0^1 t^{x-1}(1 - t)^{y-1}\, dt.$$

  The Beta function often appears in probability theory and is used to model distribution functions.

- **Gamma Function** $\Gamma(x)$:

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t}\, dt.$$

  The Gamma function generalizes factorials to non-integer values, where $\Gamma(n) = (n - 1)!$ for positive integers n.

The relationship between the Beta and Gamma functions is given by

$$B(x, y) = \frac{\Gamma(x)\,\Gamma(y)}{\Gamma(x + y)}.$$

Algorithm

1. Define the Beta function in its integral form;
2. Use the substitution $t = \frac{u}{u+v}$, where $u, v \geq 0$;
3. Simplify the integral to obtain the form of the Gamma function;
4. Conclude that $B(x, y) = \frac{\Gamma(x)\,\Gamma(y)}{\Gamma(x+y)}$.

Solution

Starting from the definition of the Beta function:

$$B(x, y) = \int_0^1 t^{x-1}(1-t)^{y-1}\, dt.$$

1. Substitute $t = \frac{u}{u+v}$, $1 - t = \frac{v}{u+v}$, where $u, v \geq 0$.
Then $dt = \frac{v}{(u+v)^2}\, du$, transforming the integral as

$$B(x, y) = \int_0^\infty \int_0^\infty \frac{u^{x-1}v^{y-1}}{(u+v)^{x+y}}\, du\, dv.$$

2. Separate the integrals and simplify to

$$B(x, y) = \frac{\Gamma(x)\,\Gamma(y)}{\Gamma(x+y)}.$$

This proves that

$$B(x, y) = \frac{\Gamma(x)\,\Gamma(y)}{\Gamma(x+y)}.$$

MATLAB Program

```
% Define symbolic variables
syms x y;

% Define Gamma functions
Gamma_x = gamma(x);
Gamma_y = gamma(y);
Gamma_xy = gamma(x + y);

% Calculate Beta function using the relationship
Beta_xy = Gamma_x * Gamma_y / Gamma_xy;

% Display the result
disp(['Beta(x, y) = ', char(Beta_xy)]);
```

Output

```
Beta(x, y) = Gamma(x) * Gamma(y) / Gamma(x + y)
```

Conclusion

The relationship between the Beta and Gamma functions has been verified as $B(x, y) = \frac{\Gamma(x)\,\Gamma(y)}{\Gamma(x+y)}$, showing how these functions are interconnected. This formula is instrumental in calculations in various fields, including probability and statistics.

Exercise Problem

Using the relationship between Beta and Gamma functions, evaluate

$$B(3, 2).$$

Chapter 3

Real Analysis

Practical No. 1

Aim

To write a MATHEMATICA program to find limit of a function.

Problem

Write a MATHEMATICA program to compute $\lim_{x\to 0} \frac{\sin x}{x}$.

Algorithm

Step 1: Write Limit[Sin[x]/x, x → 0]

Step 2: Press Shift + Enter

MATHEMATICA Code

```
Limit[Sin[x]/x, x -> 0]
```

Output

```
1
```

Exercise Problems

1. Write a MATHEMATICA program to compute $\lim_{x\to 0} \sin x$.

2. Write a MATHEMATICA program to compute $\lim_{x\to 0} \cos x$.

Practical No. 2

Aim

Generate a power series expansion for a function using MATHEMATICA.

Problem

Find Power series for the exponential function around the origin, using MATHEMATICA.

Algorithm

Step 1: Write Series[Exp[x], {x, 0, 10}]

Step 2: Press Shift + Enter

MATHEMATICA Code

```
Series[Exp[x], {x, 0, 10}]
```

Output

$1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \frac{x^5}{120} + O[x]^6$

Exercise Problems

1. Write a MATHEMATICA program to find Power series for the function $f(x) = \sin x$ around $x = 0$.
2. Write a MATHEMATICA program to find Power series for the function $f(x) = \cos x$ around $x = 0$.

Practical No. 3

Aim

Evaluate integration, using MATHEMATICA.

Problem

Find the value of $\int x^n dx$.

Algorithm

Step 1: Write

```
Integrate[x^n, x]
```

Step 2: Press Shift + Enter

MATHEMATICA Code

```
Integrate[x^n, x]
```

Output

$\frac{x^{1+n}}{1+n}$

Exercise Problems

1. Write a MATHEMATICA program to compute $\int \sin x \, dx$.
2. Write a MATHEMATICA program to compute $\int \cos x \, dx$.

Practical No. 4

Aim

Evaluate improper integral, using MATHEMATICA.

Problem

Evaluate $\int_1^\infty \frac{1}{x^2} \, dx$.

Algorithm

Step 1: Write

```
Integrate[1/(x^2), {x, 1, Infinity}]
```

Step 2: Press Shift + Enter

MATHEMATICA Code

```
Integrate[1/(x^2), {x, 1, Infinity}]
```

Output

```
1
```

Exercise Problems

1. Write a MATHEMATICA program to compute $\int_{-\infty}^{\infty} xe^{-x^2}\,dx$.
2. Write a MATHEMATICA program to compute $\int_0^3 \frac{1}{\sqrt{3-x}}\,dx$.

Practical No. 5

Aim

To write a MATHEMATICA program to find the value of a series.

Problem

Find the value of $\sum_{n=1}^{\infty} \frac{1}{n^2}$, using MATHEMATICA.

Algorithm

Step 1: Write Sum[1/n^2, {n, 1, Infinity}]

Step 2: Press Shift + Enter

MATHEMATICA Code

```
Sum[1/n^2, {n, 1, Infinity}]
```

Output

$\frac{\pi^2}{6}$

Exercise Problems

1. Write a MATHEMATICA program to compute $\displaystyle\sum_{n=1}^{100} \frac{1}{n^2}$.
2. Write a MATHEMATICA program to compute $\displaystyle\sum_{n=1}^{50} \frac{1}{n^3}$.

Chapter 4

Ordinary Differential Equations and Vector Calculus

Practical No. 1

Aim

To write an ordinary differential equation in MATLAB.

Problem

Write the following ordinary differential equation in MATLAB.

$$\frac{d^2y}{dx^2} - 2\frac{dy}{dx} - 3y = 65\cos 2x$$

Theory

An ordinary differential equation of order n in the dependent variable y and independent variable x is an equation that can be expressed in the following form

$$a_0(x)\frac{d^ny}{dx^n} + a_1(x)\frac{d^{n-1}y}{dx^{n-1}} + ... + a_{n-1}(x)\frac{dy}{dx} + a_n(x)y = b(y),$$

where, $a_0 \neq 0$.

Algorithm

Step 1: Define the dependent and independent variables using the inbuilt function,
Step 2: Use diff function to define the derivatives,
Step 3: Use the display function to display the output on the command window,
Step 4: Click on the Run button on the top panel.

Program

```
% Define symbolic variables
syms x y(x)

% Define the differential equation
eqn = diff(y, x, 2) - 2*diff(y, x) - 3*y == 65*cos(2*x);

% Convert equation to a readable string for display
eqn_str = char(eqn);

% Display the differential equation
disp('The differential equation is:');
disp(eqn_str);
```

In the above program, the text after % in the same line represents the comments. This is only written to increase the readability of the code.

Output

The differential equation is

```
diff(y(x), x, x) - 2*diff(y(x), x) - 3*y(x) == 65*cos(2*x)
```

Exercise Problem

Write the following differential equation in MATLAB

$$\frac{d^4y}{dx^4} + x^2\frac{d^3y}{dx^3} + x^3\frac{dy}{dx} = xe^x.$$

Practical No. 2

Aim

To find the order of an ordinary differential equation (ODE) in MATLAB.

Problem

Find the order of the following ordinary differential equation in MATLAB

$$\frac{d^4y}{dx^4} + 3\left(\frac{dy}{dx}\right)^5 + 5y = 0.$$

Theory

The order of an ordinary differential equation is the order of the highest-order derivative appearing explicitly in the ODE.

Algorithm

Step 1: Define the ODE using the inbuilt systems and diff function,
Step 2: Use the inbuilt numel(OdeToVectorField()) to compute the order of the ODE,
Step 3: Use the inbuilt disp function to display the solution on the command window,
Step 4: Click on the Run button on the top panel.

Program

```
% Define symbolic variables
syms x y(x)

% Define the differential equation
diff_eqn = diff(y, x, 4) + 3*(diff(y, x, 2))^5 + 5*y == 0; % Differential equatio

% Display the order of the differential equation
disp('The order of the differential equation is:');
disp(numel(odeToVectorField(diff_eqn)));
```

Output

```
The order of the differential equation is:
     4
```

Exercise Problem

Find the order of the following ordinary differential equation in MATLAB.

$$\frac{d^2y}{dx^2} - 2\frac{dy}{dx} - 3y = 65\cos 2x.$$

Practical No. 3

Aim

To find the solution of an ordinary differential equation (ODE) in MATLAB.

Problem

Find the general solution of the linear differential equation in MATLAB

$$\frac{dy}{dx} = 2x.$$

Theory

The above differential equation is solved theoretically by the method of separation of variables.

Algorithm

Step 1: Define the ODE using inbuilt systems and diff function,
Step 2: Use the inbuilt dsolve() to compute the solution of the ODE,
Step 3: Use the inbuilt disp function to display the output on the command window,
Step 4: Click on the Run button on the top panel.

Program

```
% Define symbolic variables
syms x y(x)

% Define the first derivative
Dy = diff(y);

% Define the differential equation
ode = diff(y, x) == 2*x;

% Solve the differential equation
ysol(x) = dsolve(ode);

% Display the solution of the differential equation
disp('The solution of the differential equation is:');
disp(ysol(x));
```

Output

```
The solution of the differential equation is:
x^2 + C1
```

Exercise Problem

Find the general solution of the linear differential equation in MATLAB

$$\frac{dy}{dx} = \frac{1}{(x-1)^2}.$$

Practical No. 4

Aim

To find the solution of a second-order differential equation (ODE) in MATLAB.

Problem

Find the general solution of the second-order differential equation in MATLAB

$$\frac{d^2y}{dx^2} - 7\frac{dy}{dx} + 12y \;=\; 0.$$

Theory

Theoretically, the above differential equation is solved by assuming a general solution of the form $c_1e^{m_1x} + c_2e^{m_2x}$ and then finding the respective derivatives and substituting them in the differential equation.

Algorithm

Step 1: Define the ODE using the inbuilt systems and diff function,
Step 2: Use the inbuilt dsolve() to compute the solution of the ODE,
Step 3: Use the inbuilt disp function to display the output on the command window,
Step 4: Click on the Run button on the top panel.

Program

```
% Define symbolic variable
syms y(x)

% Define the differential equation
```

```
ode = diff(y, x, 2) - 7*diff(y, x) + 12*y == 0;

% Solve the differential equation
ysol(x) = dsolve(ode);

% Display the solution of the differential equation
disp('The solution of the differential equation is:');
disp(ysol(x));
```

Output

The solution of the differential equation is:

```
C1*exp(3*x) + C2*exp(4*x)
```

Exercise Problem

Find the general solution of the second-order differential equation in MATLAB

$$\frac{d^2y}{dx^2} - 2\frac{dy}{dx} - 8y \ = \ 0.$$

Practical No. 5

Aim

To find the solution of a third-order differential equation (ODE) in MATLAB.

Problem

Find the solution of the third-order differential equation in MATLAB

$$x^3\frac{d^3y}{dx^3} - 3x^2\frac{d^2y}{dx^2} + 6x\frac{dy}{dx} - 6y = 0,$$

which satisfies $y(2) = 0,\ y'(2) = 0 \quad and \quad y''(2) = 6$.

Theory

The above differential equation is solved theoretically by first assuming a general solution of the form $c_1e^{m_1x}+c_2e^{m_2x}+c_3e^{m_3x}$ and then finding the respective derivatives and substituting them in the differential equation. The given conditions are then used to find the particular solution.

Algorithm

Step 1: Define the ODE using the inbuilt systems and diff function,
Step 2: Define the given conditions,
Step 3: Use the inbuilt dsolve() to compute the solution of the ODE,
Step 4: Use the inbuilt disp function to display the output on the command window,
Step 5: Click on the Run button on the top panel.

Program

```
% Define symbolic variables
syms x y(x)

% Define derivatives
Dy = diff(y, x);
D2y = diff(y, x, 2);

% Define the differential equation
ode = x^3 * diff(y, x, 3) - 3*x^2 * diff(y, x, 2) + 6*x * diff(y, x) - 6*y == 0;
```

```
% Define boundary/initial conditions
cond1 = y(2) == 0;
cond2 = Dy(2) == 2;
cond3 = D2y(2) == 6;

% Combine conditions
conds = [cond1 cond2 cond3];

% Solve the differential equation with conditions
ysol(x) = dsolve(ode, conds);

% Display the solution of the differential equation
disp('The solution of the differential equation is:');
disp(ysol(x));
```

Output

```
The solution of the differential equation is:
x^3 - 3*x^2 + 2*x
```

Exercise Problem

Find the general solution of the third-order differential equation in MATLAB

$$\frac{d^3y}{dx^3} - 2\frac{d^2y}{dx^2} - 4\frac{dy}{dx} + 8y = 0.$$

Practical No. 6

Aim

To find the solution of a differential equation in MATLAB.

Problem

Find the general solution to the given differential equation in MATLAB

$$\frac{d^2y}{dx^2} - 4\frac{dy}{dx} + 4y = -8sin(2x),$$

which satisfies $y(0) = 2 \quad and \quad y'(0) = 4$.

Theory

Theoretically, the above differential equation is solved by assuming a general solution of the form $c_1e^{m_1x} + c_2e^{m_2x}$ and then finding the respective derivatives and substituting them in the differential equation. The boundary conditions are then used to find the particular solution.

Algorithm

Step 1: Define the ODE using the inbuilt systems and diff function.
Step 2: Define the given boundary conditions.
Step 3: Use the inbuilt dsolve() to compute the solution of the ODE.
Step 4: Use the inbuilt disp function to display the output on the command window.
Step 5: Click on the Run button on the top panel

Program

```
% Define symbolic variable
syms y(x)

% Define the differential equation
ode = diff(y, x, 2) - 4*diff(y, x) + 4*y == -8*sin(2*x);

% Define boundary/initial conditions
cond1 = y(0) == 2;
cond2 = diff(y, x)(0) == 4;

% Combine conditions
conds = [cond1 cond2];

% Solve the differential equation with conditions
ysol(x) = dsolve(ode, conds);

% Simplify the solution
ysol = simplify(ysol);

% Display the solution of the differential equation
disp('The solution of the differential equation is:');
disp(ysol);
```

Output

```
The solution of the differential equation is:
3*exp(2*x) - cos(2*x) - 2*x*exp(2*x)
```

Exercise Problem

Find the general solution to the given differential equation in MATLAB

$$\frac{d^2y}{dx^2} + \frac{dy}{dx} - 6y = 0,$$

which satisfies $y(0) = 6 \quad and \quad y'(0) = 2.$

Practical No. 7

Aim

To find the solution to a boundary value problem in MATLAB.

Problem

Find the solution to the boundary value problem in MATLAB

$$\frac{d^2y}{dx^2} - y \ = \ 0,$$

which satisfies $y(0) = 0 \quad and \quad y(1) = 1.$

Theory

Theoretically, the above differential equation is solved by assuming a general solution of the form $c_1e^{m_1x} + c_2e^{m_2x}$ and then finding the respective derivatives and substituting them in the differential equation. The boundary conditions are then used to find the particular solution.

Algorithm

Step 1: Define the ODE using the inbuilt systems and diff function,
Step 2: Define the given boundary conditions,
Step 3: Use the inbuilt dsolve() to compute the solution of the ODE,
Step 4: Use the inbuilt disp function to display the output on the command window,
Step 5: Click on the Run button on the top panel.

Program

```
% Define symbolic variable
syms y(x)

% Define the differential equation
ode = diff(y, x, 2) - y == 0;

% Define boundary conditions
cond1 = y(0) == 0;
cond2 = y(1) == 1;

% Combine conditions
conds = [cond1 cond2];

% Solve the boundary value problem
ysol(x) = dsolve(ode, conds);

% Simplify the solution
ysol = simplify(ysol);

% Display the solution of the boundary value problem
disp('The solution of the boundary value problem is:');
disp(ysol(x));
```

Output

```
The solution of the boundary value problem is:
(exp(1 - x)*(exp(2*x) - 1))/(exp(2) - 1)
```

Exercise Problem

Find the solution to the boundary value problem in MATLAB

$$\frac{d^2y}{dx^2} - 3\frac{dy}{dx} + 2y \ = \ 0,$$

which satisfies $y(0) = 0 \quad and \quad y(1) = 10.$

Practical No. 8

Aim

To find the solution to an initial value problem in MATLAB.

Problem

Find the general solution to the initial value problem in MATLAB

$$\frac{dy}{dx} = -\frac{x}{y},$$

which satisfies $y(3) = 4$.

Theory

The above differential equation is solved theoretically by the method of separation of variables. The initial conditions are then used to find the particular solution.

Algorithm

Step 1: Define the ODE using the inbuilt systems and diff function,
Step 2: Define the given initial conditions,
Step 3: Use the inbuilt dsolve() to compute the solution of the ODE,
Step 4: Use the inbuilt disp function to display the output on the command window,
Step 5: Click on the Run button on the top panel.

Matlab Code

```
syms y(x)
Dy = diff(y);
ode = diff(y,x) == -(x/y);
cond = y(3) == 4;
ysol(x) = dsolve(ode,cond);
disp('The solution of the differential equation is:');
disp(ysol(x));
```

Output

```
 The solution of the differential equation is:
(25 - x^2)^(1/2)
```

Exercise Problem

Find the general solution to the initial value problem in MATLAB

$$\frac{dy}{dx} = 6x,$$

which satisfies $y(1) = 8$.

Practical No. 9

Aim

To find the solution of a nonlinear differential equation in MATLAB.

Problem

Find the solution of the nonlinear differential equation in MATLAB

$$\frac{dy}{dx} + y \;=\; 1.$$

Theory

The above differential equation is solved theoretically first taking square root on both sides and then solving both linear differential equations independently. The initial conditions are then used to find the solution for both equations.

Algorithm

Step 1: Define the ODE using the inbuilt systems and diff function,
Step 2: Define the given initial conditions,
Step 3: Use the inbuilt dsolve() to compute the solution of the ODE,
Step 4: Use the inbuilt disp function to display the output on the command window,
Step 5: Click on the Run button on the top panel.

MATLAB Code

```
syms y(x)
ode = (diff(y,x)+y)^2 == 1;
cond = y(0) == 0;
```

```
ysol(x) = dsolve(ode,cond);
disp('The solution of the non-linear differential equation is:');
disp(ysol(x));
```

Output

```
The solution of the non-linear differential equation is:
exp(-x) - 1
1 - exp(-x)
```

Exercise Problem

Find the solution of the nonlinear differential equation in MATLAB

$$\left(\frac{dy}{dx}\right)^2 + 5y = 20$$

which satisfies $y(0) = 0$.

Practical No. 10

Aim

To find the solution of a differential equation in MATLAB.

Problem

Find the solution of the famous Airy equation in MATLAB

$$\frac{d^2y}{dx^2} = xy.$$

Theory

The Airy function (or Airy function of the first kind) Ai(x) is a special function named after the British astronomer George Biddell Airy. The function Ai(x) and the related function Bi(x), are linearly independent solutions to the differential equation $\frac{d^2y}{dx^2} = xy$ known as the Airy equation or the Stokes equation. The Airy equation is the simplest second-order linear differential equation with a turning point.

Algorithm

Step 1: Define the ODE using the inbuilt systems and diff function,
Step 2: Use the inbuilt dsolve() to compute the solution of the ODE,
Step 3: Use the inbuilt disp function to display the output on the command window,
Step 4: Click on the Run button on the top panel.

MATLAB Code

```
syms y(x)
ode = diff(y,x,2) == x*y;
ysol(x) = dsolve(ode);
disp('The solution of the Airy equation is:');
disp(ysol(x));
```

Output

```
The solution of the Airy equation is:
C1*airy(0, -(x*(1 + 3^(1/2)*1i))/2) + C2*airy(2, -(x*(1 + 3^(1/2)*1i))/2)
```

More information about the inbuilt Airy function can be found by typing with the help Airy command in the command window, as shown below.

Exercise Problem

Find the solution of the Airy-type equation in the MATLAB

$$\frac{d^2y}{dx^2} = 4xy.$$

Practical No. 11

Aim

To find the Puiseux series solution in MATLAB.

Problem

Find the Puiseux series solution for the given differential equation in MATLAB

$$(x^2+1)\frac{d^2y}{dx^2} - 2x\frac{dy}{dx} + y \;=\; 0,$$

which satisfies $y'(0) = 1 \quad and \quad y(0) = 5$.

Theory

Theoretically, the above differential equation is solved by assuming a general solution of the form $c_1e^{m_1x} + c_2e^{m_2x}$ and then finding the respective derivatives and substituting them in the differential equation. The boundary conditions are then used to find the particular series solution.

Algorithm

Step 1: Define the ODE using the inbuilt systems and diff function,
Step 2: Define the given initial conditions,
Step 3: Use the inbuilt dsolve() to compute the solution of the ODE,
Step 4: Use the inbuilt disp function to display the output on the command window,
Step 5: Click on the Run button on the top panel.

Program

```
 syms y(x) a
 ode = (x^2+1)*diff(y,x,2)-x*diff(y,x)+y == 0;
 Dy = diff(y,x);
 cond = [Dy(0) == 1; y(0) == 5];
 ysol(x) = dsolve(ode,cond,‘ExpansionPoint’,0);
 disp(‘The solution of the differential equation is:’);
 disp(ysol(x));
```

Output

```
 The solution of the differential equation is:
(5*x^4)/24 - (5*x^2)/2 + x + 5
```

Exercise Problem

Find the series solution of the given differential equation in MATLAB

$$(x^2+1)\frac{d^2y}{dx^2} - x\frac{dy}{dx} + y \;=\; 0,$$

which satisfies $y'(0) = 1 \quad and \quad y(0) = 5.$

Practical No. 12

Aim

To check whether the differential equation is exact or not in MATLAB.

Problem

Check whether the given differential equation is exact or not in MATLAB

$$(3y+4x^2)dx + (2x+3x^2y)dy \;=\; 0.$$

Theory

A differential equation is of the form $M(x,y)dx + N(x,y)dy = 0$, where M and N have continuous partial derivatives at all points in the domain D is said to be exact if $\frac{\partial M}{\partial y} = \frac{\partial N}{\partial x}$.

Algorithm

Step 1: Define the M and N functions,
Step 2: Use the if-else command to check whether the differential equation is exact or not,
Step 3: Use the inbuilt disp function to display the output on the command window,
Step 4: Click on the Run button on the top panel.

Program

```
syms x y c
% The general form of exact differential equation is P dx + Q dy =0
 P = 3*y+4*x*x;
 Q = 2*x+3*x*x*y;
 if diff(P,y)==diff(Q,x)
```

```
    disp(‘‘Yes, the differential equation is exact’’);
else
    disp(‘‘No, the differential equation is not exact’’);
end
```

Output

```
No, the differential equation is not exact
```

Exercise Problem

Check whether the given differential equation is exact or not in MATLAB

$$(3x^2y^2 + 4x^3y^3)dx + (2x^3y + 3x^4y^2)dy \;=\; 0.$$

Practical No. 13

Aim

To find the solution to an exact differential equation in MATLAB.

Problem

Find the general solution to the given exact differential equation in MATLAB $(3x^2 + 4xy)dx + (2x^2 + 2y)dy \;=\; 0$.

Theory

A differential equation is of the form $M(x, y)dx + N(x, y)dy = 0$, where M and N have continuous partial derivatives at all points in the domain D is said to be exact if $\frac{\partial M}{\partial y} = \frac{\partial N}{\partial x}$.

Algorithm

Step 1: Define M and N functions.
Step 2: Use the inbuilt int command to compute the integral of the function M with respect to x and N with respect to y,
Step 3: Use the inbuilt disp function to display the output on the command window,
Step 4: Click on the Run button on the top panel.

Program

```
syms x y
% The general form of exact differential equation is P dx + Q dy = 0
P = 3*x^3 + 4*x*y;
Q = 2*x^2*y + 2*y;
f = int(P, x) + subs(int(Q, y), x, 0) + c;
disp('The solution of the differential equation is:');
disp(f);
```

Output

```
The solution of the exact differential equation is:
c + y^2 + x^2*(x + 2*y)
```

Exercise Problem

Find the general solution to the given exact differential equation in MATLAB

$$(2xy - sinx)dx + (x^2 - cosy)dy \ = \ 0.$$

Practical No. 14

Aim

Find the orthogonal trajectory in MATLAB.

Problem

Find the orthogonal trajectory of the family of parabolas $y = cx^2$ in MATLAB.

Theory

An orthogonal trajectory refers to a family of curves that intersect another family of curves at right angles.

Algorithm

Step 1: Define the variables using the syms command and the given equation,
Step 2: Find the derivative of the given equation using the diff command,
Step 3: Eliminate the constants using the subs command,
Step 4: Define the negative reciprocal of the right-hand side of the differential equation using the subs command,
Step 5: Use the inbuilt dsolve() to compute the solution of the obtained differential equation,
Step 6: Use the inbuilt disp function to display the output on the command window,
Step 7: Click on the Run button on the top panel.

Program

```
%Define the symbolic variables
syms x y(x) c
%Define the given equation whose orthogonal trajectory is to be computed
eq=y==c*x*x;
%find the differential equation
deq=diff(eq,x);
%Eliminate the constants ---This will vary according to question
deq1=subs(deq,c,(y/(x*x)));
f=rhs(deq1);
%Take negative recripocal of rhs
deq2=subs(deq1,f,-1/f);
%Finding the solution
sol=dsolve(deq2);
disp('The orthogonal trajectory to the family of parabolas is:');
disp(sol);
```

Output

```
 The orthogonal trajectory to the family of parabolas is:
 (2^(1/2)*(- x^2 + C1)^(1/2))/2
-(2^(1/2)*(- x^2 + C1)^(1/2))/2
```

Exercise Problem

Find the orthogonal trajectory of the family of parabolas $x^2 + y^2 = c^2$ in MATLAB.

Practical No. 15

Aim

Find the general solution of an ODE using the method of variation of parameters in MATLAB.

Problem

Find the general solution of the given ODE in MATLAB using the method of variation of parameters

$$\frac{d^2y}{dx^2} - y = 2x^2 - x - 3.$$

Theory

The method of variation of parameters is to solve the inhomogeneous linear ordinary differential equation.

Algorithm

Step 1: Define the variables using the syms command enter the coefficients of the second-order differential equation,
Step 2: Solve for the roots of the Auxiliary equation using solve command,
Step 3: On the basis of the roots of the Auxiliary equation, define the complementary function,
Step 4: Define the Wronskian,
Step 5: Define the general solution and particular integrals,
Step 6: Use the inbuilt disp function to display the output on the command window,
Step 7: Click on the Run button on the top panel.

Program

```
syms r c1 c2 x
E = input('Enter the coefficients of the 2nd ODE: ');
X = input('Enter the R.H.S of the 2nd ODE: ');

% Coefficients of the 2nd Order Differential Equations
a = E(1);
b = E(2);
c = E(3);

% Auxiliary Equation
AE = a*r^2 + b*r + c;
```

```
s = solve(AE, r);

% Roots of Auxiliary Equation (AE)
r1 = s(1);
r2 = s(2);

% Determinant of Auxiliary Equation (AE)
D = b^2 - 4*a*c;

if D > 0
    y1 = exp(r1*x);
    y2 = exp(r2*x);
    % Complementary Function
    cf = c1*y1 + c2*y2;
elseif D == 0
    y1 = exp(r1*x);
    y2 = x*exp(r2*x);
    % Complementary Function
    cf = c1*y1 + c2*y2;
else
    alpha = real(r1);
    beta = imag(r2);
    y1 = exp(alpha*x) * cos(beta*x);
    y2 = exp(alpha*x) * sin(beta*x);
    % Complementary Function
    cf = c1*y1 + c2*y2;
end

% Wronskian Calculation
W = simplify(y1*diff(y2, x) - y2*diff(y1, x));

% Particular Integral
PI = simplify((-y1) * int(y2*X/W, x) + (y2) * int(y1*X/W, x));

% General Solution
GS = simplify(cf + PI);
```

Output

```
cf =

c_2 e^x + c_1 e^{-x}

PI =
- 2x^2 + x - 1

GS =

x + c_2 e^x - 2x^2 + c_1 e^{-x} - 1
```

Exercise Problem

Find the general solution of the given ODE in MATLAB using the method of variation of parameters

$$\frac{d^2y}{dx^2} - 3\frac{dy}{dx} + 2y \;=\; e^{3x}.$$

Practical No. 16

Aim

To solve a second-order homogeneous differential equation in MATLAB.

Problem

Find the general solution of the given ODE in MATLAB

$$\frac{d^2y}{dx^2} - 6\frac{dy}{dx} + 5y \;=\; 0.$$

Theory

The second-order homogeneous differential equation is solved by defining the complementary functions on the basis of the roots of the Auxiliary equation.

Algorithm

Step 1: Define the variables using syms command enter the coefficients of the second order differential equation,
Step 2: Solve for the roots of the Auxiliary equation using the solve command,
Step 3: On the basis of roots of the Auxiliary equation, define the general solution,
Step 4: Use the inbuilt disp function to display the output on the command window.

Program

```
syms r c1 c2 n
F=input('Input the coefficients [a,b,c]: ');

% Coefficients of the 2nd Order Difference Equation
a=F(1); b=F(2); c=F(3);

% Auxiliary Equation Formed
eq=a*(r^2)+b*(r)+c;
S=solve(eq);

% Roots of the Auxiliary Equation Formed
r1=S(1); r2=S(2);

% Determinant of the Auxiliary Equation Formed
D=b^2-4*a*c;
if D>0
    y1=exp(r1*n);
    y2=exp(r2*n);
    yn=c1*y1+c2*y2;
elseif D==0
    y1=exp(r1*n);
    y2=n*exp(r2*n);
    yn=c1*y1+c2*y2;
else
    alpha=real(r1);
    beta=imag(r2);
    y1=exp(alpha*n)*cos(beta*n);
    y2=exp(alpha*n)*sin(beta*n);
    yn=c1*y1+c2*y2;
end

disp('The solution of the difference equation is yn=')
disp(yn)
```

Output

```
Input the coefficients [a,b,c]: [1,-6,5]
The solution of the difference equation is yn=
c1*exp(n) + c2*exp(5*n)
```

Exercise Problem

Find the general solution of the given ODE in MATLAB

$$\frac{d^2y}{dx^2} - 8\frac{dy}{dx} + 16y \ = \ 0.$$

Practical No. 17

Aim

Plot the Bessel function in MATLAB.

Problem

Plot the Bessel function of the first kind in MATLAB.

Theory

The Bessel functions of the first kind $J_n(x)$ are defined as the solutions to Bessel differential equation $x^2\frac{d^2y}{dx^2} + x\frac{dy}{dx} + (x^2 - n^2)y = 0$ which are nonsingular at the origin.

Algorithm

Step 1: Define the grids on the x-axis,
Step 2: Use the inbuilt Bessel command to compute the Bessel solution,
Step 3: Use the inbuilt plot function to plot the Bessel solution,
Step 4: Click on the Run button on the top panel.

Program

```
% Bessel Function of the First Kind
x = 0:0.1:10;
J0 = besselj(0, x);
plot(x, J0);
title('Bessel Function of the First Kind J_0(x)');
xlabel('x');
ylabel('J_0(x)');
grid on;
```

Output

Exercise Problem

Plot the Bessel solution $J_n(x)$ for $n = 2, 3, 4, 5$.

Practical No. 18

Aim

Plot the Legendre polynomial in MATLAB.

Problem

Plot the Legendre polynomial of the first kind in MATLAB.

Theory

The Legendre polynomials are solutions to the Legendre differential equation.

Algorithm

Step 1: Define the grids on the x-axis,
Step 2: Use the inbuilt Legendre command to compute the Legendre polynomial,
Step 3: Use the inbuilt plot function to plot the Legendre polynomial,
Step 4: Click on the Run button on the top panel.

Program

```
% Legendre Polynomials
x = linspace(-1, 1, 100);
P5 = legendreP(5, x);
plot(x, P5);
title('Legendre Polynomial P_5(x)');
xlabel('x');
ylabel('P_5(x)');
grid on;
```

Output

Exercise Problem

Plot the Legendre function of the second kind in MATLAB.

Practical No. 19

Aim

Solve a system of ordinary simultaneous differential equations in MATLAB.

Problem

Solve the given system of simultaneous ordinary differential equations in MATLAB.

$$\frac{dx}{dt} - y = -x, \qquad \frac{dy}{dt} - x = -y.$$

Theory

Linear differential equations having two or more dependent variables with a single independent variable are called simultaneous differential equations.

Algorithm

Step 1: Define two ODEs using the inbuilt systems and diff function,
Step 2: Use the inbuilt dsolve() to compute the solution of the simultaneous ODEs,
Step 3: Extract the different solutions from the common solution using command,
Step 4: Use the inbuilt disp function to display the output on the command window,
Step 5: Click on the Run button on the top panel.

Program

```
syms x(t) y(t)
ode1 = diff(x) == y - x;
ode2 = diff(y) == x - y;
odes = [ode1; ode2];
S = dsolve(odes);

xsol(t) = S.x;
```

```
ysol(t) = S.y;

disp('The solution of the system of differential equation is ')
disp('x(t)=');
disp(xsol(t));
disp('y(t)=');
disp(ysol(t));
```

Output

```
The solution of the system of differential equations is:

x(t) = C_{1} - C_{2} e^{- 2 t}
y(t) = C_{1} + C_{2} e^{- 2 t}
```

Exercise Problem

Solve the given system of simultaneous ordinary differential equations in MATLAB

$$\frac{dx}{dt} + y \;=\; e^t, \quad \frac{dy}{dt} - x \;=\; e^{-t}.$$

Practical No. 20

Aim

To find the series solution of a second-order differential equation (ODE) in MATLAB.

Problem

Find the series solution of the second-order differential equation in MATLAB

$$(x^2-1)^2\frac{d^2y}{dx^2} + (x+1)\frac{dy}{dx} - y \;=\; 0.$$

Theory

For finding the series solution to a differential equation, we assume a solution as a power series in the form $y(x) = \sum_{n=0}^{\infty} a_n(x-x_0)^n$ around the ordinary point $x = x_0$ and determine the coefficients a_n.

Algorithm

Step 1: Define the ODE using the inbuilt systems and diff function,
Step 2: Use the inbuilt dsolve() along with the Expansion point command to find the series solution,
Step 3: Use the inbuilt disp function to display the output on the command window,
Step 4: Click on the Run button on the top panel.

Program

```
syms y(x)
eqn = (x^2 - 1)^2 * diff(y,2) + (x + 1) * diff(y) - y == 0;
S1 = dsolve(eqn, 'ExpansionPoint', -1);

disp('The series solution of the differential equation is ')
disp(S1);
```

Output

```
The series solution of the differential equation is

(x + 1)*(1/4) - (5*(x + 1)^3*(3/4))/4 + (5*(x + 1)^5*(7/48))/48
+ (5*(x + 1)^7*(11/49))/336 - (115*(x + 1)^9*(15/49))/33792
+ (169*(x + 1)^11*(19/49))/118432 ...
```

Exercise Problem

Find the other series solution around the point ∞ of the second-order differential equation in MATLAB

$$(x^2-1)^2\frac{d^2y}{dx^2}+(x+1)\frac{dy}{dx}-y \;=\; 0.$$

Practical No. 21

Aim

To define and find the product of three vectors in MATLAB.

Problem

Find the product of the following vectors in MATLAB

$$\begin{bmatrix}2\\4\\8\end{bmatrix}, \begin{bmatrix}3\\5\\6\end{bmatrix} \quad and \quad \begin{bmatrix}-1\\4\\0\end{bmatrix}.$$

Theory

Vector multiplication means each entry of a vector is multiplied element-wise with the corresponding entry in the other vector.

Algorithm

Step 1: Define the vectors as an array,
Step 2: Use .* to do element-wise multiplication in MATLAB,
Step 3: Use the inbuilt disp function to display the output on the command window,
Step 4: Click on the Run button on the top panel.

Program

```
%Defining the vectors
x=[2 4 8]';
y=[3 5 6]';
z=[-1 4 0]';

% ' indicates transpose. If ' is not used then we will get a row vector

p = x .* y .* z;  %--product of three vectors

disp(``The product of vectors x, y and z is")
disp(p)
```

Output

```
The product of vectors x, y and z is

    -6
    80
     0
```

Exercise Problem

Find the product of the following vectors in MATLAB

$$\begin{bmatrix} 1 \\ -1 \\ 2 \end{bmatrix}, \begin{bmatrix} 2 \\ 0 \\ 4 \end{bmatrix}, \begin{bmatrix} -2 \\ 1 \\ 9 \end{bmatrix} \quad and \quad \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix}.$$

Practical No. 22

Aim

To ask the user for vectors and find their product in MATLA.

Problem

Write a MATLAB code to ask the user for three different vectors and find the product of these vectors.

Theory

Vector multiplication means each entry of a vector is multiplied element-wise with the corresponding entry in the other vector.

Algorithm

Step 1: Use different vectors,
Step 2: Use .* to do element-wise multiplication in MATLAB,
Step 3: Use the inbuilt disp function to display the output on the command window,
Step 4: Click on the Run button on the top panel.
Step 5: Define the vectors as input an array. Press enter after providing the value of each vector.

Program

```
%Asking users for vectors
x = input(‘‘Enter the vector x’’);
disp(x);

y = input(‘‘Enter the vector y’’);
disp(y);
```

```
z = input(‘‘Enter the vector z’’);
disp(z);

p = x .* y .* z;
disp(‘‘The product of vectors x, y and z is’’);
disp(p);
```

Output

```
Enter the vector x [9 8 1 -1]’
     9
     8
     1
    -1

Enter the vector y [2 9 -7 8]’
     2
     9
    -7
     8

Enter the vector z [8 -11 -1 1]’
     8
   -11
    -1
     1

The product of vectors x, y and z is
   144
  -792
     7
    -8
```

Exercise Problem

Write a MATLAB code to ask the user for five different vectors and then find the product of these vectors.

Practical No. 23

Aim

To ask the user for vectors and find the vector product of these vectors in MATLAB.

Problem

Write a MATLAB code to ask the user for three different vectors and then find the vector product of these vectors.

Theory

Vector product for any three vectors $a = (a_1, a_2, a_3),\ b = (b_1, b_2, b_3),\ c = (c_1, c_2, c_3)$ is given by $(a \times (b \times c)) \ = \ (a.c)b - (a.b)c$

Algorithm

Step 1: Use the inbuilt input command to ask the user for three different vectors x,y and z,
Step 2: Use the inbuilt cross command to find the cross product of y and z,
Step 3: Use the cross command again to find the cross product between x and the new vector obtained from Step 2,
Step 4: Use the inbuilt disp function to display the output on the command window,
Step 5: Click on the Run button on the top panel.
Step 6: Define the vectors as input an array.

Program

```
x = input(‘‘Enter the vector x’’);
disp(x);

y = input(‘‘Enter the vector y’’);
disp(y);

z = input(‘‘Enter the vector z’’);
disp(z);

p = cross(y, z);
q = cross(x, p);

disp(‘‘The vector product of x, y and z is’’);
disp(q);
```

Output

```
Enter the vector x [3 -1 2]'
     3
    -1
     2

Enter the vector y [2 1 -1]'
     2
     1
    -1

Enter the vector z [1 -2 2]'
     1
    -2
     2

The vector product of x, y and z is
    15
    15
   -15
```

Exercise Problem

Write a MATLAB code for a scalar a and vectors x and y and then compute the value of $a.(x \times y)$.

Chapter 5

Group Theory

Practical No. 1

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program which takes input of two integers a and d, where d is a positive integer. Apply division algorithm and calculate quotient q and remainder r when a is divided by d. Print the result. Run the program for $(a, d) = (-41, 6), (253, 17), (24139, 12)$.

Theory

According to division algorithm if we divide an integer a by a positive integer d, then we get unique integers q and $0 \leq r < d$ such that

$$a = qd + r.$$

Algorithm

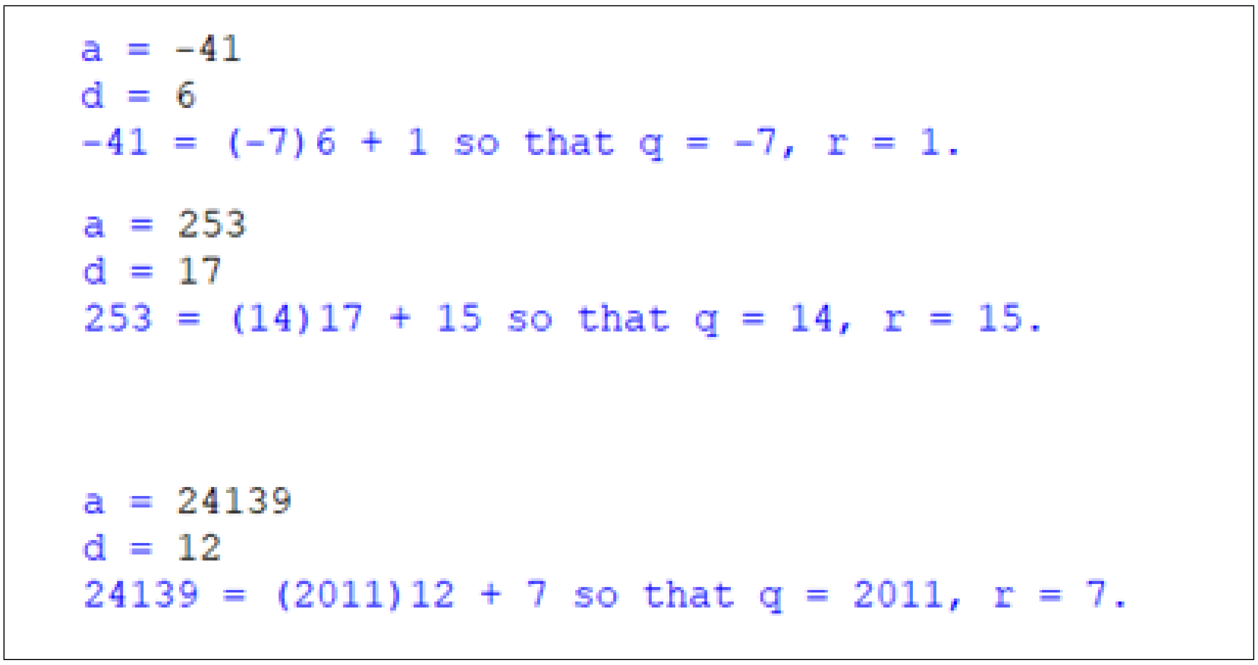
Step1: Prompt user to give input of integer a;
Step2: Prompt user to give input of integer d;
Step3: Verify if d is a positive integer. If d is not a positive integer, inform user and exit program;
Step4: Define and calculate q;
Step5: Define and calculate r;
Step6: Print q and r.

Program

```
#Step 1
a = int(input('a = '))
#Step 2
d = int(input('d = '))
#Step 3
if d < 1:
    raise ValueError('d must be a positive integer')
#Step 4
q = a//d
#Step 5
r = a%d
#Step 6
print(f'{a} = ({q}){d} + {r} so that q = {q}, r = {r}.')
```

Output

The output of the program for various inputs is as following:



```
a = -41
d = 6
-41 = (-7)6 + 1 so that q = -7, r = 1.

a = 253
d = 17
253 = (14)17 + 15 so that q = 14, r = 15.



a = 24139
d = 12
24139 = (2011)12 + 7 so that q = 2011, r = 7.
```

Exercise Problem

Write a program which takes an integer n as input from the user and prints if n is even or odd. Run the program for $n = -51, 0, 25, 203541$.

Practical No. 2

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program which takes input of two integers a and b from user. Then program prints the maximum of a and b. Run the program $(a, b) = (5, -8), (12, 12), (11, 100)$.

Theory

Let a and b be two integers.Then maximum of a and b,$max(a, b)$ is given by

$$\max(a, b) = f(x) = \begin{cases} a, & a > b \\ b, & \text{otherwise.} \end{cases}$$

Algorithm

```
**Step1:** Prompt user to give input of integers $a$ and $b$;
**Step2:** Define an integer variable m for maximum;
**Step3:** If $a > b$ then set $m = a$;
**Step4:** Otherwise set $m = b$;
**Step5:** Print value of maximum.
```

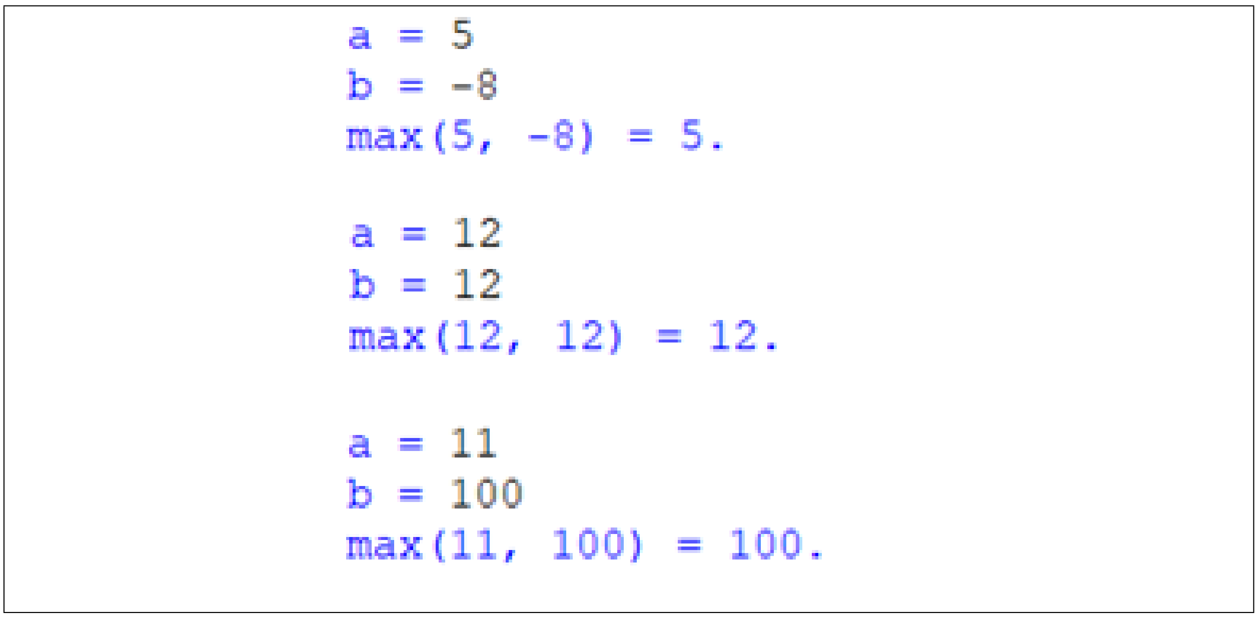
Program

The following PYTHON script implements the algorithm.

```
#Step 1
a = int(input('a = '))
b = int(input('b = '))
#Step 2
m = 0
#Step 3
if a > b:
    m = a
#Step 4
else:
    m = b
#Step 5
print(f'max({a}, {b}) = {m}.')
```

Output

The output of the program for various inputs is as following:



```
a = 5
b = -8
max(5, -8) = 5.

a = 12
b = 12
max(12, 12) = 12.

a = 11
b = 100
max(11, 100) = 100.
```

Exercise Problem

Write a program which takes input of two integers a and b from user. Then program prints the minimum of a and b. Run the program for $(a, b) = (5, -8), (12, 12), (11, 100)$.

Practical No. 3

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program which takes input of two positive integers a and b from user. The program calculates and prints the GCD of a and b. Run the program for $(a, b) = (21, 18), (30, 14), (105, 80)$.

Theory

The GCD of two positive integers a and b is defined as the least of all the positive common divisors of a and b. It can be found using Euclidean algorithm.

Algorithm

Step1: Prompt user to give input of integers a and b;
Step2: Check if a and b are positive integers. If not, show error and exit;
Step3: Define and calculate $M = \max(a, b)$;
Step4: Define and calculate $m = \min(a, b)$;
Step5: Run a loop while M mod m is non-zero;
Step6: At each iteration, set M to m and m to M mod m;
Step7: Define and set *gcd* to m;
Step8: Print the value of GCD(a, b).

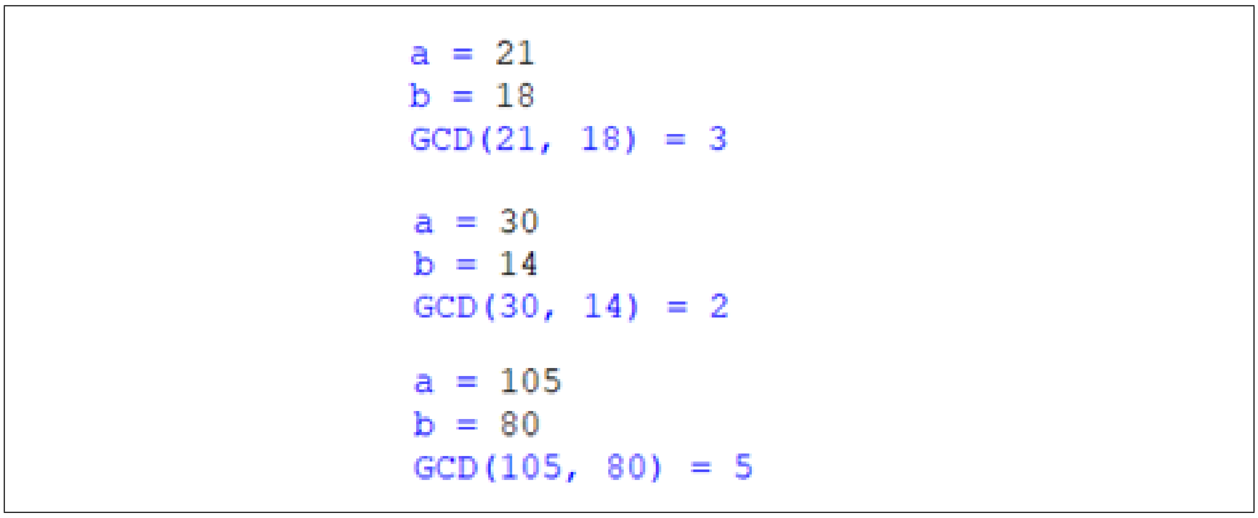
Program

The following PYTHON script implements the algorithm.

```
#Step 1
a = int(input('a = '))
b = int(input('b = '))
#Step 2
if a < 1 or b < 1:
    raise ValueError('a and b must be a positive integers.')
#Step 3
m = min(a, b)
#Step 4
M = max(a, b)
#Step 5
while M%m != 0:
    #Step 6
    r = M%m
    M = m
    m = r
#Step 7
gcd = m
#Step 8
print(f'GCD({a}, {b}) = {gcd}')
```

Output

The output of the program for various inputs is as following:



```
a = 21
b = 18
GCD(21, 18) = 3

a = 30
b = 14
GCD(30, 14) = 2

a = 105
b = 80
GCD(105, 80) = 5
```

Exercise Problem

Write a program which takes input of two positive integers a and b from user. The program calculates and prints the LCM of a and b. Run the program for $(a, b) = (12, 18), (10, 48), (7, 56)$.

Practical No. 4

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program which takes input of a positive integer $a \geq 2$ and prints if it is a prime or composite number. Run the program for $a = 2, 8, 101, 67777$.

Theory

An integer is called a prime if its only positive divisors are 1 and itself. We can search the divisors of given integer a between 1 and a. If we find a divisor between 1 and a, then it is composite and otherwise it is prime.

Algorithm

Step1: Prompt user to give input of integers a;
Step2: Check if a is a positive integer greater than 1. If not, show error and exit;
Step3: Define a Boolean variable composite and set it to false;
Step4: Run a loop from $d = 2$ to $d = a - 1$.It should not run any iteration if $a = 2$;
Step5: Check if d divides a.If $d \mid a$, then set *composite* to true and exit the loop;
Step6: Print the result.

Program

The following PYTHON script implements the algorithm.

```
#Step 1
a = int(input('a = '))
#Step 2
if a < 2:
    raise ValueError('a must be an integer greater than 1.')
#Step 3
composite = False
#Step 4
for d in range(2, a - 1):
    #Step 5
    if a%d == 0:
        composite = True
#Step 6
if composite:
    print(f'{a} is a composite number.')
else:
    print(f'{a} is a prime number.')
```

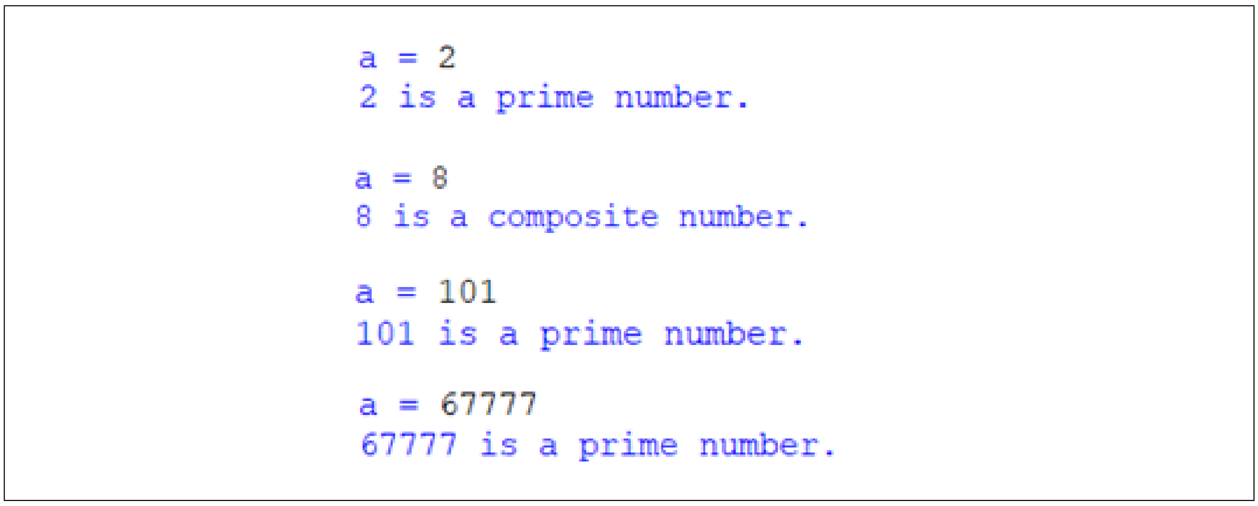
Output

The output of the program for various inputs is as following:

Exercise Problem

Write a program which takes input of a positive integer a and prints the number of prime integers less than or equal to a. Run the program for $a = 10, 100, 1000, 10000$.

Practical No. 5



```
a = 2
2 is a prime number.

a = 8
8 is a composite number.

a = 101
101 is a prime number.

a = 67777
67777 is a prime number.
```

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program which takes input of a positive integer a and prints a list of its positive divisors. Run the program for $a = 6, 15, 28, 80$.

Theory

A positive integer d is said to be divisor of an integer a if the remainder upon diving a by d vanishes.

Algorithm

Step1: Prompt user to give input of integer a;
Step2: Check if a is a positive integer. If not, show error and exit;
Step3: Define a list *divisors*;
Step4: Run a loop from $d = 1$ to $d = a$;
Step5: Check if d divides a.If $d \mid a$,then set add d to the list divisors;
Step6: Print the list divisors.

Program

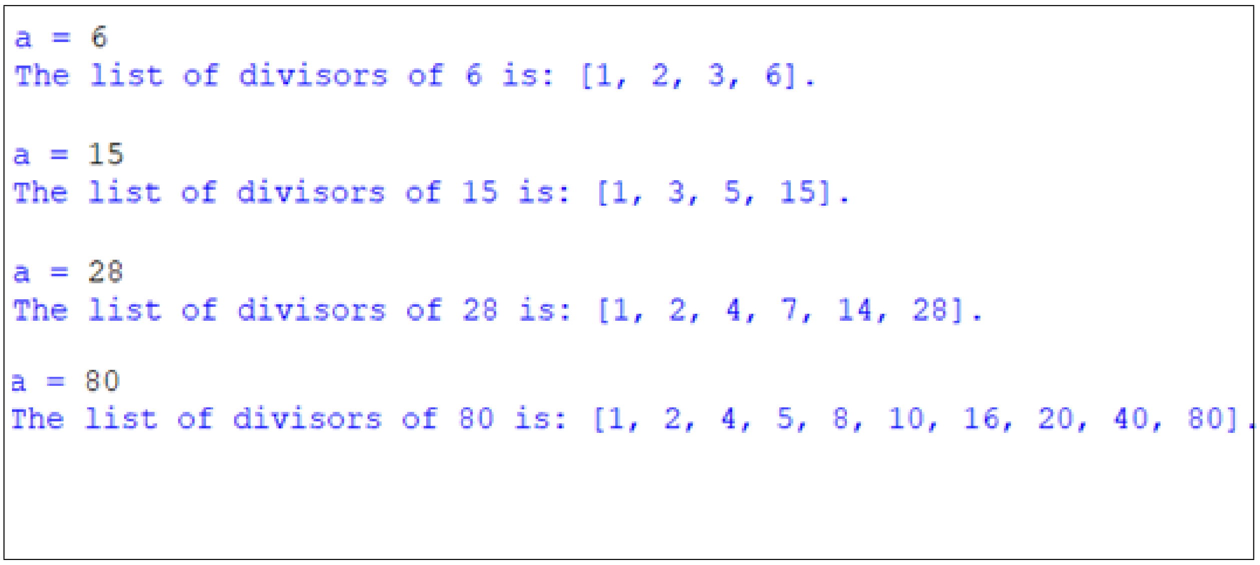
The following PYTHON script implements the algorithm.

```
#Step 1
```

```
a = int(input('a = '))
#Step 2
if a < 1:
    raise ValueError('a must be a positive integer.')
#Step 3
divisors = []
#Step 4
for d in range(1, a + 1):
    #Step 5
    if a%d == 0:
        divisors.append(d)
#Step 6
print(f'The list of divisors of {a} is: {divisors}.')
```

Output

The output of the program for various inputs is as following:



```
a = 6
The list of divisors of 6 is: [1, 2, 3, 6].

a = 15
The list of divisors of 15 is: [1, 3, 5, 15].

a = 28
The list of divisors of 28 is: [1, 2, 4, 7, 14, 28].

a = 80
The list of divisors of 80 is: [1, 2, 4, 5, 8, 10, 16, 20, 40, 80].
```

Exercise Problem

Write a program which takes input of a positive integer a and prints if it is a perfect number.A positive integer a is said to be perfect if it is equal to the sum of all its divisors less than a.Run the program for $a = 6, 15, 28, 80$.

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program which takes input of a positive integer n and prints the number of positive integers $d \leq n$ which are co-prime to n. Run the program for $n = 1, 10, 12, 60$.

Theory

Two positive integers a and b are said to be co-prime if $GCD(a, b) = 1$. The Eulers totient function $\phi(n)$ for positive integer n is defined as the number of co-prime positive integers less than or equal to n.

Algorithm

```
Step1: Prompt user to give input of integer $n$;
Step2: Check if $n$ is a positive integer.If not, show error and exit;
Step3: Define variable $phi\_n$ for $\phi(n)$ and set it to zero;
Step4: Run a loop from $d = 1$ to $d = n$;
Step5: At each iteration, if $GCD(d, n) = 1$,increase $phi\_n$ by 1;
Step6: Print the value of $\phi(n)$.
```

Program

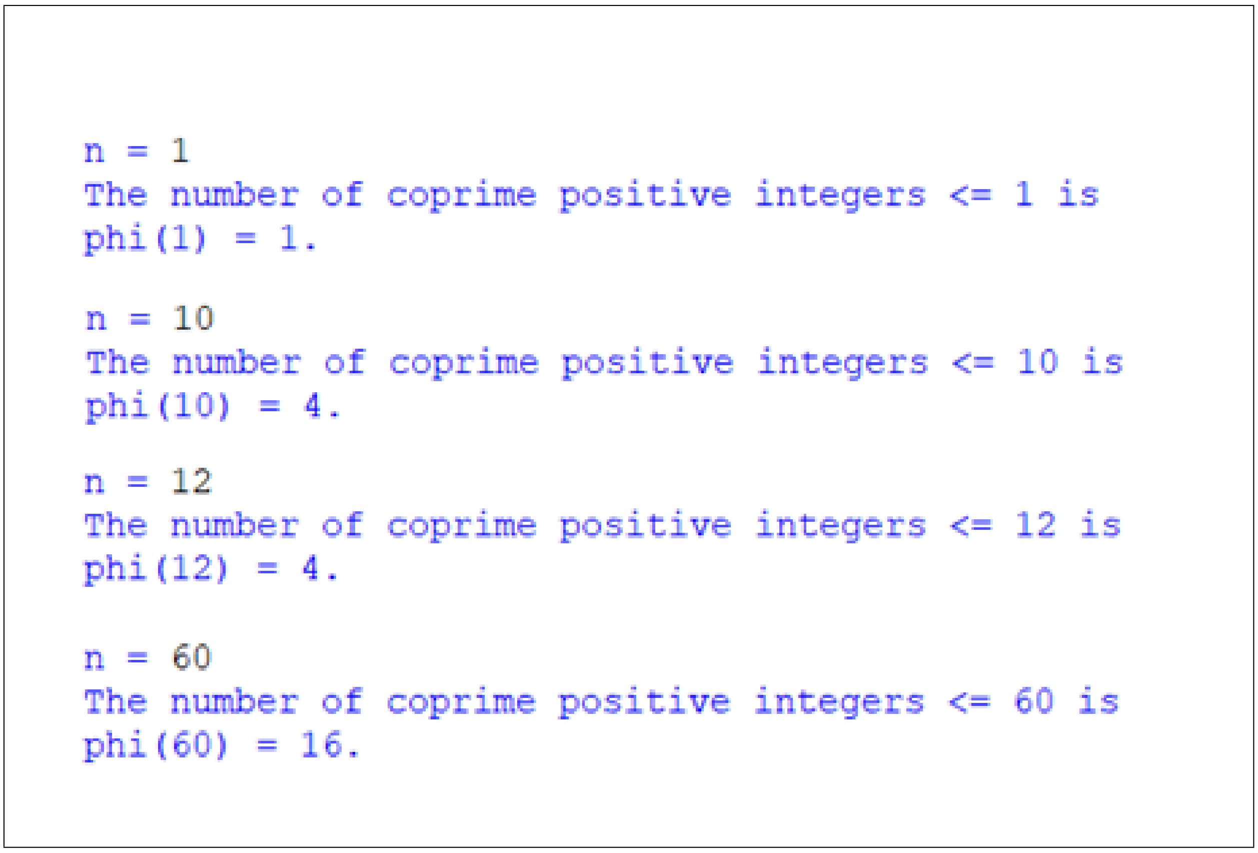
The following PYTHON script implements the algorithm.

```python
#Function to calculate GCD of two positive integers
def gcd(a, b):
    m = min(a, b)
    M = max(a, b)
    while M%m != 0:
        m, M = M%m, m
    return m

#Step 1
n = int(input('n = '))
#Step 2
if n < 1:
    raise ValueError('n must be a positive integer.')
#Step 3
```

```
phi_n = 0
#Step 4
for d in range(1, n + 1):
    #Step 5
    if gcd(d, n) == 1:
        phi_n += 1
#Step 6
print(f'The number of coprime positive integers <= {n} is\nphi({n}) = {phi_n}.')
```

Output

The output of the program for various inputs is as following:



```
n = 1
The number of coprime positive integers <= 1 is
phi(1) = 1.

n = 10
The number of coprime positive integers <= 10 is
phi(10) = 4.

n = 12
The number of coprime positive integers <= 12 is
phi(12) = 4.

n = 60
The number of coprime positive integers <= 60 is
phi(60) = 16.
```

Exercise Problem

Write a program which takes input of two positive integers a and b and prints if they are co-prime or not. Run the program for $(a, b) = (1, 1), (15, 24), (90, 143)$.

Practical No. 7

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program which takes input of a positive integer $n > 1$ and prints all its positive prime factors. Run the program for $a = 20, 104, 2985, 24781$.

Theory

An integer greater than 1 is said to be prime if its only divisors are 1 and itself.An integer is said to be factor of another integer if earlier is a divisor of the later.

Algorithm

```
Step1: Prompt user to give input of integer $n$;
Step2: Check if $n$ is a positive integer greater than 1. If not,show error and exit;
Step3: Define a list $PrimeFactors$ to save prime factors of $n$;
Step4: Define $q = n$ and define $p = 2$;
Step5: Run a loop till $q$ is not 1;
Step6: At each iteration,if $p \mid q$,then add $p$ to list $primefactors$;
Step7: Run another loop till $p \mid q$;
Step8: At each iteration of inner loop, set $q$ to $q/p$;
Step9: Exit inner loop and set $p$ to next prime;
Step10: Exit outer loop;
Step11: Print the list of prime factors.
```

Program

The following PYTHON script implements the algorithm.

```
#Function to check the primality of an integer greater than 1
def isprime(n):
    for d in range(2, int(n**0.5) + 1):
        if n % d == 0:
            return False
    return True

#Function to calculate next prime integer after n
def nextprime(n):
    while True:
        n += 1
        if isprime(n):
            return n

#Step 1
n = int(input('n = '))
#Step 2
if n < 2:
    raise ValueError('n must be a positive integer greater than 1.')
#Step 3
primeFactors = []
#Step 4
q = n
p = 2
#Step 5
while q != 1:
    #Step 6
    if q%p == 0:
        primeFactors.append(p)
    #Step 7
    while q%p == 0:
        #Step 8
        q = q/p
    #Step 9
    p = nextprime(p)
#Step 11
print(f'Prime factors of {n} are = {primeFactors}.')
```

Output

The output of the program for various inputs is as following:

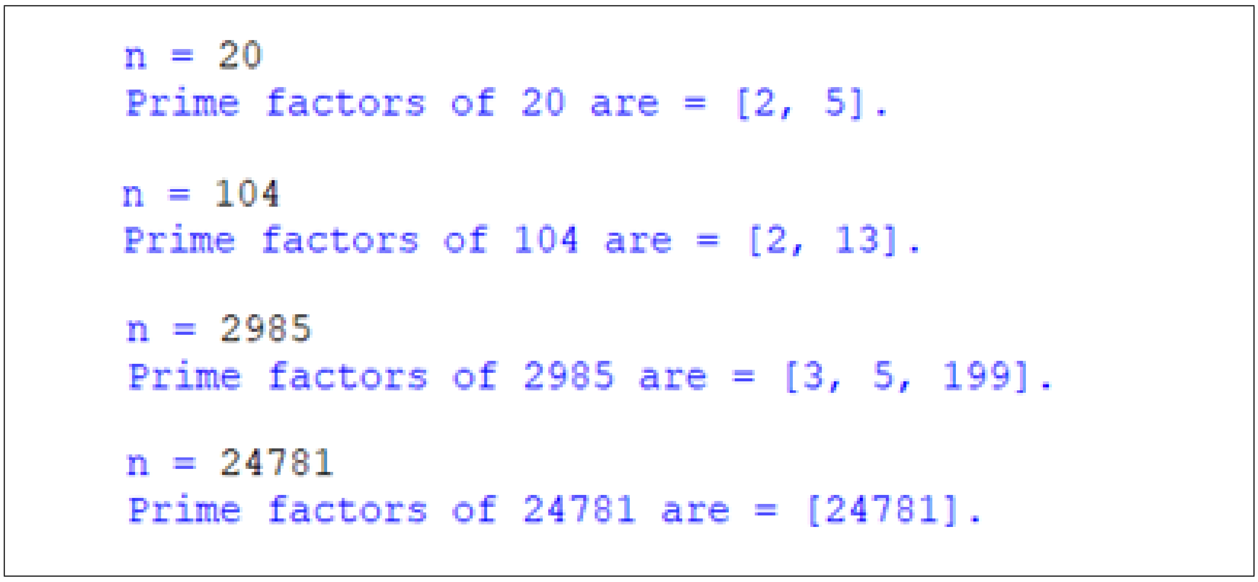
Exercise Problem

Write a program which takes input of a positive integer $n > 1$ and prints its prime factorization according to the Fundamental Theorem of Arithmetic.Run the program for $a = 30, 52, 100, 686, 2024$.

Practical No. 8

Aim

To solve the given problem with the help of PYTHON.



```
n = 20
Prime factors of 20 are = [2, 5].

n = 104
Prime factors of 104 are = [2, 13].

n = 2985
Prime factors of 2985 are = [3, 5, 199].

n = 24781
Prime factors of 24781 are = [24781].
```

Problem

Write a program which takes input of a positive integer n. Then, it prints order and all the elements of the group Z_n. Run the program for $n = 1, 2, 10, 24$.

Theory

$Z_n, n > 0$ is the group of integers addition modulo n:

$$Z_n = \{k : k\epsilon z, 0 \leq k < n\}$$

.

Algorithm

Step1: Prompt user to give input of integer n;
Step2: Check if n is a positive integer. If not, show error and exit;
Step3: Define a list zn to save elements Z_n;
Step4: Add elements of Z_n to zn;
Step5: Print order of Z_n and elements of Z_n.

Program

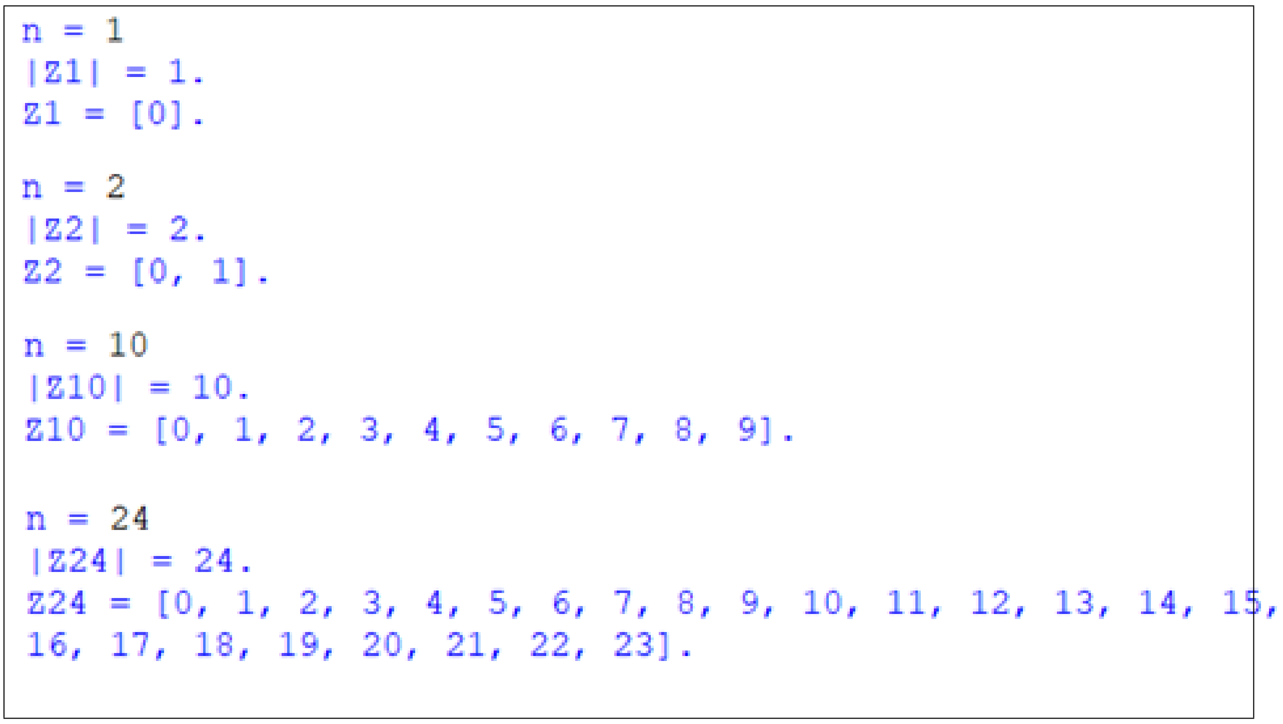
The following PYTHON script implements the algorithm.

```
#Step 1
n = int(input('n = '))
```

```
#Step 2
if n < 1:
    raise ValueError('n must be a positive integer.')
#Step 3 & 4
zn = list(range(0, n))
#Step 5
print(f'|Z{n}| = {len(zn)}.')
print(f'Z{n} = {zn}.')
```

Output

The output of the program for various inputs is as following:



```
n = 1
|Z1| = 1.
Z1 = [0].

n = 2
|Z2| = 2.
Z2 = [0, 1].

n = 10
|Z10| = 10.
Z10 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9].

n = 24
|Z24| = 24.
Z24 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
16, 17, 18, 19, 20, 21, 22, 23].
```

Exercise Problem

Write a program which takes input of a positive integer n. Then, it prints the inverse of each of the elements of the group Z_n. Run the program for $n = 1, 2, 10, 24$.

Practical No. 9

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program which takes input of an integer $n > 1$. Then it prints the order and all the elements of $U(n)$. Run the program for $n = 12, 15, 30, 36$.

Theory

For natural number $n > 1$,$U(n)$ is the group of positive integers $k < n$, such that $gcd(k, n) = 1$. The group $U(n)$ is also called the group of units in Z_n.

Algorithm

```
Step1: Prompt user to give input of integer $n$;
Step2: Check if $n$ is a positive integer. If not, show error and exit;
Step3: Define a list $un$ to save elements $U(n)$;
Step4: Run a loop from $k = 1$ to $k = n - 1$;
Step5: In each iteration, if $gcd(k, n) = 1$, then include $k$ in the list $un$;
Step6: End loop;
Step7: Print the order and elements of $U(n)$.
```

Program

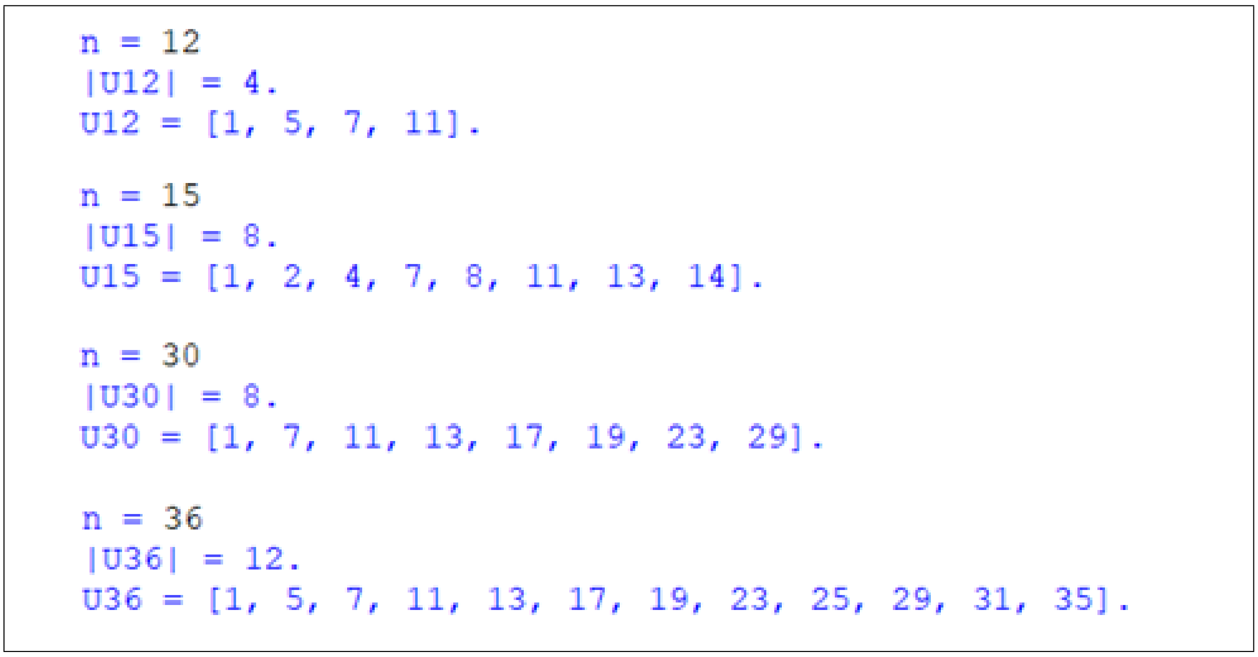
The following PYTHON script implements the algorithm.

```
#Function to calculate GCD of two positive integers
def gcd(a, b):
    m = min(a, b)
    M = max(a, b)
    while M%m != 0:
        m, M = M%m, m
    return m

#Step 1
n = int(input('n = '))
#Step 2
if n <= 1:
    raise ValueError('n must be a positive integer greater than 1.')
#Step 3
```

```
un = []
#Step 4
for k in range(1, n):
    #Step 5
    if gcd(k, n) == 1:
        un.append(k)
#Step 7
print(f'|U{n}| = {len(un)}.')
print(f'U{n} = {un}.')
```

Output

The output of the program for various inputs is as following:



```
n = 12
|U12| = 4.
U12 = [1, 5, 7, 11].

n = 15
|U15| = 8.
U15 = [1, 2, 4, 7, 8, 11, 13, 14].

n = 30
|U30| = 8.
U30 = [1, 7, 11, 13, 17, 19, 23, 29].

n = 36
|U36| = 12.
U36 = [1, 5, 7, 11, 13, 17, 19, 23, 25, 29, 31, 35].
```

Exercise Problem

Write a program which takes input of an integer $n > 1$. Then, it prints the inverse of each of the elements of $U(n)$. Run the program for $n = 12, 15, 30, 36$.

Practical No. 10

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program which takes input of an integer $n > 1$. Then it prints the order of each of the elements of $U(n)$. Run the program for $n = 10, 12, 15, 30$.

Theory

The inverse for $a \in U(n)$ is $b \in U(n)$ such that $ab \mod n = 1$.

Algorithm

```
Step1: Prompt user to give input of integer $n$;
Step2: Check if $n$ is a positive integer. If not, show error and exit;
Step3: Define a list $un$ and save elements $U(n)$ in it;
Step4: Define a list $orders$ to save element and order pair;
Step5: Run a loop for each $k$ in $U(n)$;
Step6: Define $u = 1$ and $order = 0$;
Step7: Run an inner loop;
Step8: At each iteration, set $u$ to $uk \mod n$ and $order$ to $order + 1$;
Step9: If $u = 1$, exit loop;
Step10: Save element and its order as a pair to the list orders and Exit outer loop;
Step11: Print the order of elements of $U(n)$.
```

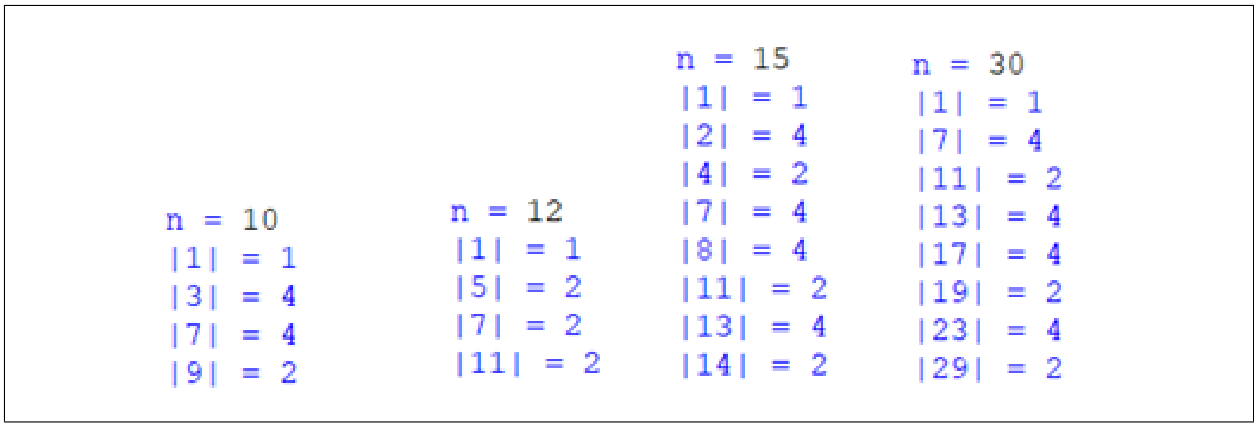
Program

The following PYTHON script implements the algorithm.

```
#Function to calculate GCD of two positive integers
def gcd(a, b):
    m = min(a, b)
    M = max(a, b)
    while M%m != 0:
        m, M = M%m, m
    return m
#Step 1
n = int(input('n = '))
#Step 2
if n <= 1:
    raise ValueError('n must be a positive integer greater than 1.')
#Step 3
un = []
for k in range(1, n):
    if gcd(k, n) == 1:
        un.append(k)
#Step 4
orders = []
#Step 5
for k in un:
#Step 6
    u = 1
    order = 0
    #Step 7
    while True:
        #Step 8
        u = u*k%n
        order += 1
        #Step 9
        if u == 1:
            break
    #Step 10
    orders.append((k, order))
#Step 11
for element, order in orders:
    print(f'|{element}| = {order}')
```

Output

The output of the program for various inputs is as following:



```
n = 10      n = 12      n = 15      n = 30
|1| = 1     |1| = 1     |1| = 1     |1| = 1
|3| = 4     |5| = 2     |2| = 4     |7| = 4
|7| = 4     |7| = 2     |4| = 2     |11| = 2
|9| = 2     |11| = 2    |7| = 4     |13| = 4
                        |8| = 4     |17| = 4
                        |11| = 2    |19| = 2
                        |13| = 4    |23| = 4
                        |14| = 2    |29| = 2
```

Exercise Problem

Write a program which takes input of a positive integer n. Then it prints the order of each of the elements of Z_n. Run the program for $n = 2, 5, 12, 18$.

Practical No. 11

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program which takes input of a 22 matrix M with integer entries and prints the determinant of M.Run the program for $M = \begin{bmatrix} 1 & -5 \\ 3 & 8 \end{bmatrix}, \begin{bmatrix} -3 & 2 \\ 6 & -4 \end{bmatrix}$.

Theory

The determinant $\mid M \mid$ of a 2×2 matrix $M = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ is given by

$$\mid M \mid = ad - bc$$

.

Algorithm

Step1: Step 1: Prompt user to give input of matrix $M = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$;
Step2: Verify if M is a 2×2 matrix having integer entries. If verification fails, show error and exit;
Step3: Define $det = \mid M \mid$ and set it to $ad - bc$;
Step4: Print the value of $\mid M \mid$.

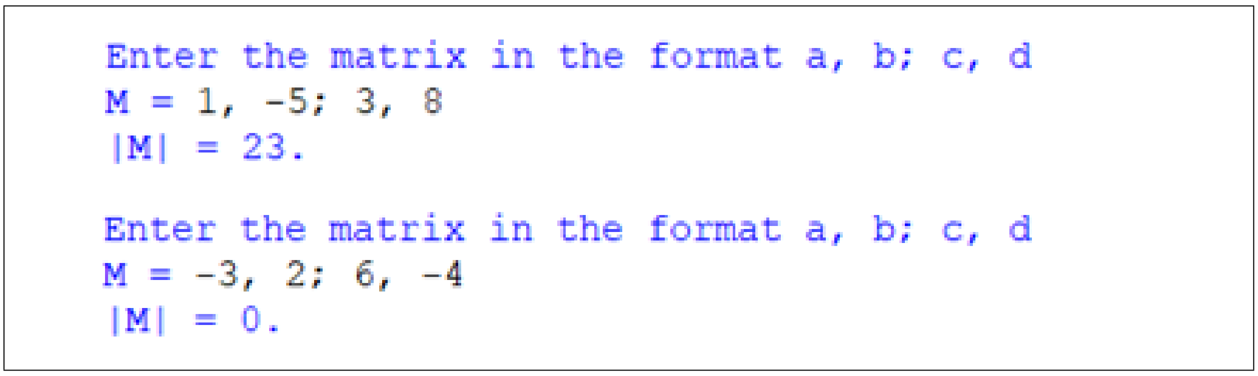
Program

The following PYTHON script implements the algorithm.

```
#Package for numerical computations
import numpy as np
#Step 1
M = np.matrix(input('Enter the matrix in the format a, b; c, d\nM = '), int)
#Step 2
rows, columns = M.shape
if rows != 2 or columns != 2:
    raise IndexError('M is not a 2x2 matrix.')
#Step 3
det = M[0, 0]*M[1, 1] - M[0, 1]*M[1, 0]
#Step 4
print(f'|M| = {det}.')
```

Output

The output of the program for various inputs is as following:



```
Enter the matrix in the format a, b; c, d
M = 1, -5; 3, 8
|M| = 23.

Enter the matrix in the format a, b; c, d
M = -3, 2; 6, -4
|M| = 0.
```

Exercise Problem

Write a program which takes input of a 2×2 matrix M with integer entries and a positive integer n. Then,it prints the matrix M^n. Run the program for $M = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 1 \\ 2 & 5 \end{bmatrix}$ with $n = 10$.

Practical No. 12

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program that takes input of a nonempty set of integers $S$and a positive integer r. It, then, finds all the possible permutations with repetitions of size r of elements of the given set. Finally, it prints the number of permutations and all the permutations to the user. Run the program for $S = \{0, 1, 2\}$ and $r = 4$.

Theory

Given a set of objects S with n elements, a permutation of size $r$of objects of S is an arrangement of objects in a definite order. If repetitions are allowed, then

no. of possible permutations $= r^n$.

Whereas, if $n \geq r$ and repetitions are not allowed, then

no. of possible permutations $= \binom{n}{k} = \frac{n!}{(n-r)!}$.

Algorithm

Step 1: Prompt user to give input of set S;
Step 2: Verify that S is a nonempty set of integers free from duplicates;
Step 3: If verification fails, show an error and exit;
Step 4: Prompt user to give input of r;
Step 5: Verify if r is a positive integer. If verification fails, show an error and exit;
Step 6: Define the list of permutations and add an empty permutation to it;
Step 7: Run a loop for r iterations;
Step 8: At each iteration, define a temporary list of permutations;
Step 9: Run a loop for each permutation in the list of permutations;
Step 10: At each iteration, run a loop for each element in S;
Step 11: At each iteration, add the element of this iteration to the permutation of this iteration at the end and append it to the temporary list of permutations;
Step 12: Exit two inner loops;
Step 13: Set list of permutations to temporary list of permutations and exit loop;
Step 14: Print the total number of permutations and all the permutations.

Program

The following PYTHON script implements the algorithm.

```
#Step 1
S = input('Enter the set as 0, 1, 2, ...\nS = ')
S = S.strip().split(',')
#Step 2 & 3
if len(S) == 0:
    raise IndexError('S must be nonempty.')
```

```
S = list(map(int, S))
if len(S) != len(set(S)):
    raise ValueError('S has duplicate elements.')
#Step 4
r = int(input('r = '))
#Step 5
if r <= 0:
    raise ValueError('r should be a positive integer.')
#Step 6
permutations = [[]]
#Step 7
for _ in range(r):
    #Step 8
    temp_permutations = []
    #Step 9
    for perm in permutations:
        #Step 10
        for element in S:
            #Step 11
            temp_permutations.append(perm + [element])
    #Step 13
    permutations = temp_permutations
#Step 14
print(f'Total number of permutations = {len(permutations)}.')
print(f'The permutations of size {r} with repetitions of set S = {S} are
as following:\n{permutations}')
```

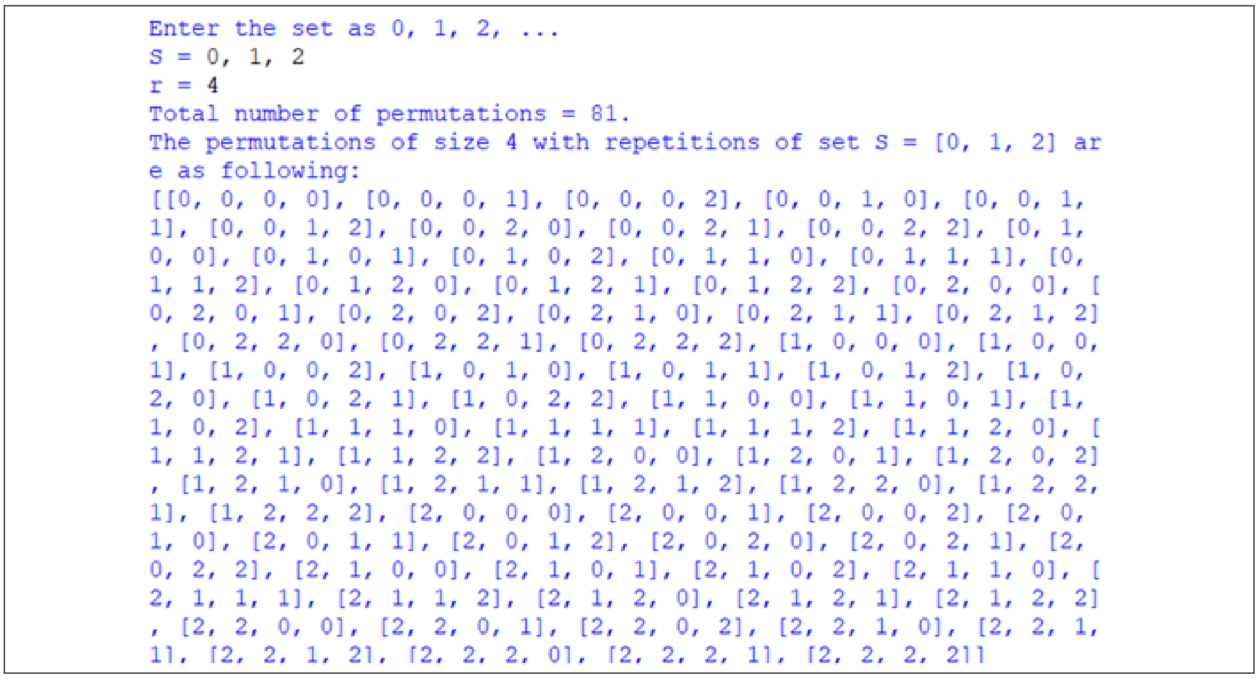
Output

The output of the program for various inputs is as following:

Exercise Problem

Write a program that takes input of a nonempty set of integers S and a positive integer r. It, then, finds all the possible permutations without repetitions of size r of elements of the given set. Finally, it prints the number of permutations and all the permutations to the user. Run the program for $S = \{0, 1, 2, 3, 4, 5, 6\}$ and $r = 4$.

Practical No. 13



```
Enter the set as 0, 1, 2, ...
S = 0, 1, 2
r = 4
Total number of permutations = 81.
The permutations of size 4 with repetitions of set S = [0, 1, 2] ar
e as following:
[[0, 0, 0, 0], [0, 0, 0, 1], [0, 0, 0, 2], [0, 0, 1, 0], [0, 0, 1,
1], [0, 0, 1, 2], [0, 0, 2, 0], [0, 0, 2, 1], [0, 0, 2, 2], [0, 1,
0, 0], [0, 1, 0, 1], [0, 1, 0, 2], [0, 1, 1, 0], [0, 1, 1, 1], [0,
1, 1, 2], [0, 1, 2, 0], [0, 1, 2, 1], [0, 1, 2, 2], [0, 2, 0, 0], [
0, 2, 0, 1], [0, 2, 0, 2], [0, 2, 1, 0], [0, 2, 1, 1], [0, 2, 1, 2]
, [0, 2, 2, 0], [0, 2, 2, 1], [0, 2, 2, 2], [1, 0, 0, 0], [1, 0, 0,
1], [1, 0, 0, 2], [1, 0, 1, 0], [1, 0, 1, 1], [1, 0, 1, 2], [1, 0,
2, 0], [1, 0, 2, 1], [1, 0, 2, 2], [1, 1, 0, 0], [1, 1, 0, 1], [1,
1, 0, 2], [1, 1, 1, 0], [1, 1, 1, 1], [1, 1, 1, 2], [1, 1, 2, 0], [
1, 1, 2, 1], [1, 1, 2, 2], [1, 2, 0, 0], [1, 2, 0, 1], [1, 2, 0, 2]
, [1, 2, 1, 0], [1, 2, 1, 1], [1, 2, 1, 2], [1, 2, 2, 0], [1, 2, 2,
1], [1, 2, 2, 2], [2, 0, 0, 0], [2, 0, 0, 1], [2, 0, 0, 2], [2, 0,
1, 0], [2, 0, 1, 1], [2, 0, 1, 2], [2, 0, 2, 0], [2, 0, 2, 1], [2,
0, 2, 2], [2, 1, 0, 0], [2, 1, 0, 1], [2, 1, 0, 2], [2, 1, 1, 0], [
2, 1, 1, 1], [2, 1, 1, 2], [2, 1, 2, 0], [2, 1, 2, 1], [2, 1, 2, 2]
, [2, 2, 0, 0], [2, 2, 0, 1], [2, 2, 0, 2], [2, 2, 1, 0], [2, 2, 1,
1], [2, 2, 1, 2], [2, 2, 2, 0], [2, 2, 2, 1], [2, 2, 2, 2]]
```

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program that takes the input of a prime number $p < 10$ from the user and prints the order and the elements of the group $GL(2, Z_p)$. Run the program for $p = 3$.

Theory

The group $GL(n, F)$ is the group of nn non-singular matrices with matrix multiplication as binary operation over field F. The addition and multiplication of elements of matrices are done as defined in the field F. So $GL(2, Z_p)$ is the group of 22 non-singular matrices over Z_p. The arithmetic operations are done modulo p for the elements in these matrices. A square matrix M is said to be non-singular if

$$det(M) \neq 0.$$

Since, in the field Z_p, we have $np = 0 \mod p$, so $M \epsilon GL(2, Z_p)$ is non-singular if

$$det(M) \neq np,$$

n being an integer.

Algorithm

Step 1: Prompt user to give input of p;
Step 2: Verify if p is a prime. If verification fails, show an error and exit;
Step 3: Define a list of permutations with repetitions of integers in Z_p of size 4;
Step 4: Define a list gl of elements of $GL(2, Z_p)$;
Step 5: Run a loop for each permutation in the list of permutations;
Step 6: At each iteration, reshape the permutation as a 22 matrix M;
Step 7: Calculate the determinant of M in modulo arithmetic;
Step 8: If the determinant of M is non-zero, then append M to the list gl;
Step 9: Exit loop;
Step 10: Print the order and elements of $GL(2, Z_p)$.

Program

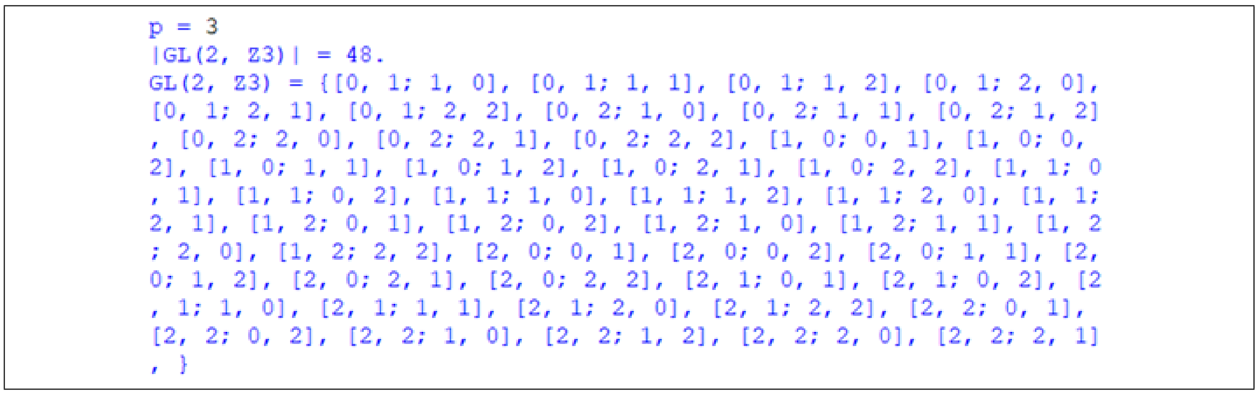
The following PYTHON script implements the algorithm.

```
import numpy as np

#Function to check primality of an integer greater than 1
def isprime(n):
    for d in range(2, int(n**0.5) + 1):
        if n % d == 0:
            return False
    return True

#Function to calculate all permutations with repetitions of given size
def perm(s, r):
    permutations = [[]]
for _ in range(r):
        temp_permutations = []
        for permutation in permutations:
            for element in s:
                temp_permutations.append(permutation + [element])
        permutations = temp_permutations
    return permutations

#Function to calculate determinant of 2x2 matrices
def det(m):
    return m[0, 0]*m[1, 1] - m[0, 1]*m[1, 0]

#Step 1
p = int(input('p = '))
#Step 2
```

```
if p <= 1 or not isprime(p):
    raise ValueError('p should be a prime number.')
#Step 3
permutations = perm(list(range(0, p)), 2*2)
#Step 4
gl = []
#Step 5
for permutation in permutations:
    #Step 6
    m = np.matrix(np.reshape(permutation, (2, 2)))
    #Step 7
    d = det(m)%p
    #Step 8
    if d!= 0:
        gl.append(m)
#Step 10
print(f'|GL(2, Z{p})| = {len(gl)}.')
print(f'GL(2, Z{p}) = {{', end='')
for m in gl:
    print(f'[{m[0, 0]}, {m[0, 1]}; {m[1, 0]}, {m[1, 1]}], ', end='')
print('}')
```

Output

The output of the program for the given input is as follows:



```
p = 3
|GL(2, Z3)| = 48.
GL(2, Z3) = {[0, 1; 1, 0], [0, 1; 1, 1], [0, 1; 1, 2], [0, 1; 2, 0],
[0, 1; 2, 1], [0, 1; 2, 2], [0, 2; 1, 0], [0, 2; 1, 1], [0, 2; 1, 2]
, [0, 2; 2, 0], [0, 2; 2, 1], [0, 2; 2, 2], [1, 0; 0, 1], [1, 0; 0,
2], [1, 0; 1, 1], [1, 0; 1, 2], [1, 0; 2, 1], [1, 0; 2, 2], [1, 1; 0
, 1], [1, 1; 0, 2], [1, 1; 1, 0], [1, 1; 1, 2], [1, 1; 2, 0], [1, 1;
2, 1], [1, 2; 0, 1], [1, 2; 0, 2], [1, 2; 1, 0], [1, 2; 1, 1], [1, 2
; 2, 0], [1, 2; 2, 2], [2, 0; 0, 1], [2, 0; 0, 2], [2, 0; 1, 1], [2,
0; 1, 2], [2, 0; 2, 1], [2, 0; 2, 2], [2, 1; 0, 1], [2, 1; 0, 2], [2
, 1; 1, 0], [2, 1; 1, 1], [2, 1; 2, 0], [2, 1; 2, 2], [2, 2; 0, 1],
[2, 2; 0, 2], [2, 2; 1, 0], [2, 2; 1, 2], [2, 2; 2, 0], [2, 2; 2, 1]
, }
```

Exercise Problem

Write a program that takes the input of a prime number $p < 10$ from the user and prints the order and the elements of the group $SL(2, Z_p)$. Run the program for $p = 3$.

Practical No. 14

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program that takes the input of a prime number $p < 10$ from the user and prints the order of each element in the group $GL(2, Z_p)$. Run the program for $p = 2$.

Theory

The order of matrix M in the group $GL(n, F)$ is least positive integer m such that

$$M^n \mod p = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Algorithm

```
Step 1: Prompt user to give input of $p$;
Step 2: Verify if p is a prime. If verification fails, show an error and exit;
Step 3: Define a list gl of elements of $GL(2, Z_p)$;
Step 4: Define a list order to save a pair of the elements and their orders;
Step 5: Run a loop for each element $M$ in $GL(2, Z_p)$;
Step 6: At each iteration, define $U = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. and $order = 0$;
Step 7: Run an inner loop;
Step 8: At each iteration, set U to UM mod p and order to order+1;
Step 9: If $U = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, save (element, order) pair to the list orders, and exit the inner loop;
Step 10: Exit outer loop;
Step 11: Print the order of elements of $GL(2, Z_p)$.
```

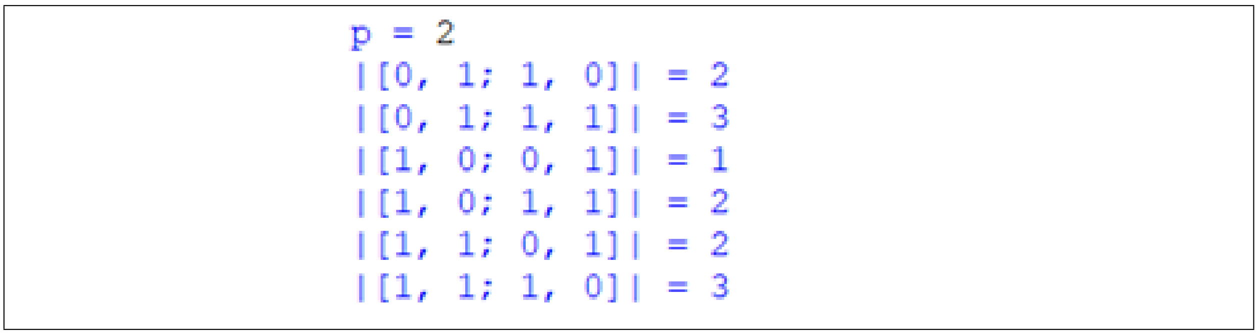
Program

The following PYTHON script implements the algorithm.

```
import numpy as np

#Function to check primality of an integer greater than 1
def isprime(n):
    for d in range(2, int(n**0.5) + 1):
        if n % d == 0:
            return False
    return True

#Function to calculate all permutations with repetitions of given size
def perm(s, r):
    permutations = [[]]
    for _ in range(r):
        temp_permutations = []
        for permutation in permutations:
            for element in s:
                temp_permutations.append(permutation + [element])
        permutations = temp_permutations
    return permutations

#Function to calculate determinant of 2x2 matrices
def det(m):
    return m[0, 0]*m[1, 1] - m[0, 1]*m[1, 0]

#Step 1
p = int(input('p = '))
#Step 2
if p <= 1 or not isprime(p):
    raise ValueError('p should be a prime number.')
#Step 3
permutations = perm(list(range(0, p)), 2*2)
gl = []
for permutation in permutations:
    m = np.matrix(np.reshape(permutation, (2, 2)))
    d = det(m)%p
    if d != 0:
        gl.append(m)
#Step 4
orders = []
I = np.matrix('1, 0; 0, 1')
#Step 5
for M in gl:
    #Step 6
    U = np.matrix(np.copy(I))
    order = 0
    #Step 7
    while True:
        #Step 8
        U = U*M %p
        order += 1
        #Step 9
        if (U==I).all():
            orders.append((M, order))
            break
#Step 11
for M, order in orders:
    print(f'|[{M[0, 0]}, {M[0, 1]}; {M[1, 0]}, {M[1, 1]}]| = {order}')
```

Output

The output of the program for the given input is as follows:



```
p = 2
|[0, 1; 1, 0]| = 2
|[0, 1; 1, 1]| = 3
|[1, 0; 0, 1]| = 1
|[1, 0; 1, 1]| = 2
|[1, 1; 0, 1]| = 2
|[1, 1; 1, 0]| = 3
```

Exercise Problem

Write a program that takes the input of a prime number $p < 10$ from the user and prints the order and the elements of the group $SL(2, Z_p)$. Run the program for $p = 3$.

Practical No. 15

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program that takes the input of a positive integer n. Then it prints the inverse of each of the elements of Z_n. Run the program for $n = 2, 5, 12, 18$.

Theory

The inverse of $k \epsilon Z_n$ is $m \epsilon Z_n$ such that $(k + m) \mod n = 0$.

Algorithm

Step 1: Prompt user to give input of integer n;
Step 2: Check if n is a positive integer. If not, show an error and exit;
Step 3: Define a list zn and save elements Z_n in it;
Step 4: Define a list $inverses$ to save element and inverse pair;
Step 5: Run a loop for each k in Z_n;
Step 6: Run an inner loop for each m in Z_n;
Step 7: At each iteration, if $(k + m) \mod n = 0$, save the element and its inverse as a pair to the list *inverses* and exit the inner loop;
Step 8: Exit the outer loop;
Step 9: Print the inverses of elements of Z_n.
.

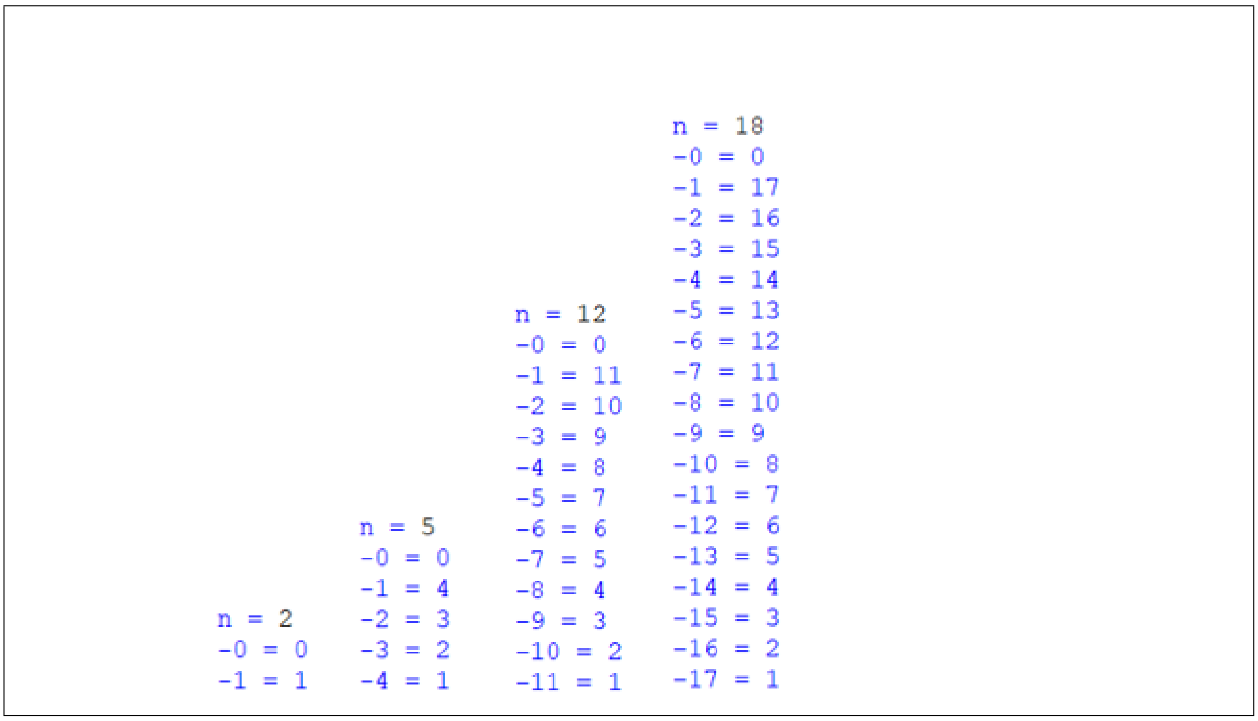
Program

The following PYTHON script implements the algorithm.

```
#Step 1
n = int(input('n = '))
#Step 2
if n <= 0:
    raise ValueError('n must be a positive integer.')
#Step 3
zn = list(range(n))
#Step 4
inverses = []
#Step 5
for k in zn:
    #Step 6
    for m in zn:
        #Step 7
        if (k+m)%n == 0:
            inverses.append((k, m))
            break
#Step 9
for element, inverse in inverses:
    print(f'-{element} = {inverse}')
```

Output

The output of the program for the given input is as follows:



```
                              n = 18
                              -0 = 0
                              -1 = 17
                              -2 = 16
                              -3 = 15
                              -4 = 14
                    n = 12    -5 = 13
                    -0 = 0    -6 = 12
                    -1 = 11   -7 = 11
                    -2 = 10   -8 = 10
                    -3 = 9    -9 = 9
                    -4 = 8    -10 = 8
                    -5 = 7    -11 = 7
          n = 5     -6 = 6    -12 = 6
          -0 = 0    -7 = 5    -13 = 5
          -1 = 4    -8 = 4    -14 = 4
n = 2     -2 = 3    -9 = 3    -15 = 3
-0 = 0    -3 = 2    -10 = 2   -16 = 2
-1 = 1    -4 = 1    -11 = 1   -17 = 1
```

Exercise Problem

Write a program that takes the input of an integer $n > 1$. Then it prints the inverse of each of the elements of $U(n)$. Run the program for $n = 10, 12, 15, 30$.

Practical No. 16

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program that takes the input of a prime number $p < 10$ from the user and prints the inverse of each element in the group $GL(2, Z_p)$. Run the program for $p = 2$.

Theory

The inverse of matrix M in the group $GL(n, F)$ is the matrix N such that

$$MN \mod p = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Algorithm

Step 1: Prompt user to give input of p;
Step 2: Verify if p is a prime. If verification fails, show an error and exit;
Step 3: Define a list gl of elements of $GL(2, Z_p)$;
Step 4: Define a list inverse to save a pair of the elements and their inverses;
Step 5: Run a loop for each element M in $GL(2, Z_p)$;
Step 6: At each iteration, Run another loop for each element N in $GL(2, Z_p)$;
Step 7: At each iteration, if

$$MN \mod p = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},$$

save the pair (M, N) to the list inverses and exit the inner loop;
Step 8: Exit outer loop;
Step 9: Print the inverses of elements of $GL(2, Z_p)$.

Program

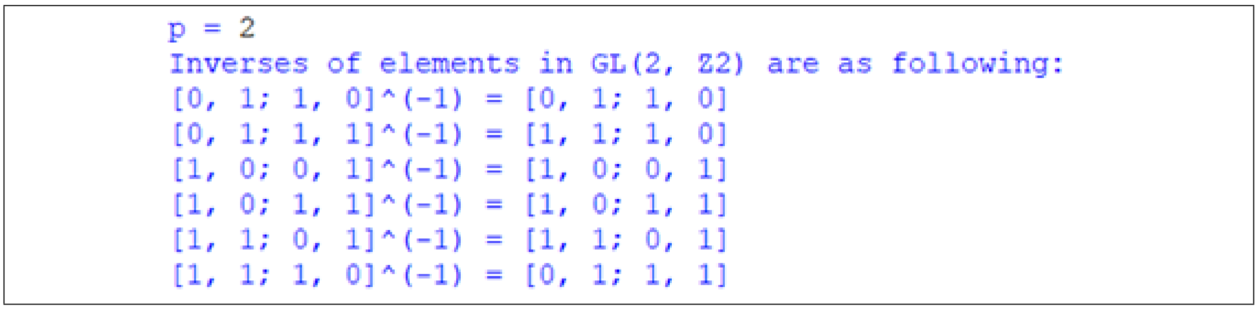
The following PYTHON script implements the algorithm.

```
import numpy as np

#Function to check primality of an integer greater than 1
def isprime(n):
    for d in range(2, int(n**0.5) + 1):
        if n % d == 0:
            return False
    return True

#Function to calculate all permutations with repetitions of given size
def perm(s, r):
    permutations = [[]]
    for _ in range(r):
        temp_permutations = []
        for permutation in permutations:
            for element in s:
                temp_permutations.append(permutation + [element])
        permutations = temp_permutations
    return permutations
#Function to calculate determinant of 2x2 matrices
def det(m):
    return m[0, 0]*m[1, 1] - m[0, 1]*m[1, 0]

#Step 1
```

```
p = int(input('p = '))
#Step 2
if p <= 1 or not isprime(p):
    raise ValueError('p should be a prime number.')
#Step 3
permutations = perm(list(range(0, p)), 2*2)
gl = []
for permutation in permutations:
    m = np.matrix(np.reshape(permutation, (2, 2)))
    d = det(m)%p
    if d != 0:
        gl.append(m)
#Step 4
inverses = []
I = np.matrix('1, 0; 0, 1')
#Step 5
for M in gl:
    #Step 6
    for N in gl:
        #Step 7
        if ((M*N % p)==I).all():
            inverses.append((M, N))
            break
#Step 9
print(f'Inverses of elements in GL(2, Z{p}) are as following:')
for M, N in inverses:
    print(f'[{M[0, 0]}, {M[0, 1]}; {M[1, 0]}, {M[1, 1]}]^(-1) =
    [{N[0, 0]}, {N[0, 1]}; {N[1, 0]}, {N[1, 1]}]')
```

Output

The output of the program for the given input is as follows:



```
p = 2
Inverses of elements in GL(2, Z2) are as following:
[0, 1; 1, 0]^(-1) = [0, 1; 1, 0]
[0, 1; 1, 1]^(-1) = [1, 1; 1, 0]
[1, 0; 0, 1]^(-1) = [1, 0; 0, 1]
[1, 0; 1, 1]^(-1) = [1, 0; 1, 1]
[1, 1; 0, 1]^(-1) = [1, 1; 0, 1]
[1, 1; 1, 0]^(-1) = [0, 1; 1, 1]
```

Exercise Problem

Write a program that takes the input of a prime number $p < 10$ from the user and prints the inverse of each element in the group $SL(2, Z_p)$. Run the program for $p = 3$.

Practical No. 17

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program that takes the input of a positive integer n. Then it prints all the unique cyclic subgroups generated by elements of Z_n. Run the program for $n = 2, 5, 12, 18$.

Theory

The cyclic subgroup k generated by $k \epsilon Z_n$ is given by

$$k = \{mk \mod n : m \epsilon Z\}$$

.

Algorithm

Step 1: Prompt user to give input of integer n;
Step 2: Check if n is a positive integer. If not, show an error and exit;
Step 3: Define a list zn and save elements Z_n in it;
Step 4: Define a list *subgroups* to save generators and generated subgroup pairs;
Step 5: Run a loop for each k in Z_n;
Step 6: At each iteration, define $u = 0$ and a list *subgroup* to save elements of k;
Step 7: Run an inner loop;
Step 8: Set $u = (u + k) \mod n$, save u to *subgroup* list. If $u = 0$, then exit the inner loop;
Step 9: If the current subgroup exists in the subgroups list, append k to the list of corresponding generators;
Step 10: Otherwise save (generators, subgroup) pair to the subgroups list;
Step 11: Exit the outer loop;
Step 12: Print all the cyclic subgroups of Z_n.

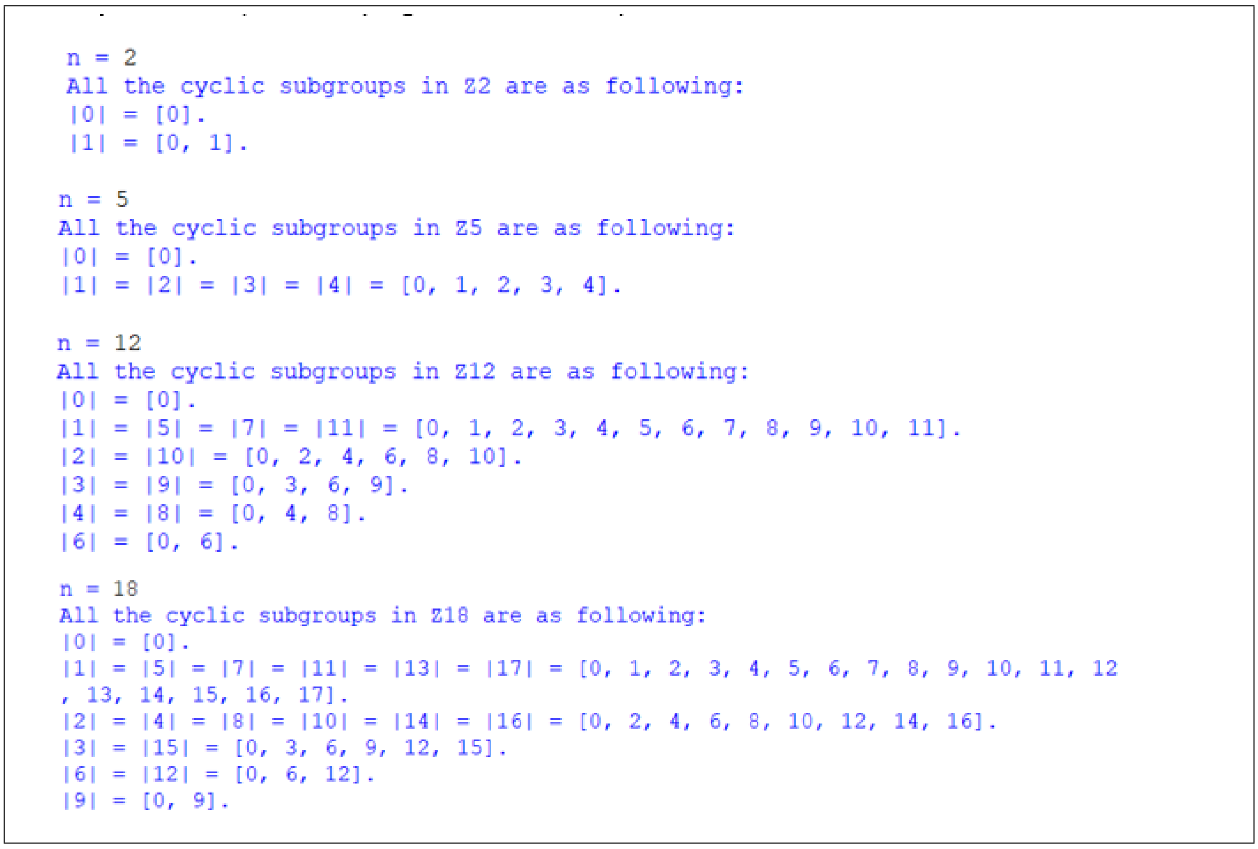
Program

The following PYTHON script implements the algorithm.

```
#Step 1
n = int(input('n = '))
#Step 2
if n <= 0:
    raise ValueError('n should be a positive integer.')
#Step 3
zn = list(range(n))
#Step 4
subgroups = []
#Step 5
for k in zn:
    #Step 6
    u = 0
    subgroup = []
    #Step 7
    while True:
        #Step 8
        u = (u + k)%n
        subgroup.append(u)
        if u == 0:
            break
    subgroup = sorted(subgroup)
    #Step 9
    saved = False
    for generators, savedSubgroup in subgroups:
        if subgroup == savedSubgroup:
            generators.append(k)
            saved = True
            break
    #Step 10
    if not saved:
        subgroups.append(([k], subgroup))
#Step 12
print(f'All the cyclic subgroups in Z{n} are as following:')
for generators, subgroup in subgroups:
    for generator in generators:
        print(f'|{generator}| = ', end='')
    print(f'{subgroup}.')
```

Output

The output of the program for the given input is as follows:



```
 n = 2
 All the cyclic subgroups in Z2 are as following:
 |0| = [0].
 |1| = [0, 1].

n = 5
All the cyclic subgroups in Z5 are as following:
|0| = [0].
|1| = |2| = |3| = |4| = [0, 1, 2, 3, 4].

n = 12
All the cyclic subgroups in Z12 are as following:
|0| = [0].
|1| = |5| = |7| = |11| = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11].
|2| = |10| = [0, 2, 4, 6, 8, 10].
|3| = |9| = [0, 3, 6, 9].
|4| = |8| = [0, 4, 8].
|6| = [0, 6].

n = 18
All the cyclic subgroups in Z18 are as following:
|0| = [0].
|1| = |5| = |7| = |11| = |13| = |17| = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
, 13, 14, 15, 16, 17].
|2| = |4| = |8| = |10| = |14| = |16| = [0, 2, 4, 6, 8, 10, 12, 14, 16].
|3| = |15| = [0, 3, 6, 9, 12, 15].
|6| = |12| = [0, 6, 12].
|9| = [0, 9].
```

Exercise Problem

Write a program that takes the input of an integer $n > 1$. Then it prints all the unique cyclic subgroups generated by elements of $U(n)$. Run the program for $n = 10, 12, 15, 30$.

Practical No. 18

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program that takes the input of a positive integer n and a prime number p. Then it prints all the unique cyclic subgroups generated by elements of $GL(n, Z_p)$. Run the program for $n = 2$ and $p = 3$.

Theory

The cyclic subgroup M generated by $M \epsilon GL(n, Z_p)$ is given by

$$M = \{M^m \mod p : m \epsilon Z\}$$

.

Algorithm

Step 1: Prompt user to give input of integer n;
Step 2: Check if n is a positive integer. If not, show an error and exit;
Step 3: Prompt user to give input of integer p;
Step 4: Check if p is a prime integer. If not, show an error and exit;
Step 5: Define a list *gl* and save elements of $GL(n, Z_p)$ in it;
Step 6: Define a list *subgroups* to save generators and generated subgroup pairs;
Step 7: Run a loop for each M in $GL(n, Z_p)$;
Step 8: At each iteration, define $U = I$ and a list *subgroup* to save elements of M;
Step 9: Run an inner loop;
Step 10: Set $U = UM \mod p$, save U to *subgroup* list. If $U = I$, then exit the inner loop;
Step 11: If the current subgroup exists in the subgroups list, append M to the list of corresponding generators;
Step 12: Otherwise save (generators, subgroup) pair to the subgroups list;
Step 13: Exit the outer loop;
Step 14: Print all the cyclic subgroups of $GL(n, Z_p)$.

Program

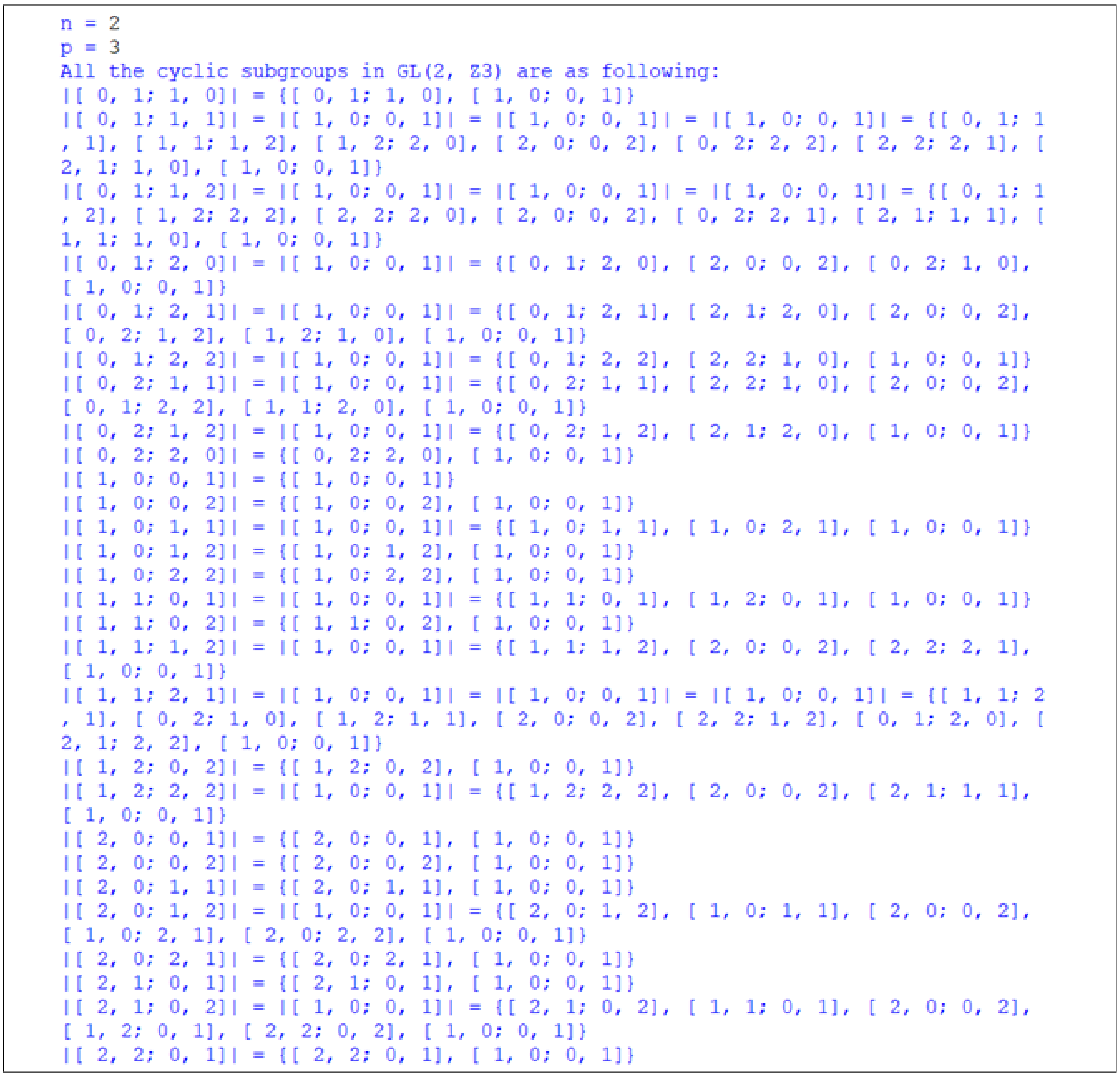
The following PYTHON script implements the algorithm.

```
import numpy as np

#Function to check primality of an integer greater than 1
def isprime(n):
    for d in range(2, int(n**0.5) + 1):
        if n % d == 0:
            return False
    return True

#Function to calculate all permutations with repetitions of given size
def perm(s, r):
    permutations = [[]]
    for _ in range(r):
        temp_permutations = []
        for permutation in permutations:
            for element in s:
                temp_permutations.append(permutation + [element])
        permutations = temp_permutations
    return permutations

#Function to calculate determinant of square matrices
def det(m):
    r, c = m.shape
    if r != c:
        raise IndexError(‘Given matrix is not a square matrix.’)
    if r == 1:
        return m[0, 0]
    d = 0
    for j in range(c):
        minor = np.delete(m, 0, 0)
        minor = np.delete(minor, j, 1)
        cofactor = (-1)**(j)*det(minor)
        d += m[0, j]*cofactor
    return d
#Function to print matrices on single line
def printmatrix(m):
    r, c = m.shape
    print(‘[ ’, end=’’)
    for i in range(r - 1):
        for j in range(c - 1):
            print(f‘{m[i, j]}, ’, end=’’)
        print(f‘{m[i, c-1]}; ’, end=’’)
    for j in range(c - 1):
        print(f‘{m[r-1, j]}, ’, end=’’)
    print(f‘{m[r-1, c-1]}]’, end=’’)

#Step 1
n = int(input(‘n = ’))
#Step 2
if n <= 0:
    raise ValueError(‘n should be a positive integer.’)
#Step 3
p = int(input(‘p = ’))
#Step 4
if p <= 1 or not isprime(p):
    raise ValueError(‘p should be a prime number.’)
```

```
#Step 5
permutations = perm(list(range(0, p)), n*n)
gl = []
for permutation in permutations:
    m = np.matrix(np.reshape(permutation, (n, n)))
    d = det(m)%p
    if d != 0:
        gl.append(m)
#Step 6
subgroups = []
I = np.matrix(np.eye(n, dtype=int))
#Step 7
for M in gl:
    #Step 8
    U = np.matrix(np.eye(n, dtype=int))
    subgroup = []
    #Step 9
    while True:
        #Step 10
        U = U*M%p
        subgroup.append(U)
        if (U == I).all():
            break
    #Step 11
    saved = False
    for generators, savedSubgroup in subgroups:
        if len(subgroup) != len(savedSubgroup):
            continue
        equal = True
        for M in subgroup:
            found = False
            for N in savedSubgroup:
                if (M==N).all():
                    found = True
                    break
            if not found:
                equal = False
                break

        if equal:
            generators.append(M)
            saved = True
            break
    #Step 12
    if not saved:
        subgroups.append(([M], subgroup))
#Step 14
print(f‘All the cyclic subgroups in GL({n}, Z{p}) are as following:’)
for generators, subgroup in subgroups:
    for generator in generators:
        print(‘|’, end=’’)
        printmatrix(generator)
        print(‘| = ’, end=’’)
    print(‘{’, end=’’)
    for M in subgroup[:-1]:
        printmatrix(M)
        print(‘, ’, end=’’)
    printmatrix(subgroup[-1])
    print(’}’)
```

Output

The output of the program for the given input is as follows:



```
n = 2
p = 3
All the cyclic subgroups in GL(2, Z3) are as following:
|[ 0, 1; 1, 0]| = {[ 0, 1; 1, 0], [ 1, 0; 0, 1]}
|[ 0, 1; 1, 1]| = |[ 1, 0; 0, 1]| = |[ 1, 0; 0, 1]| = |[ 1, 0; 0, 1]| = {[ 0, 1; 1
, 1], [ 1, 1; 1, 2], [ 1, 2; 2, 0], [ 2, 0; 0, 2], [ 0, 2; 2, 2], [ 2, 2; 2, 1], [
2, 1; 1, 0], [ 1, 0; 0, 1]}
|[ 0, 1; 1, 2]| = |[ 1, 0; 0, 1]| = |[ 1, 0; 0, 1]| = |[ 1, 0; 0, 1]| = {[ 0, 1; 1
, 2], [ 1, 2; 2, 2], [ 2, 2; 2, 0], [ 2, 0; 0, 2], [ 0, 2; 2, 1], [ 2, 1; 1, 1], [
1, 1; 1, 0], [ 1, 0; 0, 1]}
|[ 0, 1; 2, 0]| = |[ 1, 0; 0, 1]| = {[ 0, 1; 2, 0], [ 2, 0; 0, 2], [ 0, 2; 1, 0],
[ 1, 0; 0, 1]}
|[ 0, 1; 2, 1]| = |[ 1, 0; 0, 1]| = {[ 0, 1; 2, 1], [ 2, 1; 2, 0], [ 2, 0; 0, 2],
[ 0, 2; 1, 2], [ 1, 2; 1, 0], [ 1, 0; 0, 1]}
|[ 0, 1; 2, 2]| = |[ 1, 0; 0, 1]| = {[ 0, 1; 2, 2], [ 2, 2; 1, 0], [ 1, 0; 0, 1]}
|[ 0, 2; 1, 1]| = |[ 1, 0; 0, 1]| = {[ 0, 2; 1, 1], [ 2, 2; 1, 0], [ 2, 0; 0, 2],
[ 0, 1; 2, 2], [ 1, 1; 2, 0], [ 1, 0; 0, 1]}
|[ 0, 2; 1, 2]| = |[ 1, 0; 0, 1]| = {[ 0, 2; 1, 2], [ 2, 1; 2, 0], [ 1, 0; 0, 1]}
|[ 0, 2; 2, 0]| = {[ 0, 2; 2, 0], [ 1, 0; 0, 1]}
|[ 1, 0; 0, 1]| = {[ 1, 0; 0, 1]}
|[ 1, 0; 0, 2]| = {[ 1, 0; 0, 2], [ 1, 0; 0, 1]}
|[ 1, 0; 1, 1]| = |[ 1, 0; 0, 1]| = {[ 1, 0; 1, 1], [ 1, 0; 2, 1], [ 1, 0; 0, 1]}
|[ 1, 0; 1, 2]| = {[ 1, 0; 1, 2], [ 1, 0; 0, 1]}
|[ 1, 0; 2, 2]| = {[ 1, 0; 2, 2], [ 1, 0; 0, 1]}
|[ 1, 1; 0, 1]| = |[ 1, 0; 0, 1]| = {[ 1, 1; 0, 1], [ 1, 2; 0, 1], [ 1, 0; 0, 1]}
|[ 1, 1; 0, 2]| = {[ 1, 1; 0, 2], [ 1, 0; 0, 1]}
|[ 1, 1; 1, 2]| = |[ 1, 0; 0, 1]| = {[ 1, 1; 1, 2], [ 2, 0; 0, 2], [ 2, 2; 2, 1],
[ 1, 0; 0, 1]}
|[ 1, 1; 2, 1]| = |[ 1, 0; 0, 1]| = |[ 1, 0; 0, 1]| = |[ 1, 0; 0, 1]| = {[ 1, 1; 2
, 1], [ 0, 2; 1, 0], [ 1, 2; 1, 1], [ 2, 0; 0, 2], [ 2, 2; 1, 2], [ 0, 1; 2, 0], [
2, 1; 2, 2], [ 1, 0; 0, 1]}
|[ 1, 2; 0, 2]| = {[ 1, 2; 0, 2], [ 1, 0; 0, 1]}
|[ 1, 2; 2, 2]| = |[ 1, 0; 0, 1]| = {[ 1, 2; 2, 2], [ 2, 0; 0, 2], [ 2, 1; 1, 1],
[ 1, 0; 0, 1]}
|[ 2, 0; 0, 1]| = {[ 2, 0; 0, 1], [ 1, 0; 0, 1]}
|[ 2, 0; 0, 2]| = {[ 2, 0; 0, 2], [ 1, 0; 0, 1]}
|[ 2, 0; 1, 1]| = {[ 2, 0; 1, 1], [ 1, 0; 0, 1]}
|[ 2, 0; 1, 2]| = |[ 1, 0; 0, 1]| = {[ 2, 0; 1, 2], [ 1, 0; 1, 1], [ 2, 0; 0, 2],
[ 1, 0; 2, 1], [ 2, 0; 2, 2], [ 1, 0; 0, 1]}
|[ 2, 0; 2, 1]| = {[ 2, 0; 2, 1], [ 1, 0; 0, 1]}
|[ 2, 1; 0, 1]| = {[ 2, 1; 0, 1], [ 1, 0; 0, 1]}
|[ 2, 1; 0, 2]| = |[ 1, 0; 0, 1]| = {[ 2, 1; 0, 2], [ 1, 1; 0, 1], [ 2, 0; 0, 2],
[ 1, 2; 0, 1], [ 2, 2; 0, 2], [ 1, 0; 0, 1]}
|[ 2, 2; 0, 1]| = {[ 2, 2; 0, 1], [ 1, 0; 0, 1]}
```

Exercise Problem

Write a program that takes the input of a positive integer n and a prime number p. Then it prints all the unique cyclic subgroups generated by elements of $SL(n, Z_p)$. Run the program for $n = 2$ and $p = 3$.

Practical No. 19

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program to print all the left cosets of the subgroup generated by a given integer k in Z_n where $0 \leq k < n$. Assume n is less than 10. Run the program for $(n, k) = (4, 3)$ and $(8, 6)$. If $ba + k$, then what is the relation between cosets $a + k$ and $b + k$?

Theory

Let H be a subgroup of a group G. Then, the left coset aH ($a + H$ in additive notation) of H in G containing $a \epsilon G$ is defined as

$$aH = \{ah : hH\}.$$

In Z_n, we have $k = mk \mod n : m = 1, 2, = H$ (say).Then, for $a \epsilon Z_n$, the coset $a + H$ is given by

$$a + H = \{(a + mk) \mod n : m = 1, 2, \}.$$

Algorithm

Step 1: Prompt user to give input of integer n;
Step 2: Check if n is a positive integer. If not, show an error and exit;
Step 3: Prompt user to give input of integer p;
Step 4: Check if p is a prime integer. If not, show an error and exit;
Step 5: Define a list *gl* and save elements of $GL(n, Z_p)$ in it;
Step 6: Define a list *subgroups* to save generators and generated subgroup pair;
Step 7: Run a loop for each M in $GL(n, Z_p)$;
Step 8: At each iteration, define $U = I$ and a list *subgroup* to save elements of M;
Step 9: Run an inner loop;
Step 10: Set $U = UM \mod p$, save U to *subgroup* list. If $U = I$, then exit the inner loop;
Step 11: If the current subgroup exists in the subgroups list, append M to the list of corresponding generators;
Step 12: Otherwise save (generators, subgroup) pair to the subgroups list;
Step 13: Exit the outer loop;
Step 14: Print all the cyclic subgroups of $GL(n, Z_p)$.

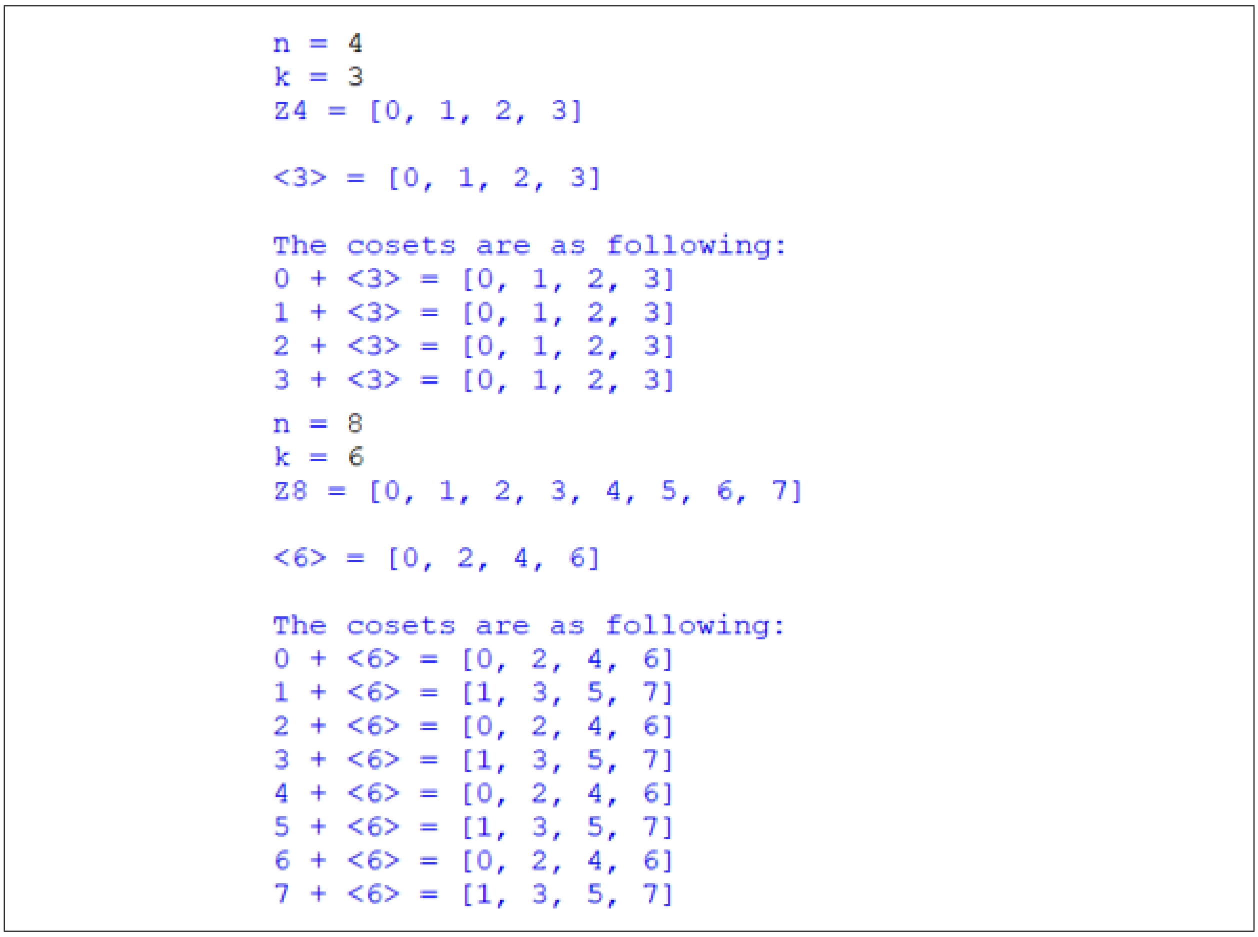
Program

The following PYTHON script implements the algorithm.

```
#Input
n = int(input("n = "))
#Step 1
if not (0 < n < 10):
    raise ValueError(f"n = {n} is not between 0 and 10.")
#Input
k = int(input("k = "))
#Step 2
if not (0 <= k < n):
    raise ValueError(f"k = {k} is either less than 0 or more than {n - 1}.")
#Step 3
elements = list(range(0, n))

#Step 4
subgroup = []
m = 0
while True:
    m = (m + k) % n
    subgroup.append(m)
    if m == 0:
        subgroup = sorted(subgroup)
        break
#Step 5
cosets = []
#Step 6
for a in elements:
    #Step 7
    coset = sorted([((a + b) % n) for b in subgroup])
    #Step 8
    cosets.append((a, coset))
#Step 10
print(f‘Z{n} = {elements}\n’)
print(f‘<{k}> = {subgroup}\n’)
print(f’The cosets are as following:’)

for a, coset in cosets:
    print(f‘{a} + <{k}> = {coset}’)
```

Output

The output of the program for the given input is as follows:



```
n = 4
k = 3
Z4 = [0, 1, 2, 3]

<3> = [0, 1, 2, 3]

The cosets are as following:
0 + <3> = [0, 1, 2, 3]
1 + <3> = [0, 1, 2, 3]
2 + <3> = [0, 1, 2, 3]
3 + <3> = [0, 1, 2, 3]
n = 8
k = 6
Z8 = [0, 1, 2, 3, 4, 5, 6, 7]

<6> = [0, 2, 4, 6]

The cosets are as following:
0 + <6> = [0, 2, 4, 6]
1 + <6> = [1, 3, 5, 7]
2 + <6> = [0, 2, 4, 6]
3 + <6> = [1, 3, 5, 7]
4 + <6> = [0, 2, 4, 6]
5 + <6> = [1, 3, 5, 7]
6 + <6> = [0, 2, 4, 6]
7 + <6> = [1, 3, 5, 7]
```

Conclusion

We observe that in each case if $b\epsilon a+k$, then $a+k=b+k$.

Exercise Problem

Write a program to print all the **distinct** left cosets of a subgroup generated by k in $U(n)$. Also, print the order of $U(n)$, order of subgroup generated by k, and the number of distinct left cosets. Assume n is less than 100. Run the program for $(n,k)=(8,5),(30,11),(36,25)$. What is the relation among the order of $U(n)$, order of subgroup generated by k, and the number of distinct left cosets?

Practical No. 20

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program that takes the input of a prime number p and two 22 matrices M and N in $GL(2, Z_p)$. Then, it prints the left and right cosets of the subgroup generated by M containing N i.e. NM and MN. Run the program for $p = 3$ and $(M, N) = (\begin{bmatrix}2 & 2\\0 & 1\end{bmatrix}, \begin{bmatrix}1 & 0\\1 & 2\end{bmatrix})$, $(\begin{bmatrix}2 & 0\\0 & 2\end{bmatrix}, \begin{bmatrix}1 & 2\\1 & 1\end{bmatrix})$. Is the left coset equal to the corresponding right coset in each case?

Theory

Let H be a subgroup of a group G. Then, the left coset aH ($a + H$ in additive notation) of H in G containing $a\epsilon G$ is defined as

$$aH = \{ah : hH\}.$$

Similarly, the right coset Ha ($H + a$ in additive notation) of H in G containing $a\epsilon G$ is defined as

$$Ha = \{ha : hH\}.$$

Algorithm

Step 1: Prompt user to give input of integer p;
Step 2: Check if p is a prime integer. If not, show an error and exit;
Step 3: Prompt user to give input of Matrix M;
Step 4: Check if M is a matrix $GL(2, Z_p)$. If not, show an error and exit; **Step 5:** Prompt user to give input of matrix N;
Step 6: Check if N is a matrix in $GL(2, Z_p)$. If not, show an error and exit;
Step 7: Define a list gl and save elements of $GL(n, Z_p)$ in it;
Step 8: Define a list *subgroup* and save elements of M in it;
Step 9: Define a list $leftCoset$ and save elements of NMzin it;
Step 10: Define a list *rightCoset* and save elements of MN in it;
Step 11: Print all the cosets NM and MN.

Program

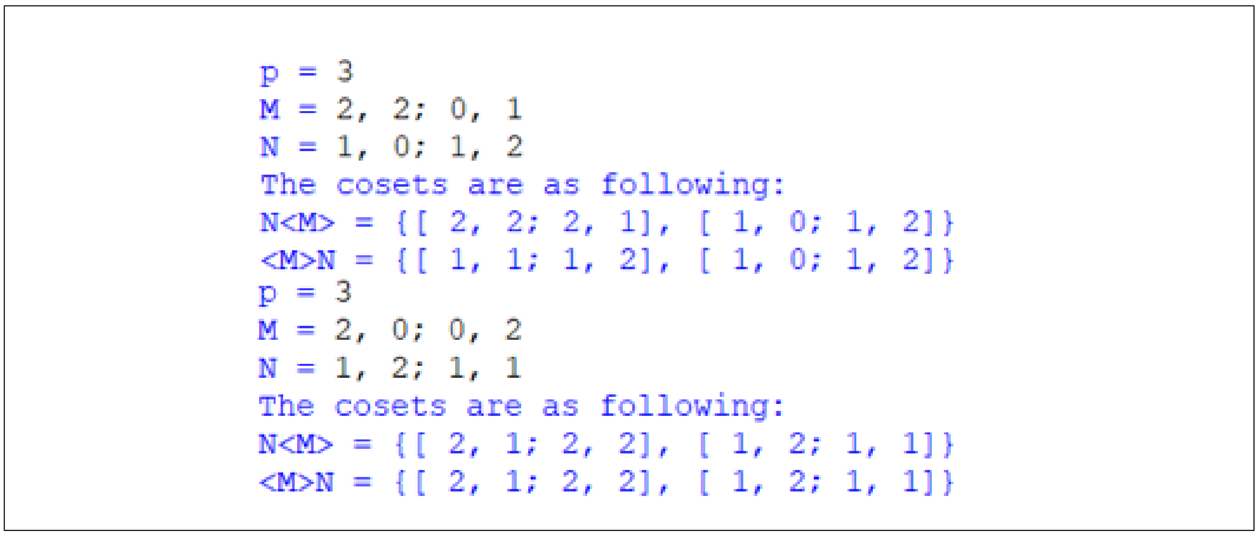
The following PYTHON script implements the algorithm.

```
import numpy as np

#Function to check primality of an integer greater than 1
def isprime(n):
    for d in range(2, int(n**0.5) + 1):
        if n % d == 0:
            return False
    return True

#Function to calculate all permutations with repetitions of given size
def perm(s, r):
    permutations = [[]]
    for _ in range(r):
        temp_permutations = []
        for permutation in permutations:
            for element in s:
                temp_permutations.append(permutation + [element])
        permutations = temp_permutations
    return permutations

#Function to calculate determinant of 2x2 matrices
def det(m):
    return m[0, 0]*m[1, 1] - m[0, 1]*m[1, 0]
#Function to print matrices on single line
def printmatrix(m):
    r, c = m.shape
    print('[ ', end='')
    for i in range(r - 1):
        for j in range(c - 1):
            print(f'{m[i, j]}, ', end='')
        print(f'{m[i, c-1]}; ', end='')
    for j in range(c - 1):
        print(f'{m[r-1, j]}, ', end='')
    print(f'{m[r-1, c-1]}]', end='')

#Step 1
p = int(input('p = '))
#Step 2
if p <= 1 or not isprime(p):
    raise ValueError('p should be a prime number.')
#Step 3
M = np.matrix(input('M = '), int)
#Step 4
mat = M % p
if (mat != M).any():
    raise ValueError(f'M contains entries not in Z{p}.')
d = det(M)%p
if d==0:
    raise ValueError(f'M does not belong to GL(2, Z{p}).')
#Step 5
N = np.matrix(input('N = '))
```

```
#Step 6
mat = N % p
if (mat != N).any():
    raise ValueError(f‘N contains entries not in Z{p}.’)
d = det(N)%p
if d==0:
    raise ValueError(f‘N does not belong to GL(2, Z{p}).’)
#Step 7
permutations = perm(list(range(0, p)), 4)
gl = []
for permutation in permutations:
    m = np.matrix(np.reshape(permutation, (2, 2)))
    d = det(m)%p
    if d != 0:
        gl.append(m)
#Step 8
I = np.matrix(np.eye(2, dtype=int))
U = np.matrix(np.eye(2, dtype=int))
subgroup = []
while True:
    U = U*M%p
    subgroup.append(U)
    if (U == I).all():
        break
#Step 9
leftCoset = [N*X%p for X in subgroup]
#Step 10
rightCoset = [X*N%p for X in subgroup]

#Step 11
print(f‘The cosets are as following:’)
print(’N<M> = {‘, end=’’)
for X in leftCoset[:-1]:
    printmatrix(X)
    print(‘, ’, end=’’)
printmatrix(leftCoset[-1])
print(’}’)
print(‘<M>N = {‘, end=’’)
for X in rightCoset[:-1]:
    printmatrix(X)
    print(‘, ’, end=’’)
printmatrix(rightCoset[-1])
print(’}’)
```

Output

The output of the program for the given input is as follows:



```
p = 3
M = 2, 2; 0, 1
N = 1, 0; 1, 2
The cosets are as following:
N<M> = {[ 2, 2; 2, 1], [ 1, 0; 1, 2]}
<M>N = {[ 1, 1; 1, 2], [ 1, 0; 1, 2]}
p = 3
M = 2, 0; 0, 2
N = 1, 2; 1, 1
The cosets are as following:
N<M> = {[ 2, 1; 2, 2], [ 1, 2; 1, 1]}
<M>N = {[ 2, 1; 2, 2], [ 1, 2; 1, 1]}
```

Conclusion

We observe that the left coset is not equal to the right coset in each case. In the first case, two cosets are not equal whereas in the second case, they are equal.

Exercise Problem

Write a program that takes the input of a prime number p and two 22 matrices M and N in $SL(2, Z_p)$. Then, it prints the left and right cosets of the subgroup generated by M containing N i.e. NM and MN. Run the program for $p = 3$ and $(M, N) = (\begin{bmatrix} 2 & 2 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 1 & 2 \end{bmatrix})$, $(\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}, \begin{bmatrix} 1 & 2 \\ 1 & 1 \end{bmatrix})$. Is the left coset equal to the corresponding right coset in each case? .

Practical No. 21

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program to calculate $a^k \mod p$ using Fermats Little Theorem for given integer a, positive integer k, and prime number p. Run the program for $(a, k, p) = (5, 19, 7), (7, 228, 11)$.

Theory

The Fermats Little Theorem provides an efficient way to calculate $a^k \mod p$. It states:
Fermats Little Theorem: For every integer a and every prime p, $a^p \mod p = a \mod p$.
Let $k > 0$. For any integer a:

- If $a \mod p = 0$, then $p \mid a \implies p \mid a^k \mod p = 0$.

- If $a \mod p \neq 0$. Then, dividing both sides by a, Fermats Little Theorem implies that $a^{(p-1)} \mod p = 1$. Let $r = k \mod (p-1)$. Then, for some integer q, we have $k = q(p-1) + r$, and therefore $a^k \mod p = a^{q(p-1)+r} \mod p = [(a^{(p-1)} \mod p)^q (a^r \mod p)] \mod p = a^r \mod p$.

Algorithm

Step 1: Prompt user to give input of a;
Step 2: Prompt user to give input of k;
Step 3: Check if k is a positive integer. If it is not, show an error and exit;
Step 4: Prompt user to give input of p;
Step 5: Check if p is a prime integer. If it is not, show an error and exit;
Step 6: Define the result variable;
Step 7: If $a \mod p = 0$, set result = 0;
Step 8: Otherwise calculate $r = k \mod (p-1)$ and set result = $a^r \mod p$;
Step 9: Print result.

Program

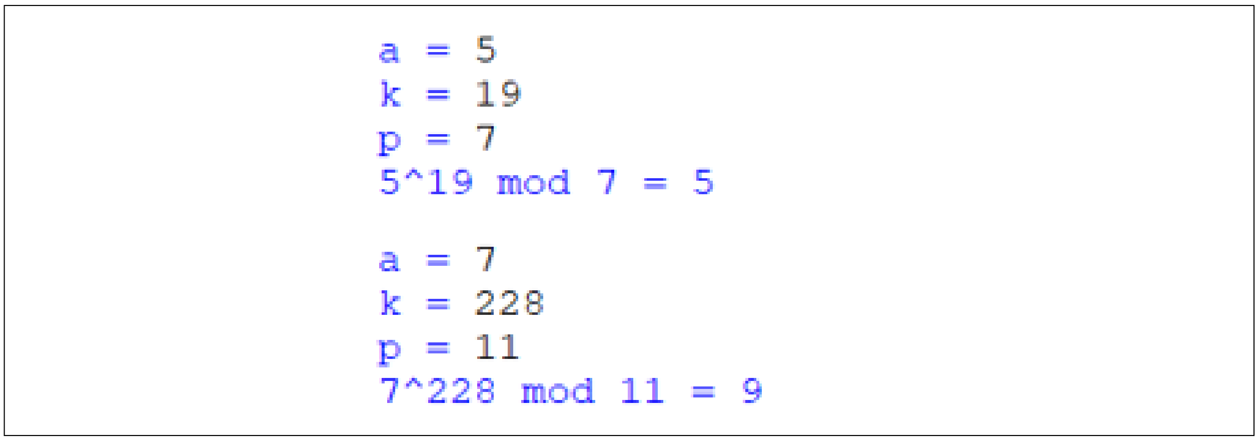
The following PYTHON script implements the algorithm.

```
#Function to check if primality of an integer greater than 2
def isprime(x):
    for i in range(2, x-1):
        if x % i == 0:
            return False
    else:
        return True
#Input
```

```
a = int(input("a = "))
k = int(input("k = "))
p = int(input("p = "))

#Step 1
if (p <= 1 or not isprime(p)):
    raise ValueError(f"p = {p} is not a prime.")

#Step 2
if k <= 0:
    raise ValueError(f"k = {k} is not a positive integer.")

#Step 3
result = 0

#Step 4
if a%p == 0:
    result = 0
#Step 5
else:
    r = k % (p - 1)
    result = a**r % p
#Step 6
print(f'{a}^{k} mod {p} = {result}')
```

Output

The output of the program for the given input is as follows:



```
a = 5
k = 19
p = 7
5^19 mod 7 = 5

a = 7
k = 228
p = 11
7^228 mod 11 = 9
```

Exercise Problem

Write a program to calculate $a^k \mod m$ using Eulers theorem for given non-negative integers a, k and $m > 1$ where a and m are relatively prime. Run the program for $(a, k, m) = (20, 61, 9), (32, 164, 15)$.

Chapter 6

Partial Differential Equations

Practical No. 1

Aim

To solve the first-order partial linear differential equation using Mathematica.

Problem

Write the differential equation $p+q=x+y$ in Mathematica. Also find the solution of the given equation.

Theory

If $P(x,y,z)p+Q(x,y,z)q=R(x,y,z)$ is a linear partial differential equation then the Lagrange's auxiliary equation becomes

$$\frac{dx}{P}=\frac{dy}{Q}=\frac{dz}{R}$$

After solving these equation, we get $f_1(x,y,z)=c_1$ and $f_2(x,y,z)=c_2$. The solution of problem is given by $\alpha(f_1,f_2)=0$.

Algorithm

Step1: write the partial differential equation taking taking D[u[x,y],x] as $\frac{\partial u}{\partial x}$ and D[u(x,y),y] as $\frac{\partial u}{\partial y}$;
Step2: click the run command button to see the given differential equation in Mathematica;
Step3: The partial derivatives $\frac{\partial u}{\partial x}$ and $\frac{\partial u}{\partial y}$ looks like $u^{(1,0)}$ and $u^{(0,1)}$;

Step4: Write the program to solve the above equation. The command for partial differential in Mathematica is DSolve;
Step5: Write solution of the equation as $DSolve[equation, dependentvariable, independentvariables]$;
Step6: Click the Run command button to see the solution of the given differential equation.

Program

The input and the output of the program is as following:

```
In[1]:= eqn = D[u[x, y], y] + D[u[x, y], x] - (x + y) == 0;
        DSolve[eqn, u[x, y], {x, y}]

Out[1]= -x - y + Derivative[0, 1][u][x, y] + Derivative[1, 0][u][x, y] == 0

Out[2]= {{u[x, y] -> x y + C[1][-x + y]}}
```

Conclusion

The solution of linear partial differential equation with dependent variable can be find out.

Exercise Problem

Write the differential equation $3p - 2q = 2 + y$ in Mathematica. Also find the solution of the given equation.

Practical No. 2

Aim

To solve the first-order nonlinear partial differential equation using Mathematica.

Problem

Write the differential equation $pq = 1$ in Mathematica. Also find the solution of the given equation.

Theory

The general nonlinear partial differential of order one of function $u[x, y]$ is given by $f(x, y, z, p, q) = 0$.
The Charpit's auxiliary is given

$$\frac{dx}{f_p} = \frac{dy}{f_q} = \frac{dz}{pf_p + qf_q} = \frac{-dp}{f_x + pf_z} = \frac{-dq}{f_x + qf_z} = \frac{df}{0}.$$

After solving these equation, we get $f_1(x, y, z) = p$ and $f_2(x, y, z) = q$. The solution of problem is given by $z = \int pdx + \int qdy + a$.

Algorithm

Step1: Write the partial differential equation taking D[u(x,y),x] as $\frac{\partial u}{\partial x}$ and D[u(x,y),y] as $\frac{\partial u}{\partial y}$;
Step2: Click the run command button to see the given differential equation in Mathematica;
Step3: The partial derivatives $\frac{\partial u}{\partial x}$ and $\frac{\partial u}{\partial y}$ look like $u^{(1,0)}$ and $u^{(0,1)}$;
Step4: Write the program to solve the above equation. The command for partial differential equations in Mathematica is DSolve;
Step5: Write solution of the equation as $DSolve[equation, dependentvariable, independentvariabl$
Step6: Click the Run command button to see the solution of the given differential equation.

Program

The input and the output of the program is as following:

```
In[52]:= DSolve[D[u[x, y], y] * D[u[x, y], x] == 1, u[x, y], {x, y}]

Out[52]= {{u[x, y] -> c1 + x/c2 + y c2}}
```

Conclusion

The solution of nonlinear partial differential equation with dependent variable can be find out.

Exercise Problem

Write the differential equation $pq = p + q$ in Mathematica. Also find the solution of the given equation.

Practical No. 3

Aim

To solve the first-order partial linear differential with variable coefficients using Mathematica.

Problem

Write the differential equation $p + zq = 0$ in Mathematica. Also find the solution of the given equation.

Theory

If $P(x, y, z)p + Q(x, y, z)q = R(x, y, z)$ is a linear partial differential equation then the Lagrange's auxiliary equation becomes

$$\frac{dx}{P} = \frac{dy}{Q} = \frac{dz}{R}$$

After solving these equation, we get $f_1(x, y, z) = c_1$ and $f_2(x, y, z) = c_2$. The solution of problem is given by $\alpha(f_1, f_2) = 0$.

Algorithm

Step1: Write the partial differential equation taking D[u[x,y],x] as $\frac{\partial u}{\partial x}$ and D[u[x,y],y] as $\frac{\partial u}{\partial y}$;
Step2: Click the run command button to see the given differential equation in Mathematica;
Step3: The partial derivatives $\frac{\partial u}{\partial x}$ and $\frac{\partial u}{\partial y}$ looks like $u^{(1,0)}$ and $u^{(0,1)}$;
Step4: Write the program to solve the above equation. The command for partial differential equation in Mathematica is DSolve;
Step5: Write solution of the equation as $Dsolve[equation, dependentvariable, independentvariables]$;
Step6: Click the Run command button to see the solution of the given differential equation.

Program

The input and the output of the program is as following:

```
In[1]:= z := u[x, y]
        p := D[u[x, y], x]
        q := D[u[x, y], y]
        BurgersEquation = p + z q == 0;
        sol = DSolve[BurgersEquation, u, {x, y}]

Out[5]= Solve[u[x, y] == c1 * (x - y/u[x, y]), u[x, y]]
```

Conclusion

The solution of linear partial differential equation with dependent variable can be find out.

Exercise Problem

Write the differential equation $xp + yq = xy$ in Mathematica. Also find the solution of the given equation.

Practical No. 4

Aim

To solve the first-order nonlinear partial differential equation with variable coefficients using Mathematica.

Problem

Write the differential equation $xyp + pq + yq = yz$ in Mathematica. Also find and verify the solution of the given equation.

Theory

The general first-order nonlinear partial differential equation for an unknown function $u[x, y]$ is given by $f(x, y, z, p, q) = 0$.
The Charpit's auxiliary equation is given

$$\frac{dx}{f_p} = \frac{dy}{f_q} = \frac{dz}{pf_p + qf_q} = \frac{-dp}{f_x + pf_z} = \frac{-dq}{f_x + qf_z} = \frac{df}{0}.$$

After solving these equation, we get $f_1(x, y, z) = p$ and $f_2(x, y, z) = q$. The solution of problem is given by $z = \int pdx + \int qdy + a$.

Algorithm

Step1: Write the partial differential equation taking D[u[x,y],x] as $\frac{\partial u}{\partial x}$ and D[u[x,y],y] as $\frac{\partial u}{\partial y}$;
Step2: Click the run command button to see the given differential equation in Mathematica;
Step3: The partial derivatives $\frac{\partial u}{\partial x}$ and $\frac{\partial u}{\partial y}$ looks like $u^{(1,0)}$ and $u^{(0,1)}$;
Step4: Write the program to solve the above equation. The command for partial differential equation in Mathematica is DSolve;
Step5: Write solution of the equation as $DSolve[equation, dependentvariable, independentvariables]$;
Step6: Click the Run command button to see the solution of the given differential equation.

Program

The input and the output of the program is as following:

```
In[22]:= eqn = x y p + p q + y q == y z;
         sol = DSolve[eqn, u, {x, y}]
Out[23]= {{u -> Function[{x, y}, x c1 + e^(y - c1 Log[y c1]) c2]}}
```

Conclusion

The solution of nonlinear partial differential equation with dependent variable can be find out.

Exercise Problem

Write the differential equation $xp + yq = pq$ in Mathematica. Also find the solution of the given equation.

Practical No. 5

Aim

To solve the first-order nonlinear partial differential equation of standard form-I using Mathematica.

Problem

Write the differential equation $p^2 + q^2 = 1$ in Mathematica. Also, find and verify the solution of the given equation.

Theory

The general first-order nonlinear partial differential equation for an unknown function $u[x, y]$ is given by $f(x, y, z, p, q) = 0$.
The Charpit's auxiliary equation is given

$$\frac{dx}{f_p} = \frac{dy}{f_q} = \frac{dz}{pf_p + qf_q} = \frac{-dp}{f_x + pf_z} = \frac{-dq}{f_x + qf_z} = \frac{df}{0}.$$

After solving these equation, we get $f_1(x, y, z) = p$ and $f_2(x, y, z) = q$. The solution of problem is given by $z = \int pdx + \int qdy + a$.

Algorithm

Step1: Write the partial differential equation taking D[u[x,y],x] as $\frac{\partial u}{\partial x}$ and D[u[x,y],y] as $\frac{\partial u}{\partial y}$;
Step2: Click the Run command button to see the given differential equation in Mathematica;
Step3: The partial derivatives $\frac{\partial u}{\partial x}$ and $\frac{\partial u}{\partial y}$ looks like $u^{(1,0)}$ and $u^{(0,1)}$;
Step4: Write the program to solve the above equation. The command for partial differential equation in Mathematica is DSolve;
Step5: Write solution of the equation as $DSolve[equation, dependentvariable, independentvariabl$
Step6: Click the run command button to see the solution of the given differential equation.

Program

The input and the output of the program is as following:

```
 In[30]:= eqn = p^2 + q^2 == 1;
          sol = u /. DSolve[eqn, u, {x, y}][[1]] /. C[1][t_] -> t
          eqn /. {u -> sol}

Out[31]= Function[{x, y}, c1 + y c2 - x Sqrt[1 - c2^2]]

Out[32]= True
```

Conclusion

The solution of nonlinear partial differential equation with dependent variable can be find out.

Exercise Problem

Write the differential equation $p^2 + q = q^2$ in Mathematica. Also find the solution of the given equation.

Practical No. 6

Aim

To solve the first-order nonlinear partial differential equation of standard form-II(Clairat equation) using Mathematica.

Problem

Write the differential equation $z = xp + yq + 2pq\sqrt{1 - p^2}$ in Mathematica. Also find and verify the solution of the given equation.

Theory

The general first-order nonlinear partial differential equation for an unknown function $u[x, y]$ is given by $f(x, y, z, p, q) = 0$.
The Charpit's auxiliary equation is given

$$\frac{dx}{f_p} = \frac{dy}{f_q} = \frac{dz}{pf_p + qf_q} = \frac{-dp}{f_x + pf_z} = \frac{-dq}{f_x + qf_z} = \frac{df}{0}.$$

After solving these equation, we get $f_1(x, y, z) = p$ and $f_2(x, y, z) = q$. The solution of problem is given by $z = \int pdx + \int qdy + a$.

Algorithm

Step1: Write the partial differential equation taking D[u[x,y],x] as $\frac{\partial u}{\partial x}$ and D[u[x,y],y] as $\frac{\partial u}{\partial y}$;
Step2: Click the Run command button to see the given differential equation in Mathematica;
Step3: The partial derivatives $\frac{\partial u}{\partial x}$ and $\frac{\partial u}{\partial y}$ look like $u^{(1,0)}$ and $u^{(0,1)}$;

Step4: Write the program to solve the above equation. The command for partial differential equation in Mathematica is DSolve;
Step5: Write solution of the equation as $DSolve[equation, dependentvariable, independentvariabl$
Step6: Click the run command button to see the solution of the given differential equation.

Program

The input and the output of the program is as following:

```
In[18]:= Clairaut = z == x*p + y*q + 2*p*q*Sqrt[1 - p^2];
         sol = DSolve[Clairaut, u, {x, y}]

Out[19]= {{u -> Function[{x, y}, x c1 + y c2 + 2 c1 Sqrt[1 - c1^2] c2]}}
```

Conclusion

The solution of nonlinear partial differential equation with dependent variable can be find out.

Exercise Problem

Write the differential equation $pq = 4z$ in Mathematica. Also find the solution of the given equation.

Practical No. 7

Aim

To solve the first-order nonlinear partial differential equation of standard form-III (Only p, q and z involve) using Mathematica.

Problem

Write the differential equation $p^2 + q^2 + 4z = 4$ in Mathematica. Also, find and verify the solution of the given equation.

Theory

The general first-order nonlinear partial differential equation for an unknown function $u[x, y]$ is given by $f(x, y, z, p, q) = 0$. The Charpit's auxiliary equation is given

$$\frac{dx}{f_p} = \frac{dy}{f_q} = \frac{dz}{pf_p + qf_q} = \frac{-dp}{f_x + pf_z} = \frac{-dq}{f_x + qf_z} = \frac{df}{0}.$$

After solving these equation, we get $f_1(x, y, z) = p$ and $f_2(x, y, z) = q$. The solution of problem is given by $z = \int pdx + \int qdy + a$.

Algorithm

Step1: Write the partial differential equation taking D[u[x,y],x] as $\frac{\partial u}{\partial x}$ and D[u[x,y],y] as $\frac{\partial u}{\partial y}$;
Step2: Click the Run command button to see the given differential equation in Mathematica;
Step3: The partial derivatives $\frac{\partial u}{\partial x}$ and $\frac{\partial u}{\partial y}$ looks like $u^{(1,0)}$ and $u^{(0,1)}$;
Step4: Write the program to solve the above equation. The command for partial differential equation in Mathematica is DSolve;
Step5: Write solution of the equation as DSolve[equation,dependent variable,indepndent variables];
Step6: Click the Run command button to see the solution of the given differential equation.

Program

The input and the output of the program is as following:

```
In[1]:= sol = DSolve[4 z + p^2 + q^2 == 4, u, {x, y}]

Out[1]= {{u -> Function[{x, y},
          (1 - y^2 - 2 x y c1 + c1^2 + x^2 c1^2 - 2 y c2 - 2 x c1 c2 - c2^2) /
          (1 + c1^2)]}}
```

Conclusion

The solution of nonlinear partial differential equation with dependent variable can be find out.

Exercise Problem

Write the differential equation $pq = 4z$ in Mathematica. Also find the solution of the given equation.

Practical No. 8

Aim

To solve the first-order nonlinear partial differential equation of standard form-IV ($f_{1(x,p)} = f_{2(y,q)}$) using Mathematica.

Problem

Write the differential equation $x(1+y)p = y(1+x)q$ in Mathematica. Also find and verify the solution of the given equation.

Theory

The general first order nonlinear partial differential equation for an unknown function $u[x,y]$ is given by $f(x,y,z,p,q) = 0$. The Charpit's auxiliary equation is given

$$\frac{dx}{f_p} = \frac{dy}{f_q} = \frac{dz}{pf_p + qf_q} = \frac{-dp}{f_x + pf_z} = \frac{-dq}{f_x + qf_z} = \frac{df}{0}.$$

After solving these equation, we get $f_1(x,y,z) = p$ and $f_2(x,y,z) = q$. The solution of problem is given by $z = \int pdx + \int qdy + a$.

Algorithm

Step1: Write the partial differential equation taking D[u[x,y],x] as $\frac{\partial u}{\partial x}$ and D[u[x,y],y] as $\frac{\partial u}{\partial y}$;
Step2: Click the Run command button to see the given differential equation in Mathematica;
Step3: The partial derivatives $\frac{\partial u}{\partial x}$ and $\frac{\partial u}{\partial y}$ look like $u^{(1,0)}$ and $u^{(0,1)}$;
Step4: Write the program to solve the above equation. The command for partial differential equation in Mathematica is DSolve;
Step5: Write solution of the equation as DSolve[equation,dependent variable,indepndent variables];
Step6: Click the Run command button to see the solution of the given differential equation.

Program

The input and the output of the program is as following:

```
In[26]:= eqn = x (1 + y) p == y (1 + x) q;
         sol = DSolve[eqn, u, {x, y}]
```

```
Out[27]= {{u -> Function[{x, y}, c1 y - Log[Exp[x] / (x y)]]}}
```

Conclusion

The solution of nonlinear partial differential equation with dependent variable can be find out.

Exercise Problem

Write the differential equation $p - 3x^2 = q^2 - y$ in Mathematica. Also find the solution of the given equation.

Practical No. 9

Aim

To find the singular solution of non-linear partial differential equation using Mathematica.

Problem

Write the differential equation $4z + p^2 + q^2 = 4$ in Mathematica. Also find and verify the solution of the given equation.

Theory

The general first order nonlinear partial differential equation for an unknown function $u[x, y]$ is given by $f(x, y, z, p, q) = 0$. The Charpit's auxiliary equation is given

$$\frac{dx}{f_p} = \frac{dy}{f_q} = \frac{dz}{pf_p + qf_q} = \frac{-dp}{f_x + pf_z} = \frac{-dq}{f_x + qf_z} = \frac{df}{0}.$$

After solving these equation, we get $f_1(x, y, z) = p$ and $f_2(x, y, z) = q$. The solution of problem is given by $z = \int pdx + \int qdy + a$.

Algorithm

Step1: Write the partial differential equation taking D[u[x,y],x] as $\frac{\partial u}{\partial x}$ and D[u[x,y],y] as $\frac{\partial u}{\partial y}$;

Step2: Click the Run command button to see the given differential equation in Mathematica;
Step3: The partial derivatives $\frac{\partial u}{\partial x}$ and $\frac{\partial u}{\partial y}$ looks like $u^{(1,0)}$ and $u^{(0,1)}$;
Step4: Write the program to solve the above equation. The command for partial differential equation in Mathematica is DSolve;
Step5: Write solution of the equation as DSolve[equation,dependent variable,indepndent variables];
Step6: Click the Run command button to see the solution of the given differential equation.

Program

The input and the output of the program is as following:

```
In[28]:= sol = DSolve[4 z + p^2 + q^2 == 4, u, {x, y}];
twoparameterfamily = u[x, y] == (u[x, y] /. sol[[1]]);
envelopeoftwoparameterfamily = Eliminate[{twoparameterfamily,
D[twoparameterfamily, C[1]], D[twoparameterfamily, C[2]]}, {C[1], C[2]}]

Out[30]= u[x, y] == 1
```

Conclusion

The solution of nonlinear partial differential equation with dependent variable can be find out.

Exercise Problem

Find the singular integral of nonlinear partial differential equation $z^2(p^2+q^2+1) = 1$

Practical No. 10

Aim

To solve the homogeneous second-order linear partial differential equation with constant coefficients using Mathematica.

Problem

To solve the homogeneous second-order linear partial differential equation with constant coefficients using Mathematica.

$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$ in Mathematica. Also, find the solution of the given equation.

Theory

The second order linear partial differential equation is given by

$$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2} + d\frac{\partial u}{\partial x} + e\frac{\partial u}{\partial y} + fu = g$$

Where $u = u(x, y)$ and a,b,c,d,e,f,g are functions of x and y only. If g=0, then the equation becomes homogeneous. The solution of second order partial differential equation possible in mathematica by command DSolve if principle part, i.e. $a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2}$ have constant coefficients and the non principle part is vanishing, i.e. 0.

Algorithm

Step1: Write the partial differential equation taking D[u[x,y]{x,2}] as $\frac{\partial^2 u}{\partial x^2}$ and D[u[x,y]{y,2}] as $\frac{\partial^2 u}{\partial y^2}$;
Step2: Click the Run command button to see the given differential equation in Mathematica;
Step3: The partial derivatives $\frac{\partial^2 u}{\partial x^2}$ and $\frac{\partial^2 u}{\partial y^2}$ looks like $u^{(2,0)}$ and $u^{(0,2)}$;
Step4: Write the program to solve the above equation. The command for partial differential equation in Mathematica is DSolve;
Step5: Write solution of the equation as DSolve[equation,dependent variable,{independent variables}];
Step6: Click the Run command button to see the solution of the given differential equation.

Program

The input and the output of the program is as following:

```
In[64]:= LaplaceEquation = D[u[x, y], {x, 2}] + D[u[x, y], {y, 2}] == 0;
         DSolve[LaplaceEquation, u[x, y], {x, y}]

Out[64]= u^(0,2)[x, y] + u^(2,0)[x, y] == 0

Out[65]= {{u[x, y] -> c1[i x + y] + c2[-i x + y]}}
```

Conclusion

The solution of homogeneous second order linear partial differential equation with constant coefficients can be find out.

Exercise Problem

Write the differential equation $\frac{\partial^2 u}{\partial x^2} + 2\frac{\partial^2 u}{\partial y^2} = 0$. in Mathematica. Also find the solution of the given equation.

Practical No. 11

Aim

To solve the non-homogeneous second order linear partial linear differential equation with constant coefficients using Mathematica.

Problem

Write the differential equation $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 1$ in Mathematica. Also find the solution of the given equation.

Theory

The second order linear partial differential equation is given by

$$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2} + d\frac{\partial u}{\partial x} + e\frac{\partial u}{\partial y} + fu = g.$$

where $u = u(x, y)$ and a,b,c,d,e,f,g are functions of x and y only.If g=0, then the equation becomes homogeneous. The solution of second order partial differential equation possible in Mathematica by command DSolve if principle part is vanishing, i.e.0.

Algorithm

Step1: Write the partial differential equation taking D[u[x,y] {x,2}] as $\frac{\partial^2 u}{\partial x^2}$ and D[u[x,y] {y,2}] as $\frac{\partial^2 u}{\partial y^2}$;
Step2: Click the run command button to see the given differential equation in Mathematica; **Step3:** The partial derivatives $\frac{\partial^2 u}{\partial x^2}$ and $\frac{\partial^2 u}{\partial y^2}$ looks like $u^{(2,0)}$ and $u^{(0,2)}$;
Step4: Write the program to solve the above equation. The command for partial differential equation in Mathematica is Dsolve;
Step5: Write solution of the equation as DSolve[equation,dependent variable,{independent variables}];
Step6: Click the Run command button to see the solution of the given differential equation.

Program

The input and the output of the program is as following:

```
In[66]:= eqn = D[u[x, y], {x, 2}] + D[u[x, y], {y, 2}] == 1;
         DSolve[eqn, u[x, y], {x, y}]

Out[66]= u^(0,2)[x, y] + u^(2,0)[x, y] == 1

Out[67]= {{u[x, y] -> x^2/2 + c1[i x + y] + c2[-i x + y]}}
```

Conclusion

The solution of non-homogeneous second order linear partial differential equation with constant coefficients can be find out.

Exercise Problem

Write the differential equation $\frac{\partial^2 u}{\partial x^2} + 2\frac{\partial^2 u}{\partial y^2} = 1$ in Mathematica. Also find the solution of the given equation.

Practical No. 12

Aim

To solve the non-homogeneous second order linear partial differential equation with constant coefficients using Mathematica.

Problem

Write the differential equation $\frac{\partial^2 u}{\partial x^2} = 2$ in Mathematica. Also find the solution of the given equation.

Theory

The second order linear partial differential equation is given by

$$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2} + d\frac{\partial u}{\partial x} + e\frac{\partial u}{\partial y} + fu = g.$$

where, $u = u(x, y)$ and a,b,c,d,e,f,g are functions of x and y only. If g=0, then the equation becomes homogeneous. The solution of second order partial differential equation possible in Mathematica by command DSolve if principle part, i.e. $a\frac{\partial^2 u}{\partial x^2}+b\frac{\partial^2 u}{\partial x \partial y}+c\frac{\partial^2 u}{\partial y^2}$ have constant coefficients and the non principle part is vanishing, i.e.0.

Algorithm

Step1: Write the partial differential equation taking D[u[x,y], {x,2}] as $\frac{\partial^2 u}{\partial x^2}$ and D[u[x,y], {y,2}] as $\frac{\partial^2 u}{\partial y^2}$;
Step2: Click the run command button to see the given differential equation in Mathematica;
Step3: The partial derivatives $\frac{\partial^2 u}{\partial x^2}$ and $\frac{\partial^2 u}{\partial y^2}$ looks like $u^{(2,0)}$ and $u^{(0,2)}$;
Step4: Write the program to solve the above equation. The command for partial differential equation in Mathematica is DSolve;
Step5: Write solution of the equation as DSolve[equation,dependent variable,{independent variables}];
Step6: Click the run command button to see the solution of the given differential equation.

Program

The input and the output of the program is as following:

```
 In[66]:= eqn = D[u[x, y], {x, 2}] + D[u[x, y], {y, 2}] == 1;
         DSolve[eqn, u[x, y], {x, y}]

Out[66]= u^(0,2)[x, y] + u^(2,0)[x, y] == 1

Out[67]= {{u[x, y] -> x^2/2 + c1[i x + y] + c2[-i x + y]}}
```

Conclusion

The solution of homogeneous second order linear partial differential equation with constant coefficients can be find out.

Exercise Problem

Write the differential equation $\frac{\partial^2 u}{\partial x^2}+2\frac{\partial^2 u}{\partial y^2}=0$ in Mathematica. Also find the solution of the given equation.

Practical No. 13

Aim

To solve the hyperbolic equation using Mathematica.

Problem

Write the differential equation $2\frac{\partial^2 u}{\partial x^2}+7\frac{\partial^2 u}{\partial x \partial y}-1\frac{\partial^2 u}{\partial y^2}=0.$ in Mathematica. Also, find and verify the solution of the given equation.

Theory

The second order linear partial differential equation is given by

$$a\frac{\partial^2 u}{\partial x^2}+b\frac{\partial^2 u}{\partial x \partial y}+c\frac{\partial^2 u}{\partial y^2}+d\frac{\partial u}{\partial x}+e\frac{\partial u}{\partial y}+fu=g.$$

where, $u=u(x,y)$ and a,b,c,d,e,f,g are functions of x and y only. If g=0, then the equation becomes homogeneous. The solution of second order partial differential equation possible in Mathematica by command DSolve if principle part, i.e. $a\frac{\partial^2 u}{\partial x^2}+b\frac{\partial^2 u}{\partial x \partial y}+c\frac{\partial^2 u}{\partial y^2}$ have constant coefficients and the non principle part is vanishing, i.e.0.

Algorithm

Step1: Write the partial differential equation taking D[u[x,y], {x,2}] as $\frac{\partial^2 u}{\partial x^2}$ and D[u[x,y], {y,2}] as $\frac{\partial^2 u}{\partial y^2}$;
Step2: Click the run command button to see the given differential equation in Mathematica;
Step3: The partial derivatives $\frac{\partial^2 u}{\partial x^2}$ and $\frac{\partial^2 u}{\partial y^2}$ looks like $u^{(2,0)}$ and $u^{(0,2)}$;
Step4: Write the program to solve the above equation. The command for partial differential equation in Mathematica is DSolve;
Step5: Write solution of the equation as DSolve[equation,dependent variable,{independent variables}];
Step6: Click the run command button to see the solution of the given differential equation.

Problem

The input and the output of the program is as following:

```
In[12]:= eqn = 2*D[u[x, y], {x, 2}] + 7*D[u[x, y], x, y] - D[u[x, y], {y, 2}] == 0;
         sol = DSolve[eqn, u, {x, y}];
         eqn /. sol // Simplify
```

```
Out[12]= -u^(0,2)[x, y] + 7 u^(1,1)[x, y] + 2 u^(2,0)[x, y] == 0

Out[13]= {{u -> Function[{x, y},
            c1[-1/4 (7 + Sqrt[57]) x + y] + c2[-1/4 (7 - Sqrt[57]) x + y]]}}

Out[14]= {True}
```

Conclusion

The solution of homogeneous second order linear partial differential equation with constant coefficients can be find out.

Exercise Problem

Write the differential equation $1\frac{\partial^2 u}{\partial x^2} + 5\frac{\partial^2 u}{\partial x \partial y} - 1\frac{\partial^2 u}{\partial y^2} = 0$ in Mathematica. Also find the solution of the given equation.

Practical No. 14

Aim

To solve the elliptic equation using Mathematica.

Problem

Write the differential equation $3\frac{\partial^2 u}{\partial x^2} + 1\frac{\partial^2 u}{\partial x \partial y} + 5\frac{\partial^2 u}{\partial y^2} = 0$ in Mathematica. Also find and verify the solution of the given equation.

Theory

The second order linear partial differential equation is given by

$$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2} + d\frac{\partial u}{\partial x} + e\frac{\partial u}{\partial y} + fu = g.$$

where, $u = u(x, y)$ and a,b,c,d,e,f,g are functions of x and y only. If g=0, then the equation becomes homogeneous. The solution of second order partial differential equation possible in Mathematica by command DSolve if principle part, i.e.$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2}$ have constant coefficients and the non principle part is vanishing, i.e.0.

Algorithm

Step1: Write the partial differential equation taking D[u[x,y], {x,2}] as $\frac{\partial^2 u}{\partial x^2}$ and D[u[x,y], {y,2}] as $\frac{\partial^2 u}{\partial y^2}$;
Step2: Click the run command button to see the given differential equation in Mathematica;
Step3: The partial derivatives $\frac{\partial^2 u}{\partial x^2}$ and $\frac{\partial^2 u}{\partial y^2}$ looks like $u^{(2,0)}$ and $u^{(0,2)}$;
Step4: Write the program to solve the above equation. The command for partial differential equation in Mathematica is DSolve;
Step5: Write solution of the equation as DSolve[equation,dependent variable,{independent variables}];
Step6: Click the run command button to see the solution of the given differential equation.

Problem

The input and the output of the program is as following:

```
In[15]:= eqn = 3*D[u[x, y], {x, 2}] + 1*D[u[x, y], x, y] + 5*D[u[x, y], {y, 2}] == 0;
         sol = DSolve[eqn, u, {x, y}];
         eqn /. sol // Simplify

Out[15]= 5 u^(0,2)[x, y] + u^(1,1)[x, y] + 3 u^(2,0)[x, y] == 0

Out[16]= {{u -> Function[{x, y},
            c1[1/6 (-1 + I Sqrt[59]) x + y] + c2[1/6 (-1 - I Sqrt[59]) x + y]]}}

Out[17]= {True}
```

Conclusion

The solution of homogeneous second order linear partial differential equation with constant coefficients can be find out.

Exercise Problem

Write the differential equation $1\frac{\partial^2 u}{\partial x^2} + 1\frac{\partial^2 u}{\partial x \partial y} + 5\frac{\partial^2 u}{\partial y^2} = 0$ in Mathematica. Also find the solution of the given equation.

Practical No. 15

Aim

To solve the parabolic equation using Mathematica.

Problem

Write the differential equation $4\frac{\partial^2 u}{\partial x^2} + 4\frac{\partial^2 u}{\partial x \partial y} + 1\frac{\partial^2 u}{\partial y^2} = 0$ in Mathematica. Also, find and verify the solution of the given equation.

Theory

The second order linear partial differential equation is given by

$$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2} + d\frac{\partial u}{\partial x} + e\frac{\partial u}{\partial y} + fu = g.$$

where, $u = u(x, y)$ and a,b,c,d,e,f,g are functions of x and y only. If g=0, then the equation becomes homogeneous. The solution of second order partial differential equation possible in Mathematica by command DSolve if principle part, i.e.$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2}$ have constant coefficients and the non principle part is vanishing, i.e.0.

Algorithm

Step1: Write the partial differential equation taking D[u[x,y], {x,2}] as $\frac{\partial^2 u}{\partial x^2}$ and D[u[x,y], {y,2}] as $\frac{\partial^2 u}{\partial y^2}$;
Step2: Click the run command button to see the given differential equation in Mathematica;
Step3: The partial derivatives $\frac{\partial^2 u}{\partial x^2}$ and $\frac{\partial^2 u}{\partial y^2}$ looks like $u^{(2,0)}$ and $u^{(0,2)}$;
Step4: Write the program to solve the above equation. The command for partial differential equation in Mathematica is DSolve;
Step5: Write solution of the equation as DSolve[equation,dependent variable,{independent variables}];
Step6: Click the run command button to see the solution of the given differential equation.

Program

The input and the output of the program is as following:

```
In[18]:= eqn = 4*D[u[x, y], {x, 2}] + 4*D[u[x, y], x, y] + 1*D[u[x, y], {y, 2}] =
         sol = DSolve[eqn, u, {x, y}];
         eqn /. sol // Simplify
```

```
Out[18]= u^(0,2)[x, y] + 4 u^(1,1)[x, y] + 4 u^(2,0)[x, y] == 0

Out[19]= {{u -> Function[{x, y},
          c1[-x/2 + y] + c2[-x/2 - y]]}}

Out[20]= {True}
```

Conclusion

The solution of homogeneous second order linear partial differential equation with constant coefficients can be find out.

Exercise Problem

Write the differential equation $1\frac{\partial^2 u}{\partial x^2} + 2\frac{\partial^2 u}{\partial x \partial y} + 1\frac{\partial^2 u}{\partial y^2} = 0$ in Mathematica. Also find the solution of the given equation.

Practical No. 16

Aim

To solve the wave equation using Mathematica.

Problem

Write the differential equation $\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = 0$ in Mathematica. Also, find the solution of the given equation.

Theory

The second-order linear partial differential equation is given by

$$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2} + d\frac{\partial u}{\partial x} + e\frac{\partial u}{\partial y} + fu = g.$$

where, $u = u(x, y)$ and a,b,c,d,e,f,g are functions of x and y only. If g=0, then the equation becomes homogeneous. The solution of second order partial differential equation possible in Mathematica by command DSolve if principle part, i.e.$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2}$ have constant coefficients and the non principle part is vanishing, i.e.0.

Algorithm

Step1: Write the partial differential equation taking D[u[x,y], {x,2}] as $\frac{\partial^2 u}{\partial x^2}$ and D[u[x,y], {y,2}] as $\frac{\partial^2 u}{\partial y^2}$;
Step2: Click the run command button to see the given differential equation in Mathematica;
Step3: The partial derivatives $\frac{\partial^2 u}{\partial x^2}$ and $\frac{\partial^2 u}{\partial y^2}$ looks like $u^{(2,0)}$ and $u^{(0,2)}$. **Step4:** Write the program to solve the above equation. The command for partial differential equation in Mathematica is DSolve;
Step5: Write solution of the equation as DSolve[equation,dependent variable,{independent variables}];
Step6: Click the run command button to see the solution of the given differential equation.

Program

The input and the output of the program is as following:

```
In[1]:= WaveEquation = D[u[x, t], {x, 2}] - D[u[x, t], {t, 2}] == 0;
        DSolve[WaveEquation, u[x, t], {t, x}]

Out[1]= {{u[x, t] -> c1[-t + x] + c2[t + x]}}
```

Conclusion

The solution of homogeneous second order linear partial differential equation with constant coefficients can be find out.

Exercise Problem

Write the differential equation $\frac{\partial^2 u}{\partial x^2} - 2\frac{\partial^2 u}{\partial y^2} = 0$ in Mathematica. Also find the solution of the given equation.

Practical No. 17

Aim

To solve the initial Boundary value problem for wave equation using Mathematica.

Problem

Write the differential equation $9\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = 0$ in Mathematica. Also find and plot the solution of the given equation with initial boundary condition

$$u(0,t) = 0, u(50,t) = 0, u(x,0) = 3sin(2\pi x), \frac{\partial u(x,0)}{\partial y} = 5sin(3\pi x).$$

Theory

The second order linear partial differential equation is given by

$$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2} + d\frac{\partial u}{\partial x} + e\frac{\partial u}{\partial y} + fu = g.$$

where, $u = u(x,y)$ and a,b,c,d,e,f,g are functions of x and y only. If g=0, then the equation becomes homogeneous. The solution of second order partial differential equation possible in Mathematica by command DSolve if principle part, i.e.$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2}$ have constant coefficients and the non principle part is vanishing, i.e.0.

Algorithm

Step1: Write the partial differential equation taking D[u[x,y], {x,2}] as $\frac{\partial^2 u}{\partial x^2}$ and D[u[x,y], {y,2}] as $\frac{\partial^2 u}{\partial y^2}$;
Step2: Click the run command button to see the given differential equation in Mathematica;
Step3: The partial derivatives $\frac{\partial^2 u}{\partial x^2}$ and $\frac{\partial^2 u}{\partial y^2}$ looks like $u^{(2,0)}$ and $u^{(0,2)}$;
Step4: Write the program to solve the above equation. The command for partial differential equation in Mathematica is DSolve;
Step5: Write solution of the equation as DSolve[equation,dependent variable,{independent variables}];
Step6: Click the run command button to see the solution of the given differential equation.

Program

The input and the output of the program is as following:

```
(* Initial boundary value problem for wave equation *)

eqn = D[u[x, t], {t, 2}] == 9 * D[u[x, t], {x, 2}];

f3[x_] := 3 * Sin[2 * Pi * x];
f4[x_] := 5 * Sin[3 * Pi * x];

waveEqSoln = u[x, t] /. DSolve[{eqn, u[0, t] == 0, u[50, t] == 0,
    u[x, 0] == f3[x], Derivative[0, 1][u][x, 0] == f4[x]},
   u[x, t], {x, t}] /. K[1] -> n

   3 Cos[6  t] Sin[2  x] + (5 Sin[9  t] Sin[3  x]) / (9 )
```

Conclusion

The solution of homogeneous second order linear partial differential equation with constant coefficients can be find out.

Exercise Problem

Write the differential equation $\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = -9.80665$ in Mathematica. Also find and plot the solution of the given equation with initial boundary condition

$$u(0,t) = 0, u(1,t) = 0, u(x,0) = 10x^2(1-x)^2, \frac{\partial u(x,0)}{\partial y} = 0.$$

Practical No. 18

Aim

To solve the wave equation with initial boundary conditions using Mathematica.

Problem

Write the differential equation $\frac{\partial^2 u}{\partial y^2} - 4\frac{\partial^2 u}{\partial x^2} = 0$ in Mathematica. Also find and plot the solution of the given equation with initial condition $u(0,y) = 0, u_y(x,0) = 0, u(x,0) = 10x^4$.

Theory

The second order linear partial differential equation is given by

$$a\frac{\partial^2 u}{\partial x^2}+b\frac{\partial^2 u}{\partial x\partial y}+c\frac{\partial^2 u}{\partial y^2}+d\frac{\partial u}{\partial x}+e\frac{\partial u}{\partial y}+fu=g.$$

where, $u=u(x,y)$ and a,b,c,d,e,f,g are functions of x and y only. If g=0, then the equation becomes homogeneous. The solution of second order partial differential equation possible in Mathematica by command DSolve if principle part, i.e.$a\frac{\partial^2 u}{\partial x^2}+b\frac{\partial^2 u}{\partial x\partial y}+c\frac{\partial^2 u}{\partial y^2}$ have constant coefficients and the non principle part is vanishing, i.e.0.

Algorithm

Step1: Write the partial differential equation taking D[u[x,y], {x,2}] as $\frac{\partial^2 u}{\partial x^2}$ and D[u[x,y], {y,2}] as $\frac{\partial^2 u}{\partial y^2}$;
Step2: Click the run command button to see the given differential equation in Mathematica;
Step3: The partial derivatives $\frac{\partial^2 u}{\partial x^2}$ and $\frac{\partial^2 u}{\partial y^2}$ looks like $u^{(2,0)}$ and $u^{(0,2)}$;
Step4: Write the program to solve the above equation. The command for partial differential equation in Mathematica is DSolve;
Step5: Write solution of the equation as DSolve[equation,dependent variable,{independent variables}];
Step6: Click the run command button to see the solution of the given differential equation.

Program

The input and the output of the program is as following:

```
In[14]:= weqn = D[u[x, t], {t, 2}] == 4 * D[u[x, t], {x, 2}];

In[15]:= sol = DSolveValue[
  {weqn, u[x, 0] == 10 * x^4, Derivative[0, 1][u][x, 0] == 0,
   u[0, t] == 0, u[x, t], {x, t}}
];

Out[14]= u^(0,2)[x, t] == 4 u^(2,0)[x, t]

Out[15]= Piecewise[{
   {1/2 (10 (-2 t + x)^4 + 10 (2 t + x)^4), x > 2 t >= 0},
   {1/2 (-10 (2 t - x)^4 + 10 (2 t + x)^4), 0 <= x <= 2 t}
   }, Indeterminate]
```

Conclusion

The solution of homogeneous second order linear partial differential equation with constant coefficients can be find out.

Exercise Problem

Write the differential equation $\frac{\partial^2 u}{\partial y^2} - 4\frac{\partial^2 u}{\partial x^2} = 0$ in Mathematica. Also find and plot the solution of the given equation with initial condition $u(0,y) = 0, u_y(x,0) = 0, u(x,0) = x^4$.

Practical No. 19

Aim

To solve the Vibrating string initial value problem using Mathematica.

Problem

Write the differential equation $\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} = 0$ in Mathematica. Also, find and plot the solution of the given equation with initial condition

$$u_t(x,0) = 0, u(x,0) = x^2(\pi - x), u(0,t) = u(\pi,t) = 0.$$

Theory

The second order linear partial differential equation is given by

$$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2} + d\frac{\partial u}{\partial x} + e\frac{\partial u}{\partial y} + fu = g.$$

where, $u = u(x,y)$ and a,b,c,d,e,f,g are functions of x and y only. If g=0, then the equation becomes homogeneous. The solution of second order partial differential equation possible in Mathematica by command DSolve if principle part, i.e.$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2}$ have constant coefficients and the non principle part is vanishing, i.e.0.

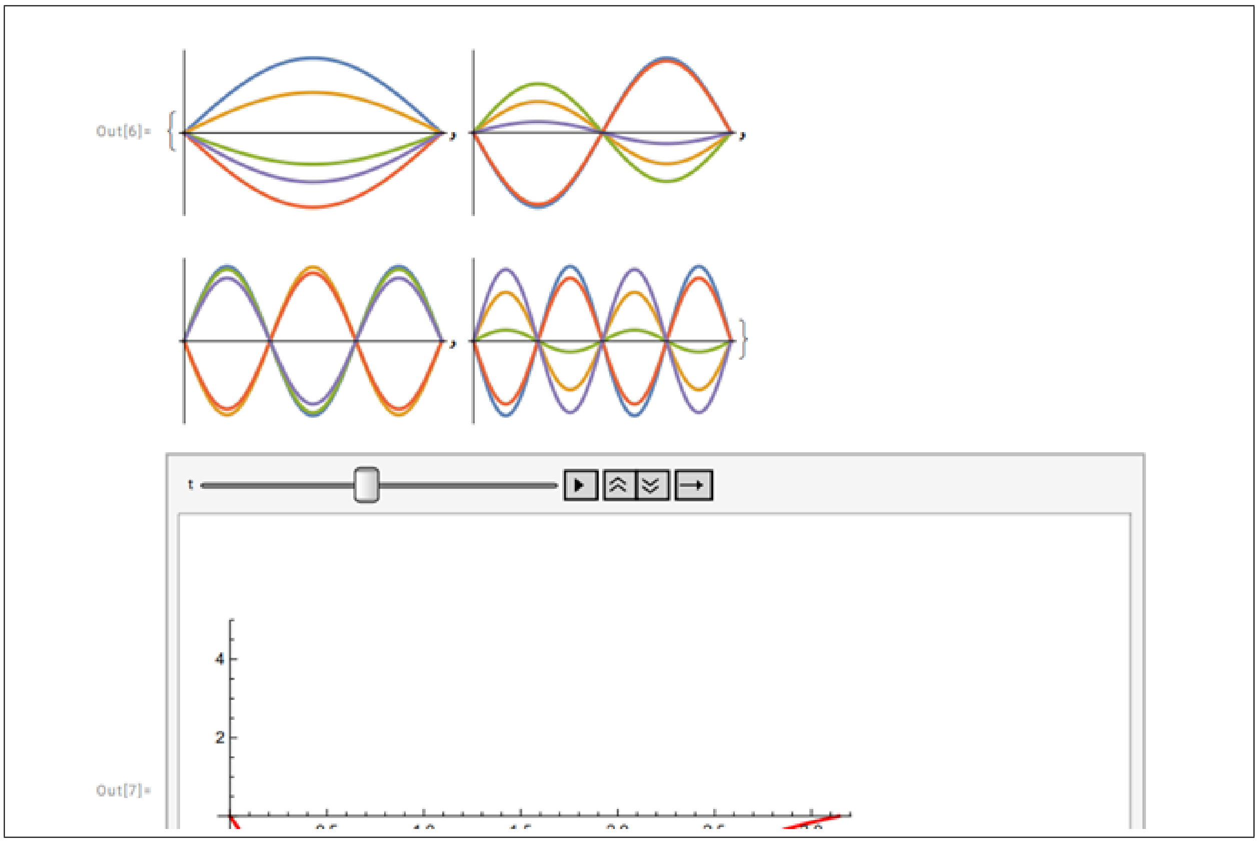
Algorithm

Step1: Write the partial differential equation taking D[u[x,y], {x,2}] as $\frac{\partial^2 u}{\partial x^2}$ and D[u[x,y], {y,2}] as $\frac{\partial^2 u}{\partial y^2}$;
Step2: Click the run command button to see the given differential equation in Mathematica;
Step3: The partial derivatives $\frac{\partial^2 u}{\partial x^2}$ and $\frac{\partial^2 u}{\partial y^2}$ looks like $u^{(2,0)}$ and $u^{(0,2)}$;
Step4: Write the program to solve the above equation. The command for partial differential equation in Mathematica is DSolve;
Step5: Write solution of the equation as DSolve[equation,dependent variable,{independent variables}];
Step6: Click the run command button to see the solution of the given differential equation.

Program

The input and the output of the program is as following:

```
In[1]:= weqn = D[u[x, t], {t, 2}] == D[u[x, t], {x, 2}];

In[2]:= bc = {u[0, t] == 0, u[, t] == 0};

In[3]:= ic = {u[x, 0] == x^2 ( - x), u[0,1][x, 0] == 0};

In[4]:= dsol = DSolve[{weqn, bc, ic}, u[x, t], {x, t}] /. {K[1] -> m};

In[5]:= asol[x_, t_] = u[x, t] /. dsol[[1]] // Activate;

In[6]:= Table[Show[
  Plot[Table[asol[x, t][[m]], {t, 0, 4}], {x, 0, }, Ticks -> False,
  ImageSize -> 150}], {m, 4}]

In[7]:= Animate[
  Plot[asol[x, t], {x, 0, }, PlotRange -> {-5, 5}, ImageSize -> Medium,
  PlotStyle -> Red], {t, 0, 2 }, SaveDefinitions -> True]

Out[4]= {{u -> Function[{x, t},
  Sum[4 (1 + 2 (-1)^m) Cos[t m] Sin[x m] / m^3, {m, 1, }]]}}

Out[5]= 4 Cos[t] Sin[x] - (3/2) Cos[2 t] Sin[2 x] + (4/27) Cos[3 t] Sin[3 x]
        - (3/16) Cos[4 t] Sin[4 x]
```



Conclusion

The solution of homogeneous second order linear partial differential equation with constant coefficients can be find out.

Exercise Problem

Write the differential equation $\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} = 0$ in Mathematica. Also find and plot the solution of the given equation with initial condition $u_t(x,0) = 0, u(x,0) = 4(\pi - x), u(0,t) = u(\pi,t) = 0$.

Practical No. 20

Aim

To solve the Cauchy problem for wave equation using Mathematica.

Problem

Write the differential equation $\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} = 0$ in Mathematica. Also find the solution of the given equation with Cauchy data $u(x,0) = e^{-x^2}, u_t(x,0) = 1$.

Theory

The second-order linear partial differential equation is given by

$$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2} + d\frac{\partial u}{\partial x} + e\frac{\partial u}{\partial y} + fu = g.$$

where, $u = u(x,y)$ and a,b,c,d,e,f,g are functions of x and y only. If g=0, then the equation becomes homogeneous. The solution of second order partial differential equation possible in Mathematica by command DSolve if principle part, i.e.$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2}$ have constant coefficients and the non principle part is vanishing, i.e.0.

Algorithm

Step1: Write the partial differential equation taking D[u[x,y], {x,2}] as $\frac{\partial^2 u}{\partial x^2}$ and D[u[x,y], {y,2}] as $\frac{\partial^2 u}{\partial y^2}$;
Step2: Click the run command button to see the given differential equation in Mathematica;
Step3: The partial derivatives $\frac{\partial^2 u}{\partial x^2}$ and $\frac{\partial^2 u}{\partial y^2}$ looks like $u^{(2,0)}$ and $u^{(0,2)}$;
Step4: Write the program to solve the above equation. The command for partial differential equation in Mathematica is DSolve;
Step5: Write solution of the equation as DSolve[equation,dependent variable,{independent variables}];
Step6: Click the run command button to see the solution of the given differential equation.

Program

The input and the output of the program is as following:

```
In[23]:= weqn = D[u[x, t], {t, 2}] == D[u[x, t], {x, 2}];

In[24]:= ic = {u[x, 0] == E^(-x^2), Derivative[0, 1][u][x, 0] == 1};

In[25]:= DSolveValue[{weqn, ic}, u[x, t], {x, t}]

Out[23]= u^(0,2)[x, t] == u^(2,0)[x, t]

Out[24]= {u[x, 0] == E^(-x^2), u^(0,1)[x, 0] == 1}
```

```
Out[25]= {
    (1/2) (E^(-(-t + x)^2) + E^(-(-t - x)^2)) + t,   t >= 0
    Indeterminate,   True
}
```

Conclusion

The solution of homogeneous second order linear partial differential equation with constant coefficients can be find out.

Exercise Problem

Write the differential equation $\frac{\partial^2 u}{\partial t^2} - 2\frac{\partial^2 u}{\partial x^2}$ in Mathematica. Also find the solution of the given equation.

Practical No. 21

Aim

To solve the Homogeneous wave equation using Mathematica.

Problem

Write the differential equation $\frac{\partial^2 u}{\partial t^2} - 3\frac{\partial^2 u}{\partial x^2} = 0$in Mathematica. Also, find the solution of the given equation.

Theory

The second order linear partial differential equation is given by

$$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2} + d\frac{\partial u}{\partial x} + e\frac{\partial u}{\partial y} + fu = g.$$

where, $u = u(x, y)$ and a,b,c,d,e,f,g are functions of x and y only. If g=0, then the equation becomes homogeneous. The solution of second order partial differential equation possible in Mathematica by command DSolve if principle part, i.e.$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2}$ have constant coefficients and the non principle part is vanishing, i.e.0.

Algorithm

Step1: Write the partial differential equation taking D[u[x,y], {x,2}] as $\frac{\partial^2 u}{\partial x^2}$ and D[u[x,y], {y,2}] as $\frac{\partial^2 u}{\partial y^2}$;
Step2: Click the run command button to see the given differential equation in Mathematica;
Step3: The partial derivatives $\frac{\partial^2 u}{\partial x^2}$ and $\frac{\partial^2 u}{\partial y^2}$ looks like $u^{(2,0)}$ and $u^{(0,2)}$;
Step4: Write the program to solve the above equation. The command for partial differential equation in Mathematica is DSolve;
Step5: Write solution of the equation as DSolve[equation,dependent variable,{independent variables}];
Step6: Click the run command button to see the solution of the given differential equation.

Program

The input and the output of the program is as following:

```
In[20]:= eqn = D[u[x, t], {t, 2}] == 3 D[u[x, t], {x, 2}];
          DSolveValue[eqn, u[x, t], {x, t}]

Out[20]= u^(0,2)[x, t] == 3 u^(2,0)[x, t]

Out[21]= c1[t - x/Sqrt[3]] + c2[t + x/Sqrt[3]]
```

Conclusion

The solution of homogeneous second order linear partial differential equation with constant coefficients can be find out.

Exercise Problem

Write the differential equation $\frac{\partial^2 u}{\partial t^2} - 2\frac{\partial^2 u}{\partial x^2} = 0$ in Mathematica. Also find the solution of the given equation.

Practical No. 22

Aim

To solve the Non homogeneous wave equation using Mathematica.

Problem

Write the differential equation $\frac{\partial^2 u}{\partial t^2} - 3\frac{\partial^2 u}{\partial x^2} = 1$ in Mathematica. Also find the solution of the given equation.

Theory

The second order linear partial differential equation is given by

$$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2} + d\frac{\partial u}{\partial x} + e\frac{\partial u}{\partial y} + fu = g.$$

where, $u = u(x, y)$ and a,b,c,d,e,f,g are functions of x and y only. If g=0, then the equation becomes homogeneous. The solution of second order partial differential equation possible in Mathematica by command DSolve if principle part, i.e.$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2}$ have constant coefficients and the non principle part is vanishing, i.e.0.

Algorithm

Step1: Write the partial differential equation taking D[u[x,y], {x,2}] as $\frac{\partial^2 u}{\partial x^2}$ and D[u[x,y], {y,2}] as $\frac{\partial^2 u}{\partial y^2}$;
Step2: Click the run command button to see the given differential equation in Mathematica;
Step3: The partial derivatives $\frac{\partial^2 u}{\partial x^2}$ and $\frac{\partial^2 u}{\partial y^2}$ looks like $u^{(2,0)}$ and $u^{(0,2)}$;
Step4: Write the program to solve the above equation. The command for partial differential equation in Mathematica is DSolve;
Step5: Write solution of the equation as DSolve[equation,dependent variable,{independent variables}];
Step6: Click the run command button to see the solution of the given differential equation.

Program

The input and the output of the program is as following:

```
In[22]:= eqn = D[u[x, t], {t, 2}] == 1 + 3 D[u[x, t], {x, 2}];
         DSolveValue[eqn, u[x, t], {x, t}]

Out[22]= u^(0,2)[x, t] == 1 + 3 u^(2,0)[x, t]

Out[23]= -x^2/6 + c1[t - x/Sqrt[3]] + c2[t + x/Sqrt[3]]
```

Conclusion

The solution of homogeneous second order linear partial differential equation with constant coefficients can be find out.

Exercise Problem

Write the differential equation $2\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial t^2} = 1$ in Mathematica. Also find the solution of the given equation.

Practical No. 23

Aim

To solve the one dimensional heat equation using Mathematica.

Problem

Write the differential equation $\frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} = 0$ in Mathematica. Also find the solution of the given equation.

Theory

The second order linear partial differential equation is given by

$$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2} + d\frac{\partial u}{\partial x} + e\frac{\partial u}{\partial y} + fu = g.$$

where, $u = u(x, y)$ and a,b,c,d,e,f,g are functions of x and y only. If g=0, then the equation becomes homogeneous. The solution of second order partial differential equation possible in Mathematica by command DSolve if principle part, i.e.$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2}$ have constant coefficients and the non principle part is vanishing, i.e.0.

Algorithm

Step1: Write the partial differential equation taking D[u[x,y], {x,2}] as $\frac{\partial^2 u}{\partial x^2}$ and D[u[x,y], {y,2}] as $\frac{\partial^2 u}{\partial y^2}$;
Step2: Click the run command button to see the given differential equation in Mathematica;
Step3: The partial derivatives $\frac{\partial^2 u}{\partial x^2}$ and $\frac{\partial^2 u}{\partial y^2}$ looks like $u^{(2,0)}$ and $u^{(0,2)}$;

Step4: Write the program to solve the above equation. The command for partial differential equation in Mathematica is DSolve;
Step5: Write solution of the equation as DSolve[equation,dependent variable,{independent variables}];
Step6: Click the run command button to see the solution of the given differential equation.

Program

```
The input and the output of the program is as following:
In[26]:= heqn = D[u[x, t], t] == D[u[x, t], {x, 2}];
         DSolveValue[heqn, u[x, t], {x, t}]

Out[26]= u^(0,1)[x, t] == u^(2,0)[x, t]

Out[27]= 1 + Cosh[c1 + x c2 + t c2] + Sinh[c1 + x c2 + t c2]
```

Conclusion

The solution of homogeneous second order linear partial differential equation with constant coefficients can be find out.

Exercise Problem

Write the differential equation $\frac{\partial u}{\partial t} - 2\frac{\partial^2 u}{\partial x^2} = 0$ in Mathematica. Also find the solution of the given equation.

Practical No. 24

Aim

To solve the Heat equation with constant boundary conditions using Mathematica.

Problem

Write the differential equation $\frac{\partial u}{\partial t} - 9\frac{\partial^2 u}{\partial x^2} = 0$ in Mathematica. Also find the solution of the given equation with initial condition $u(0,t) = 0, u(3,t) = 0, u(x,0) = 20$.

Theory

The second order linear partial differential equation is given by

$$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2} + d\frac{\partial u}{\partial x} + e\frac{\partial u}{\partial y} + fu = g.$$

where, $u = u(x, y)$ and a,b,c,d,e,f,g are functions of x and y only. If g=0, then the equation becomes homogeneous. The solution of second order partial differential equation possible in Mathematica by command DSolve if principle part, i.e.$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2}$ have constant coefficients and the non principle part is vanishing, i.e.0.

Algorithm

Step1: Write the partial differential equation taking D[u[x,y], {x,2}] as $\frac{\partial^2 u}{\partial x^2}$ and D[u[x,y], {y,2}] as $\frac{\partial^2 u}{\partial y^2}$;
Step2: Click the run command button to see the given differential equation in Mathematica;
Step3: The partial derivatives $\frac{\partial^2 u}{\partial x^2}$ and $\frac{\partial^2 u}{\partial y^2}$ looks like $u^{(2,0)}$ and $u^{(0,2)}$;
Step4: Write the program to solve the above equation. The command for partial differential equation in Mathematica is DSolve;
Step5: Write solution of the equation as DSolve[equation,dependent variable,{independent variables}];
Step6: Click the run command button to see the solution of the given differential equation.

Program

The input and the output of the program is as following:

```
In[56]:= Heqn = D[u[x, t], t] == 9 D[u[x, t], {x, 2}];
         heatEqSoln = u[x, t] /.
         DSolve[{Heqn, u[0, t] == 0, u[3, t] == 0, u[x, 0] == 20},
                u[x, t], {x, t}][[1]] /. K[1] -> n

Out[56]= u^(0,1)[x, t] == 9 u^(2,0)[x, t]

Out[57]=
40 Sum[(-1 + (-1)^n) E^(-n^2 ^2 t) Sin[(n  x)/3]/(n ), {n, 1, }]
```

Conclusion

The solution of homogeneous second order linear partial differential equation with constant coefficients can be find out.

Exercise Problem

Write the differential equation $\frac{\partial u}{\partial t} - 9\frac{\partial^2 u}{\partial x^2} = 0$ in Mathematica. Also find the solution of the given equation with initial condition $u(0,t) = 0, u(2,t) = 0, u(x,0) = 10$.

Practical No. 25

Aim

To solve the Heat equation with Dirichlet boundary conditions using Mathematica.

Problem

Write the differential equation $\frac{\partial u}{\partial t} - 9\frac{\partial^2 u}{\partial x^2} = 0$ in Mathematica. Also find and plot the solution of the given equation with initial condition $u(0,t) = 0, u(3,t) = 0, u(x,0) = x^2 * (3 - x)$.

Theory

The second-order linear partial differential equation is given by

$$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2} + d\frac{\partial u}{\partial x} + e\frac{\partial u}{\partial y} + fu = g.$$

where, $u = u(x,y)$ and a,b,c,d,e,f,g are functions of x and y only. If g=0, then the equation becomes homogeneous. The solution of second order partial differential equation possible in Mathematica by command DSolve if principle part, i.e.$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2}$ have constant coefficients and the non principle part is vanishing, i.e.0.

Algorithm

Step1: Write the partial differential equation taking D[u[x,y], {x,2}] as $\frac{\partial^2 u}{\partial x^2}$ and D[u[x,y], {y,2}] as $\frac{\partial^2 u}{\partial y^2}$;
Step2: Click the run command button to see the given differential equation in Mathematica;
Step3: The partial derivatives $\frac{\partial^2 u}{\partial x^2}$ and $\frac{\partial^2 u}{\partial y^2}$ looks like $u^{(2,0)}$ and $u^{(0,2)}$;
Step4: Write the program to solve the above equation. The command for partial differential equation in Mathematica is DSolve;
Step5: Write solution of the equation as DSolve[equation,dependent variable,{independent variables}];
Step6: Click the run command button to see the solution of the given differential equation.

Program

The input and the output of the program is as following:

```
In[60]:= heatEq = D[u[x, t], t] == 9*D[u[x, t], {x, 2}];
          f3[x_] = x^2*(3 - x);
          heatEqSoln = u[x, t] /.
            DSolve[
              {heatEq, u[0, t] == 0, u[3, t] == 0, u[x, 0] == f3[x]},
              u[x, t], {x, t}][[1]] /. K[1] -> n

Out[60]= u^(0,1)[x, t] == 9 u^(2,0)[x, t]

Out[62]=
Sum[(108 (1 + 2 (-1)^n) E^(-n^2 ^2 t) Sin[(n  x)/3])/(n^3 ^3), {n, 1, }]
```

Conclusion

The solution of homogeneous second order linear partial differential equation with constant coefficients can be find out.

Exercise Problem

Write the differential equation $\frac{\partial u}{\partial t} - 9\frac{\partial^2 u}{\partial x^2} = 0$ in Mathematica. Also find and plot the solution of the given equation with initial condition $u(0,t) = 1, u(5,t) = 0, u(x,0) = Sinx$.

Practical No. 26

Aim

To solve the Heat equation with Neumann boundary conditions using Mathematica.

Problem

Write the differential equation $4\frac{\partial u}{\partial y} - \frac{\partial 2u}{\partial x^2} = 0$ in Mathematica. Also find and plot the solution of the given equation with initial condition $u_x(0,x) = 0, u_x(10,x) = 0, u(x,0) = x^2 * (15-x)$.

Theory

The second order linear partial differential equation is given by

$$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2} + d\frac{\partial u}{\partial x} + e\frac{\partial u}{\partial y} + fu = g.$$

where, $u = u(x, y)$ and a,b,c,d,e,f,g are functions of x and y only. If g=0, then the equation becomes homogeneous. The solution of second order partial differential equation possible in Mathematica by command DSolve if principle part, i.e.$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2}$ have constant coefficients and the non principle part is vanishing, i.e.0.

Algorithm

Step1: Write the partial differential equation taking D[u[x,y], {x,2}] as $\frac{\partial^2 u}{\partial x^2}$ and D[u[x,y], {y,2}] as $\frac{\partial^2 u}{\partial y^2}$;
Step2: Click the run command button to see the given differential equation in Mathematica;
Step3: The partial derivatives $\frac{\partial^2 u}{\partial x^2}$ and $\frac{\partial^2 u}{\partial y^2}$ looks like $u^{(2,0)}$ and $u^{(0,2)}$;
Step4: Write the program to solve the above equation. The command for partial differential equation in Mathematica is DSolve;
Step5: Write solution of the equation as DSolve[equation,dependent variable,{independent variables}];
Step6: Click the run command button to see the solution of the given differential equation.

Program

The input and the output of the program is as following:

```
In[63]:= heatEq = 4*D[u[x, t], t] == D[u[x, t], {x, 2}];
          f3[x_] = x^2*(15 - x);
          heatEqSoln = u[x, t] /.
            DSolve[
              {heatEq, Derivative[1, 0][u][0, t] == 0,
               Derivative[1, 0][u][10, t] == 0,
               u[x, 0] == f3[x]},
              u[x, t], {x, t}][[1]] /. K[1] -> n

Out[63]= 4 u^(0,1)[x, t] == u^(2,0)[x, t]

Out[65]=
250 * (1/5) *
  Sum[(60000 (-1 + (-1)^n) E^(-n^2 ^2 t/250) Cos[(n  x)/10])/(n^4 ^4), {n, 1, }]
```

Conclusion

The solution of homogeneous second order linear partial differential equation with constant coefficients can be find out.

Exercise Problem

Write the differential equation $4\frac{\partial u}{\partial y} - \frac{\partial^2 u}{\partial x^2} = 0$ in Mathematica. Also find and plot the solution of the given equation with initial condition $u_x(0,x) = 0, u_x(10,x) = 0, u(x,0) = x^4 * (15 - x)$.

Practical No. 27

Aim

To solve the Heat equation with mixed boundary conditions using Mathematica.

Problem

Write the differential equation $\frac{\partial u}{\partial t} - 9\frac{\partial^2 u}{\partial x^2} = 0$ in Mathematica. Also find and plot the solution of the given equation with initial condition $u(0,t) = 0, u_t(3,t) = 0, u(x,0) = x * (6 - x)$.

Theory

The second-order linear partial differential equation is given by

$$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2} + d\frac{\partial u}{\partial x} + e\frac{\partial u}{\partial y} + fu = g.$$

where, $u = u(x, y)$ and a,b,c,d,e,f,g are functions of x and y only. If g=0, then the equation becomes homogeneous. The solution of second order partial differential equation possible in Mathematica by command DSolve if principle part, i.e.$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2}$ have constant coefficients and the non principle part is vanishing, i.e.0.

Algorithm

Step1: Write the partial differential equation taking D[u[x,y], {x,2}] as $\frac{\partial^2 u}{\partial x^2}$ and D[u[x,y], {y,2}] as $\frac{\partial^2 u}{\partial y^2}$;
Step2: Click the run command button to see the given differential equation in Mathematica;
Step3: The partial derivatives $\frac{\partial^2 u}{\partial x^2}$ and $\frac{\partial^2 u}{\partial y^2}$ looks like $u^{(2,0)}$ and $u^{(0,2)}$;
Step4: Write the program to solve the above equation. The command for partial differential equation in Mathematica is DSolve;
Step5: Write solution of the equation as DSolve[equation,dependent variable,{independent variables}];
Step6: Click the run command button to see the solution of the given differential equation.

Program

The input and the output of the program is as following:

```
In[63]:= heatEq = 4*D[u[x, t], t] == D[u[x, t], {x, 2}];
          f3[x_] = x^2*(15 - x);
          heatEqSoln = u[x, t] /.
            DSolve[
              {heatEq, Derivative[1, 0][u][0, t] == 0,
               Derivative[1, 0][u][10, t] == 0,
               u[x, 0] == f3[x]},
              u[x, t], {x, t}][[1]] /. K[1] -> n

Out[63]= 4 u^(0,1)[x, t] == u^(2,0)[x, t]

Out[65]=
250 * (1/5) *
  Sum[(60000 (-1 + (-1)^n) E^(-n^2 ^2 t/250) Cos[(n  x)/10])/(n^4 ^4), {n, 1, }]
```

Conclusion

The solution of homogeneous second order linear partial differential equation with constant coefficients can be find out.

Exercise Problem

Write the differential equation $\frac{\partial u}{\partial t} - 9\frac{\partial^2 u}{\partial x^2} = 0$ in Mathematica. Also find and plot the solution of the given equation with initial condition $u(0,t) = 0, u_t(1,t) = 0, u(x,0) = x * (10 - x)$.

Chapter 7

Ring Theory

Practical No. 1

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program which takes input of an integer n greater than 1. Then it prints the additive and multiplicative inverses of each element of the ring $Z_n[i]$. Run the program for $n = 5$.

Theory

The additive inverse z_2 of $z_1 \in Z_n[i]$ is defined as

$$z_1 + z_2 = 0,$$

where addition and multiplication are to be performed in modulo n arithmetic. The additive inverse z_2 of z_1 is denoted by $-z_1$. Similarly, the multiplicative inverse z_3 of non-zero $z_1 \in Z_n[i]$ is defined as

$$z_1 z_3 = 1,$$

The multiplicative inverse z_3 of z_1 is denoted by z^{-1}.

Algorithm

Step 1: Prompt user to give input of integer n greater than 1;

Step 2: Check if n is a positive integer greater than 1. If not, show an error and exit;

Step 3: Define a list *zn* and save elements of Z_n in it;

Step 4: Define a list *zni* and save elements of $Z_n[i]$ in it;

Step 5: Define a list *inverses* to save element, additive inverse and multiplicative inverse triplet;

Step 6: Run a loop for each z_1 in $Z_n[i]$;

Step 7: Define *additive Inverse* and *multiplicativeInverse* to save inverse values;

Step 8: Run an inner loop for each z_2 in $Z_n[i]$;

Step 9: At each iteration, if $z_1 + z_2 = 0$, save z_2 to *additiveInverse*;

Step 10: Also, if $z_1 z_2 = 1$, save z_2 to *multiplicativeInverse.* If both inverses found, exit the inner loop;

Step 11: Save *(element, additiveInverse)* pair or *(element, additiveInverse, multiplicativeInverse)* triplet in list inverses;

Step 12: Exit the outer loop;

Step 13: Print the inverses of elements of $Z_n[i]$.

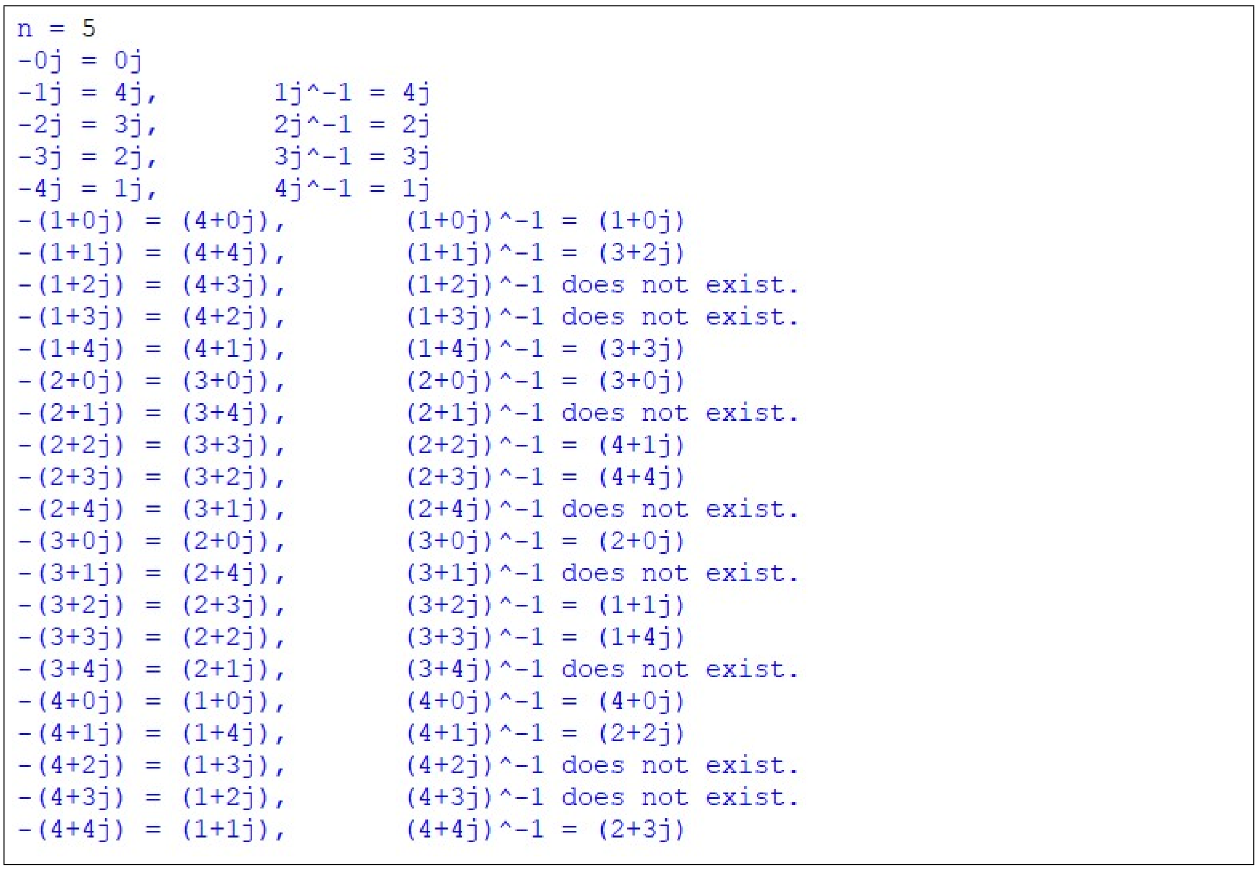
Program

The following PYTHON script implements the algorithm.

```
#Step 1
n = int(input('n = '))
#Step 2
if n <= 1:
    raise ValueError('n must be an integer greater than 1.')
#Step 3
zn = list(range(n))
#Step 4
zni = [complex(a, b) for a in zn for b in zn]
#Step 5
inverses = []
#Step 6
for z1 in zni:
    #Step 7
    additiveInverse = None
    multiplicativeInverse = None

    #Step 8
    for z2 in zni:
        #Step 9
        summation = z1 + z2
        result = complex(summation.real % n, summation.imag % n)
        if result == 0:
            additiveInverse = z2

        #Step 10
        if z1 != 0:
            product = z1*z2
            result = complex(product.real % n, product.imag % n)
            if result == 1:
                multiplicativeInverse = z2

        #Step 11
        if (additiveInverse is not None) and (multiplicativeInverse is not None):
            break;

    #Step 12
    if(multiplicativeInverse is not None):
        inverses.append((z1, additiveInverse, multiplicativeInverse))
    else:
        inverses.append((z1, additiveInverse))

#Step 13
for elem in inverses:
    if(len(elem) == 3):
        z1, z2, z3 = elem
        print(f‘-{z1} = {z2}, \t{z1}^-1 = {z3}’)
    else:
        z1, z2 = elem
        if z1== 0:
            print(f‘-{z1} = {z2}’)
        else:
            print(f‘-{z1} = {z2}, \t{z1}^-1 does not exist.’)
```

Output

The output of the program for various inputs is as follows:



```
n = 5
-0j = 0j
-1j = 4j,       1j^-1 = 4j
-2j = 3j,       2j^-1 = 2j
-3j = 2j,       3j^-1 = 3j
-4j = 1j,       4j^-1 = 1j
-(1+0j) = (4+0j),       (1+0j)^-1 = (1+0j)
-(1+1j) = (4+4j),       (1+1j)^-1 = (3+2j)
-(1+2j) = (4+3j),       (1+2j)^-1 does not exist.
-(1+3j) = (4+2j),       (1+3j)^-1 does not exist.
-(1+4j) = (4+1j),       (1+4j)^-1 = (3+3j)
-(2+0j) = (3+0j),       (2+0j)^-1 = (3+0j)
-(2+1j) = (3+4j),       (2+1j)^-1 does not exist.
-(2+2j) = (3+3j),       (2+2j)^-1 = (4+1j)
-(2+3j) = (3+2j),       (2+3j)^-1 = (4+4j)
-(2+4j) = (3+1j),       (2+4j)^-1 does not exist.
-(3+0j) = (2+0j),       (3+0j)^-1 = (2+0j)
-(3+1j) = (2+4j),       (3+1j)^-1 does not exist.
-(3+2j) = (2+3j),       (3+2j)^-1 = (1+1j)
-(3+3j) = (2+2j),       (3+3j)^-1 = (1+4j)
-(3+4j) = (2+1j),       (3+4j)^-1 does not exist.
-(4+0j) = (1+0j),       (4+0j)^-1 = (4+0j)
-(4+1j) = (1+4j),       (4+1j)^-1 = (2+2j)
-(4+2j) = (1+3j),       (4+2j)^-1 does not exist.
-(4+3j) = (1+2j),       (4+3j)^-1 does not exist.
-(4+4j) = (1+1j),       (4+4j)^-1 = (2+3j)
```

Exercise Problem

Write a program which takes input of an integer n greater than 1. Then it prints the additive and multiplicative inverses of each element of the ring Z_n. Run the program for $n = 2, 4, 5, 8$.

Practical No. 2

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program which takes input of an integer n greater than 1. Then it prints all the zero divisors of the ring Z_n. If there are no zero divisors, print the same. Run the program for $n = 3, 8, 17, 30$.

Theory

A non-zero integer $k \in Z_n$ is called a zero divisor, if there exists a non-zero integer $m \in Z_n$ such that

$$km \bmod n = 0.$$

Algorithm

```
Step 1: Prompt user to give input of integer $n$ greater than 1;
Step 2: Check if $n$ is a positive integer greater than 1. If not, show an error and exit;
Step 3: Define a list $zn$ and save elements of $Z_n$ in it;
Step 4: Define a list $zeroDivisors$ to save zero divisors;
Step 5: Run a loop for each $k$ in $Z_n$;
Step 6: If $k$ is zero, then continue to next iteration;
Step 7: Otherwise, run an inner loop for each $m$ in $Z_n$;
Step 8: At each iteration of inner loop, if $m$ is not zero and $km \bmod n = 0$, save $k$ to list
        $zeroDivisors$ and exit the inner loop;
Step 9: Exit the outer loop;
Step 10: Print the zero divisors of elements of $Z_n$.
```

Program

The following PYTHON script implements the algorithm.

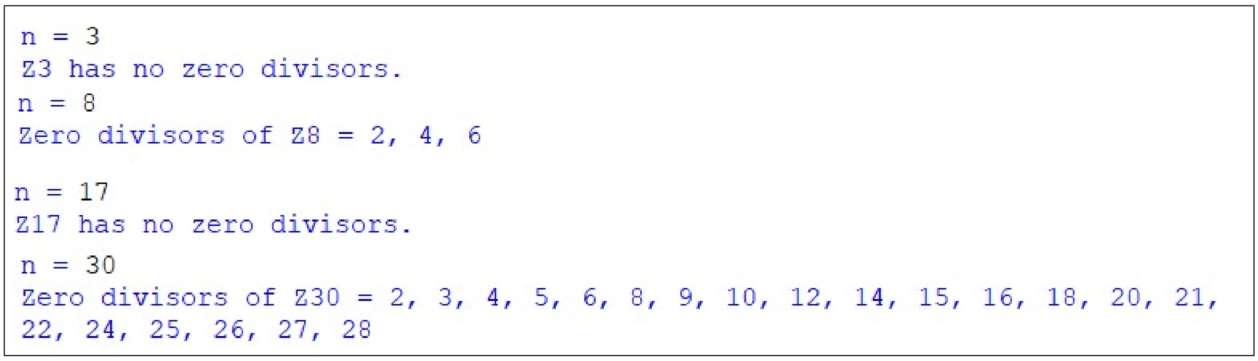
```
#Step 1
n = int(input('n = '))
#Step 2
if n <= 1:
    raise ValueError('n must be an integer greater than 1.')
#Step 3
zn = list(range(n))
#Step 4
zeroDivisors = []
#Step 5
for k in zn:
    #Step 6
    if k == 0:
        continue

    #Step 7
    for m in zn:
        #Step 8
        if m != 0 and k*m%n == 0:
            zeroDivisors.append(k)
            break

#Step 10
if len(zeroDivisors) == 0:
    print(f'Z{n} has no zero divisors.')
else:
    zeroDivisorsStr = ', '.join([str(x) for x in zeroDivisors ])
    print(f'Zero divisors of Z{n} = {zeroDivisorsStr}')
```

Output

The output of the program for various inputs is as follows:

Exercise Problem

Write a program which takes input of an integer n greater than 1. Then it prints all the zero divisors of the ring $Z_n[i]$. If there are no zero divisors, print the same. Run the program for $n = 3, 5$.



```
n = 3
Z3 has no zero divisors.
n = 8
Zero divisors of Z8 = 2, 4, 6
n = 17
Z17 has no zero divisors.
n = 30
Zero divisors of Z30 = 2, 3, 4, 5, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20, 21,
22, 24, 25, 26, 27, 28
```

Practical No. 3

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program which takes input of an integer n greater than 1. Then it prints all the units of the ring $Z_n[i]$. If there are no units, print the same. Run the program for $n = 5, 7$.

Theory

A non-zero number $z \in Z_n[i]$ is called a unit, if it has a multiplicative inverse.

Algorithm

```
Step 1: Prompt user to give input of integer $n$ greater than 1;
Step 2: Check if $n$ is a positive integer greater than 1. If not, show an error and exit;
Step 3: Define a list zn and save elements of $Z_n$ in it;
Step 4: Define a list zni and save elements of $Z_n[i]$ in it;
Step 5: Define a list units to save units;
Step 6: Run a loop for each $z_1$ in $Z_n$;
Step 7: If $z_1$ is zero, then continue to next iteration;
Step 8: Otherwise, run an inner loop for each $z_2$ in $Z_n[i]$;
Step 9: At each iteration of inner loop, if $z_2$ is not zero and $z_1 z_2 = 1$, save $z_1$ to list units
        and exit the inner loop;
Step 10: Exit the outer loop;
Step 11: Print the units of elements of $Z_n[i]$.
```

Program

The following PYTHON script implements the algorithm.

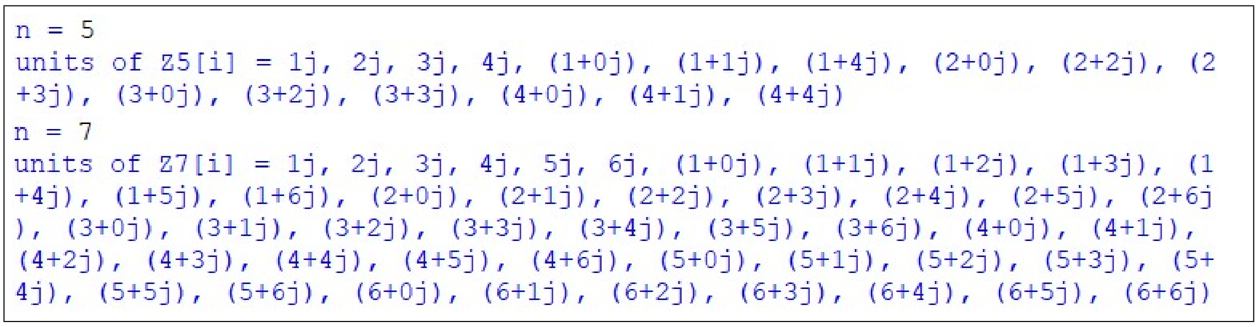
```
#Step 1
n = int(input('n = '))
#Step 2
if n <= 1:
    raise ValueError('n must be an integer greater than 1.')
#Step 3
zn = list(range(n))
#Step 4
zni = [complex(a, b) for a in zn for b in zn]
#Step 5
units = []
#Step 6
for z1 in zni:
    #Step 7
    if z1 == 0:
        continue
    #Step 8
    else:
        for z2 in zni:
         #Step 9
            product = z1*z2
            result = complex(product.real % n, product.imag % n)
            if result == 1:
                units. append(z1)
                break

#Step 11
if len(units) == 0:
    print(f'Z{n}[i] has no units.')
else:
    unitsStr = ', '.join([str(x) for x in units ])
    print(f'units of Z{n}[i] = {unitsStr}')
```

Output

The output of the program for various inputs is as follows:

Exercise Problem

Write a program which takes input of an integer n greater than 1. Then it prints all the units of the ring Z_n. If there are no units, print the same. Run the program for $n = 5, 8, 15, 47$.



```
n = 5
units of Z5[i] = 1j, 2j, 3j, 4j, (1+0j), (1+1j), (1+4j), (2+0j), (2+2j), (2
+3j), (3+0j), (3+2j), (3+3j), (4+0j), (4+1j), (4+4j)
n = 7
units of Z7[i] = 1j, 2j, 3j, 4j, 5j, 6j, (1+0j), (1+1j), (1+2j), (1+3j), (1
+4j), (1+5j), (1+6j), (2+0j), (2+1j), (2+2j), (2+3j), (2+4j), (2+5j), (2+6j
), (3+0j), (3+1j), (3+2j), (3+3j), (3+4j), (3+5j), (3+6j), (4+0j), (4+1j),
(4+2j), (4+3j), (4+4j), (4+5j), (4+6j), (5+0j), (5+1j), (5+2j), (5+3j), (5+
4j), (5+5j), (5+6j), (6+0j), (6+1j), (6+2j), (6+3j), (6+4j), (6+5j), (6+6j)
```

Practical No. 4

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program which takes input of an integer n greater than 1. Then it prints all the characteristic of the ring Z_n using definition of characteristic. Run the program for $n = 3, 8, 17, 30$.

Theory

The characteristic of a ring R is the least positive integer k such that $kx = 0$ for all x in R. If no such integer exists, we say that R has characteristic 0.

Algorithm

Step 1: Prompt user to give input of integer n greater than 1;

Step 2: Check if n is a positive integer greater than 1. If not, show an error and exit;

Step 3: Define a list *zn* and save elements of Z_n in it;

Step 4: Define characteristic variable *charZn* and initialize it to 1;

Step 5: Run an infinite while loop;

Step 6: Define another variable *charFound* and initialize it to *True*;

Step 7: At each iteration, run an inner loop for each x in Z_n;

Step 8: At each iteration of inner loop, if $\text{char}\,(Z_n)\,x \bmod n \neq 0$, increase *charZn*, set *charFound* to *False* and exit inner loop;

Step 9: If *charFound* is true, exit the outer loop;

Step 10: Print the characteristic of Z_n.

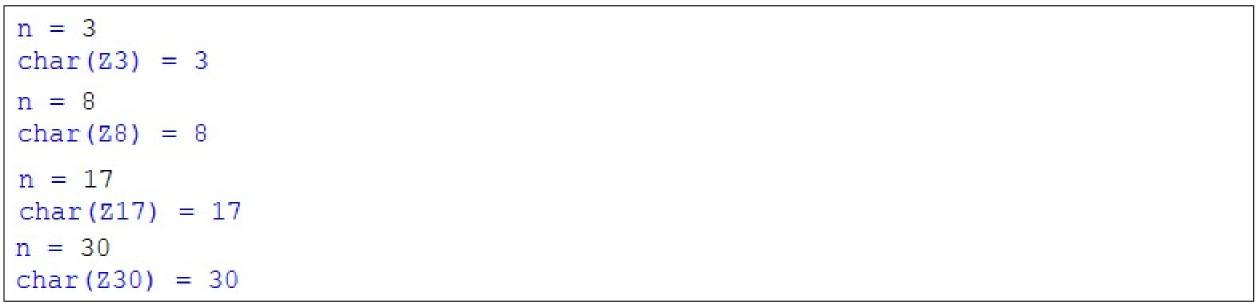
Program

The following PYTHON script implements the algorithm.

```
#Step 1
n = int(input('n = '))
#Step 2
if n <= 1:
    raise ValueError('n must be an integer greater than 1.')
#Step 3
zn = list(range(n))
#Step 4
charZn = 1

#Step 5
while True:
    #Step 6
    charFound = True

    #Step 7
    for x in zn:
        #Step 8
        if charZn*x%n != 0:
            charZn = charZn + 1
            charFound = False
            break
     #Step 9
    if charFound:
        break

#Step 11
print(f'char(Z{n}) = {charZn}')
```

Output

The output of the program for various inputs is as follows:



```
n = 3
char(Z3) = 3
n = 8
char(Z8) = 8
n = 17
char(Z17) = 17
n = 30
char(Z30) = 30
```

Exercise Problem

Write a program which takes input of an integer n greater than 1. Then it prints all the characteristic of the ring $Z_n[i]$ using definition of characteristic. Run the program for $n = 5, 12, 25, 60$.

Practical No. 5

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program which takes input of an integer n greater than 1. It also takes input of a positive integer m. Then, it prints the order of the subring mZ_n. Run the program for $(n, m) = (6, 2), (12, 5), (15, 9)$.

Theory

For any $m \in Z_n$ the subring mZ_n of the ring Z_n is defined as

$$mZ_n = \{mk \bmod n : \; k \in Z_n\}.$$

The order of a subring is same as its cardinality.

Algorithm

Step 1: Prompt user to give input of integer n greater than 1;

Step 2: Check if n is a positive integer greater than 1. If not, show an error and exit;

Step 3: Prompt user to give input of a positive integer n;

Step 4: Check if m is a positive integer. If not, show an error and exit;

Step 5: Define a list *mzn* and save elements of mZ_n in it;

Step 6: Convert the list *mzn* to a set to remove duplicate elements in the list *mzn*;

Step 7: Convert *mzn* back to a list;

Step 8: Print the order of mZ_n.

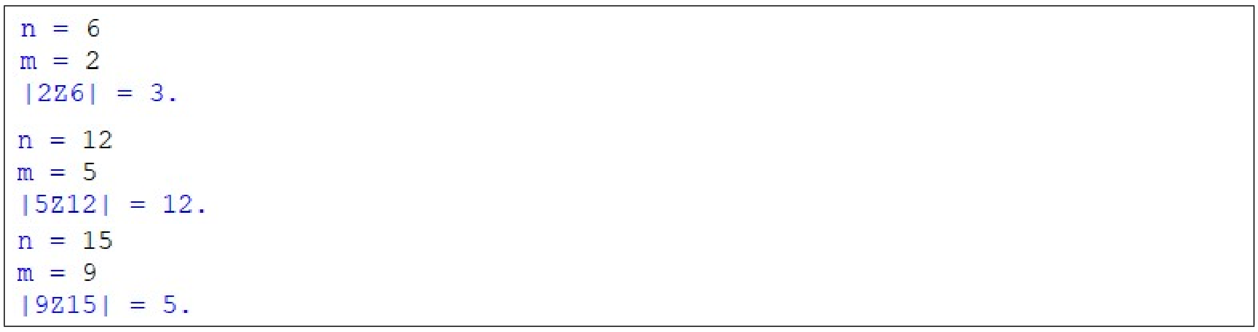
Program

The following PYTHON script implements the algorithm.

```
#Step 1
n = int(input('n = '))
#Step 2
if n <= 1:
    raise ValueError('n must be an integer greater than 1.')
#Step 3
m = int(input('m = '))
#Step 4
if m <= 0:
    raise ValueError('m must be positive integer.')
#Step 5
mzn = [m*k%n for k in range(n)]
#Step 6
mzn = set(mzn)
#Step 7
mzn = list(mzn)
#Step 8
print(f'|{m}Z{n}| = {len(mzn)}.')
```

Output

The output of the program for various inputs is as follows:



```
n = 6
m = 2
|2Z6| = 3.
n = 12
m = 5
|5Z12| = 12.
n = 15
m = 9
|9Z15| = 5.
```

Exercise Problem

Write a program which takes input of an integer n greater than 1. It also takes input of a positive integer m. Then, it prints the multiplicative identity of mZ_n as a subring of of Z_n. If multiplicative identity does not exist, it prints suitable message. Run the program for $(n, m) = (6, 2), (12, 5), (15, 9)$.

Practical No. 6

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program which takes input of an integer n greater than 1. Then it prints if $Z_n[i]$ is an integral domain or not. Run the program for $n = 4, 7, 13, 19$.

Theory

A commutative ring with unity is called an integral domain if there are no zero divisors in it.

Algorithm

```
Step 1: Prompt user to give input of integer $n$ greater than 1;
Step 2: Check if $n$ is a positive integer greater than 1. If not, show an error and exit;
Step 3: Define a list $zn$ and save elements of $Z_n$ in it;
Step 4: Define a list $zni$ and save elements of $Z_n[i]$ in it;
Step 5: Define a variable integralDomain and set it to True;
Step 6: Run a loop for each $z_1$ in $Z_n[i]$;
Step 7: If $z_1$ is zero, then continue to next iteration;
Step 8: Otherwise, run an inner loop for each $z_2$ in $Z_n[i]$;
Step 9: If $z_2$ is zero, then continue to next iteration of inner loop;
Step 10: Otherwise, if $z_1 z_2 = 0$, set variable integralDomain to false and exit the inner
         loop;
Step 11: If variable integralDomain is false, exit the outer loop;
Step 12: Print the result.
```

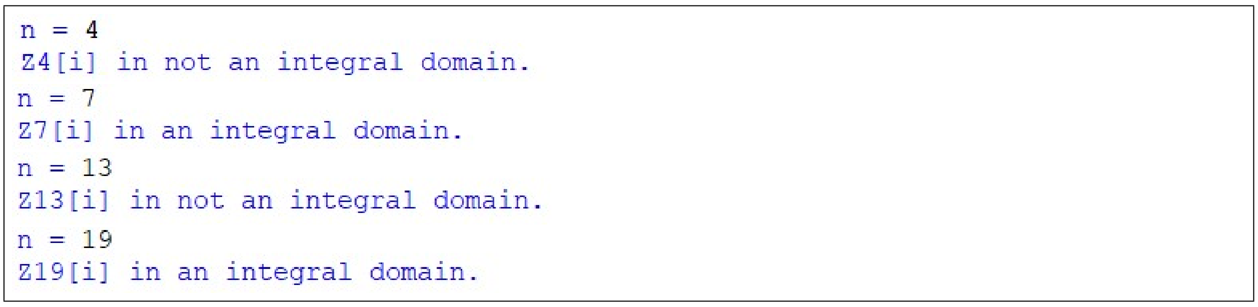
Program

The following PYTHON script implements the algorithm.

```
#Step 1
n = int(input('n = '))
#Step 2
if n <= 1:
    raise ValueError('n must be an integer greater than 1.')
#Step 3
zn = list(range(n))
#Step 4
zni = [complex(a, b) for a in zn for b in zn]
#Step 5
integralDomain = True
#Step 6
for z1 in zni:
    #Step 7
    if z1 == 0:
        continue

    #Step 8
    for z2 in zni:
        #Step 9
        if z2 == 0:
            continue
        #Step 10
        product = z1*z2
        result = complex(product.real % n, product.imag % n)
        if result == 0:
            integralDomain = False
            break
    #Step 11
    if not integralDomain:
         break

#Step 13
if integralDomain:
    print(f'Z{n}[i] in an integral domain.')
else:
    print(f'Z{n}[i] in not an integral domain.')
```

Output

The output of the program for various inputs is as follows:



```
n = 4
Z4[i] in not an integral domain.
n = 7
Z7[i] in an integral domain.
n = 13
Z13[i] in not an integral domain.
n = 19
Z19[i] in an integral domain.
```

Exercise Problem

Write a program which takes input of an integer n greater than 1. Then it prints if $Z_n[i]$ is a field or not. Run the program for $n = 4, 7, 13, 19$.

Practical No. 7

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program which takes input of an integer n greater than 1. Then it prints if $(x+y)^n = x^n+y^n$ holds for all x, y in the ring Z_n or not. Run the program for $n = 4, 7, 15, 29$.

Theory

In general $(x + y)^n = x^n + y^n$ does not hold. However if underlying ring is of prime characteristic, then the relation holds for all the elements of ring Z_n.

Algorithm

Step 1: Prompt user to give input of integer n greater than 1;

Step 2: Check if n is a positive integer greater than 1. If not, show an error and exit;

Step 3: Define a list zn and save elements of Z_n in it;

Step 4: Define a variable *holds* and set it to True;

Step 5: Run a loop for each x in Z_n;

Step 6: Run an inner loop for each y in Z_n;

Step 7: At each iteration of inner loop, if relation does not hold, set *holds* to False and exit inner loop;

Step 8: If relation does not hold, exit the outer loop;

Step 9: Print the result.

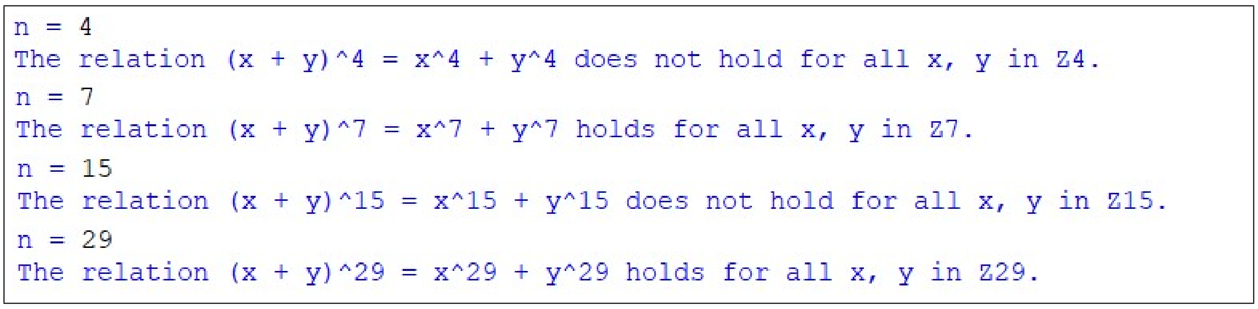
Program

The following PYTHON script implements the algorithm.

```
#Step 1
n = int(input('n = '))
#Step 2
if n <= 1:
    raise ValueError(‘n must be an integer greater than 1.’)
#Step 3
zn = list(range(n))
#Step 4
holds = True
#Step 5
for x in zn:
    #Step 6
    for y in zn:
        #Step 7
        if ((x + y)**n)%n != (x**n + y**n)%n:
            holds = False
            break
    #Step 8
    if not holds:
        break

#Step 9
if holds:
    print(f‘The relation (x + y)^{n} = x^{n} + y^{n} holds for all x, y in Z{n}.’)
else:
    print(f‘The relation (x + y)^{n} = x^{n} + y^{n} does not hold for all x, y in Z{n}.’)
```

Output

The output of the program for various inputs is as follows:



```
n = 4
The relation (x + y)^4 = x^4 + y^4 does not hold for all x, y in Z4.
n = 7
The relation (x + y)^7 = x^7 + y^7 holds for all x, y in Z7.
n = 15
The relation (x + y)^15 = x^15 + y^15 does not hold for all x, y in Z15.
n = 29
The relation (x + y)^29 = x^29 + y^29 holds for all x, y in Z29.
```

Exercise Problem

Write a program which takes input of an integer n greater than 1. It also takes input of a positive integer m. Then it prints if $(x+y)^{n^m} = x^{n^m} + y^{n^m}$ holds for all x, y in the ring Z_n or not. Run the program for $(n, m) = (4, 3), (7, 5), (15, 5), (29, 2)$.

Practical No. 8

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program which takes input of an integer n greater than 1 and two positive integers p and q. Then it finds and prints all the principal ideals $\langle a\rangle$ of Z_n such that $\langle p\rangle + \langle q\rangle = \langle n\rangle$. Run the program for $(n, p, q) = (20, 4, 10), (65, 10, 25)$.

Theory

An ideal I of a commutative ring R is called a principal ideal generated by a of R if

$$I = \{ra : r \in R\}.$$

I is denoted by $\langle a\rangle$.

Algorithm

```
Step 1: Prompt user to give input of integer $n$ greater than 1;
Step 2: Check if $n$ is a positive integer greater than 1. If not, show an error and exit;
Step 3: Prompt user to give input of positive integer $p$;
Step 4: Check if $p$ is a positive integer. If not, show an error and exit;
Step 5: Prompt user to give input of positive integer $q$;
Step 6: Check if $q$ is a positive integer. If not, show an error and exit;
Step 7: Define a list zn and save elements of $Z_n$ in it;
Step 8: Define a list pzn and save elements of principal ideal $\langle p\rangle$ in it;
Step 9: Define a list qzn and save elements of principal ideal $\langle q\rangle$ in it;
Step 10: Define a list pPlusqZn and save elements of subring $\langle p\rangle + \langle q\rangle$ in it;
Step 11: Define a list foundA to save generators;
Step 12: Run a loop for each $a$ in $Z_n$;
Step 13: Define a list azn and save elements of principal ideal $\langle a\rangle$ in it;
```

Step 14: If $\langle p\rangle + \langle q\rangle$ and $\langle a\rangle$ are same, save a in the list *foundA*;

Step 15: Print the zero divisors of elements of Z_n.

Program

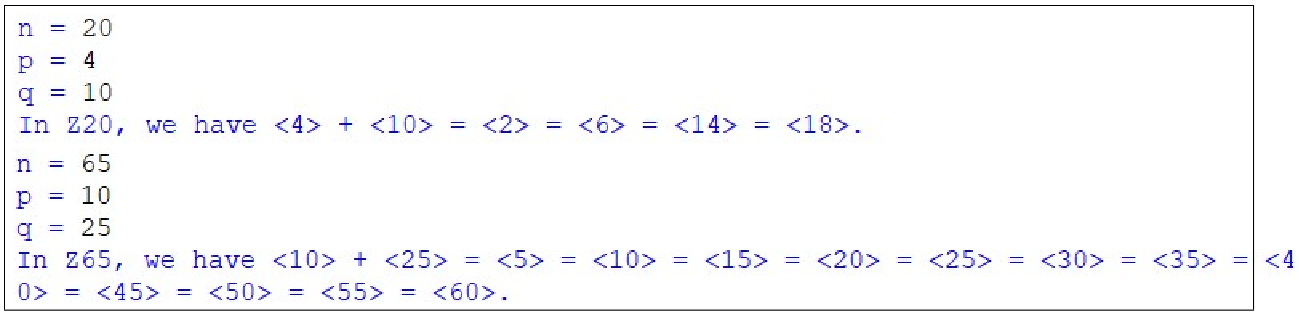
The following PYTHON script implements the algorithm.

```
#Step 1
n = int(input('n = '))
#Step 2
if n <= 1:
    raise ValueError('n must be an integer greater than 1.')

#Step 3
p = int(input('p = '))
#Step 4
if p <= 0:
    raise ValueError('p must be a positive integer.')

#Step 5
q = int(input('q = '))
#Step 6
if q <= 0:
    raise ValueError('q must be a positive integer.')

#Step 7
zn = list(range(n))

#Step 8
pzn = list(set([p*k%n for k in zn]))
#Step 9
qzn = list(set([q*k%n for k in zn]))
#Step 10
pPlusqZn = sorted(set([(pk + qk)%n for pk in pzn for qk in qzn]))
#Step 11
foundA = []

#Step 12
for a in zn:
    #Step 13
    azn = sorted(set([a*k%n for k in zn]))
    #Step 14
    if pPlusqZn == azn:
        foundA.append(a)
```

```
#Step 15
if len(foundA) == 0:
    print(f‘No a in Z{n} exists such that <p> + <q> = <a>.’)
else:
    aStr = ‘ = ’.join([f’<{str(a)}>’ for a in foundA])
    print(f‘In Z{n}, we have <{p}> + <{q}> = {aStr}.’)
```

Output

The output of the program for various inputs is as follows:



```
n = 20
p = 4
q = 10
In Z20, we have <4> + <10> = <2> = <6> = <14> = <18>.
n = 65
p = 10
q = 25
In Z65, we have <10> + <25> = <5> = <10> = <15> = <20> = <25> = <30> = <35> = <4
0> = <45> = <50> = <55> = <60>.
```

Exercise Problem

Write a program which takes input of an integer n greater than 1 and two positive integers p and q. Then it finds and prints all the principal ideals $\langle a\rangle$ of Z_n such that $\langle p\rangle\langle q\rangle = \langle n\rangle$. Run the program for $(n, p, q) = (12, 4, 6), (40, 2, 15)$.

Practical No. 9

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program which takes input of an integer a, two integers m and n greater than 1 and less than 100. Then it prints if the mapping $f(x) = ax$ from Z_m to Z_n is a homomorphism or not. Run the program for $(a, m, n) = (5, 4, 10), (3, 4, 12)$.

Theory

A ring homomorphism f from a ring R to a ring S is a mapping from R to S that preserves the two ring operations; that is, for all a, b in R,

$$f(a + b) = f(a) + f(b), \qquad \text{and} \qquad f(ab) = f(a)f(b).$$

Algorithm

```
Step 1: Prompt user to give input of integer $a$;
Step 2: Prompt user to give input of integer $m$ greater than 1 and less than 100;
Step 3: Check if $m$ is a positive integer greater than 1 and less than 100. If not, show an error and exit;
Step 4: Prompt user to give input of integer $n$ greater than 1 and less than 100;
Step 5: Check if $n$ is a positive integer greater than 1 and less than 100. If not, show an error and exit;
Step 6: Define a list zm and save elements of $Z_n$ in it;
Step 7: Define a variable isHomomorphism and save elements of $Z_m$ in it;
Step 8: Define variables $x$ and $y$ to hold the test points and set them to zero;
Step 9: Run a loop for each $x$ in $Z_m$;
Step 10: At each iteration, run an inner loop for each $y$ in $Z_m$;
Step 11: At each iteration of inner loop, calculate $f(x), f(y), x + y, f(x + y)$;
Step 12: If $f(x + y) = f(x) + f(y)$ does not hold, set isHomomorphism to False and exit inner loop;
```

Step 13: Next calculate $f(x), f(y), xy, f(xy)$;

Step 14: If $f(xy) = f(x)f(y)$ does not hold, set *isHomomorphism* to False and exit inner loop;

Step 15: If *isHomomorphism* is false, exit outer loop;

Step 16: Print the result.

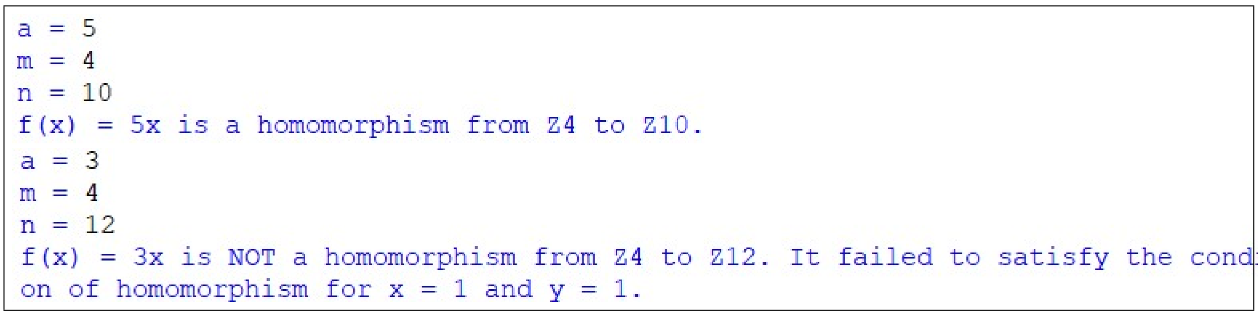
Program

The following PYTHON script implements the algorithm.

```
#Step 1
a = int(input("a = "))
#Step 2
m = int(input("m = "))
#Step 3
if (not (1 < m < 100)):
    raise ValueError(f"m should be a positive integer greater than 1 and less tha
#Step 4
n = int(input("n = "))
#Step 5
if (not (1 < n < 100)):
    raise ValueError(f"n should be a positive integer greater than 1 and less tha
#Step 6
zm = list(range(0, m))
#Step 7
isHomomorphism = True
#Step 8
x = 0
y = 0
#Step 9
for x in zm:
    #Step 10
    for y in zm:
        #Step 11
        phi_x = a*x % n
        phi_y = a*y % n
        xPlusy = (x + y) % m
        phi_xy = a*xPlusy % n
        #Step 12
        if phi_xy != (phi_x + phi_y) % n:
            #Step 13
            isHomomorphism = False
            break
        #Step 14
        phi_x = a*x % n
        phi_y = a*y % n
        xy = (x*y) % m
        phi_xy = a*xy % n
        #Step 15
        if phi_xy != (phi_x*phi_y) % n:
            #Step 16
            isHomomorphism = False
            break
    #Step 17
    if not isHomomorphism:
        break
#Step 18
if isHomomorphism:
    print(f'f(x) = {a if a!=1 and a!=0 else ""}{"x" if a!=0 else 0} is a homomorphism from
else:
    print(f'f(x) = {a if a!=1 and a!=0 else ""}{"x" if a!=0 else 0} is NOT a homomorphism f
```

Output

The output of the program for various inputs is as follows:



```
a = 5
m = 4
n = 10
f(x) = 5x is a homomorphism from Z4 to Z10.
a = 3
m = 4
n = 12
f(x) = 3x is NOT a homomorphism from Z4 to Z12. It failed to satisfy the conditi
on of homomorphism for x = 1 and y = 1.
```

Exercise Problem

Write a program which takes input of an integer a, two integers m and n greater than 1 and less than 100. Then it prints if the mapping $f(x) = x^a$ from Z_m to Z_n is a homomorphism or not. Run the program for $(a, m, n) = (2, 2, 2), (4, 6, 12)$.

Practical No. 10

Aim

To solve the given problem with the help of PYTHON.

Problem

Write a program which implements synthetic division for dividing polynomial $p(x) = a_nx^n + a_{n-1}x^{n-1} + \cdots + a_1x + a_0$ of degree 2 or more by $d(x) = x - a$. Run the program for $p(x) = 2x^5 - 9x^4 - 3x^2 + 23$ and $d(x) = x + 2$.

Theory

To divide $a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$ by $x - a$, follow these steps:

1. Write down the coefficients $a_n, a_{n-1}, \ldots, a_1, a_0$ in decreasing powers of x.

$$\begin{array}{c|cccc} a & a_n & a_{n-1} & \ldots & a_0 \\ & & aa_n & \ldots & \ldots \\ \hline & a_n & a_{n-1} + aa_n & \ldots & \ldots \end{array}$$

2. Set up the division by writing a from $x - a$ to the left of the coefficients.
3. Bring down the leading coefficient, a_n, as the starting value for the quotient.
4. Multiply this starting value by a and write the result under the next coefficient.
5. Add the result to the next coefficient, and repeat this process across all coefficients.

Algorithm

Step 1: Prompt user to give input of degree n of $p(x)$;

Step 2: Check if n is a positive integer greater than 1. If not, show an error and exit;

Step 3: Define a list *coefficients* to save coefficients of $p(x)$;

Step 4: Run a loop from $i = n$ to $i = 0$ to take input of coefficient of x^i in $p(x)$;

Step 5: At each iteration, verify that coefficient of x^n is non-zero. If not, show error and exit;

Step 6: Otherwise save the coefficient to the list of coefficients and exit the loop;

Step 7: Prompt user to give input of a of $d(x)$;

Step 8: Define a list quotient to save coefficients of quotient polynomial. Append it the coefficient of x^n;

Step 9: Run a loop for $i = 1$ to $i = n - 1$. At each iteration, calculate the coefficient of x^{n-1-i} of the quotient;

Step 10: Calculate the remainder of division of $p(x)$ by $d(x)$;

Step 11: Print the results.

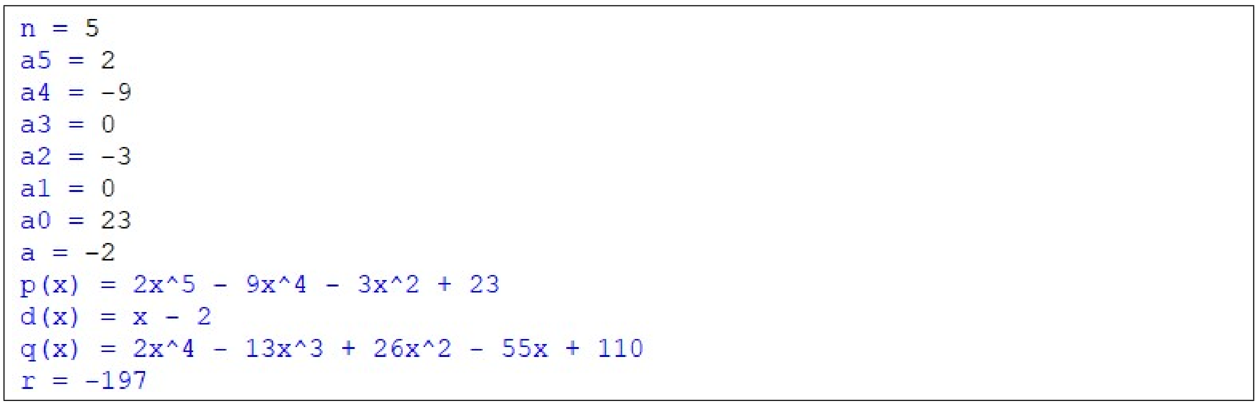
Program

The following PYTHON script implements the algorithm.

```
# Function to convert coefficients to polynomial
def polynomialStr(coefficients):
    n = len(coefficients) - 1
    polynomial = ''
    for idx, coefficient in enumerate(coefficients):
        if idx == 0:
            if n == 1:
                if coefficient == 1:
                    polynomial += f'x'
                elif coefficient == -1:
                    polynomial += f'x'
                elif coefficient != 0:
                    polynomial += f'{coefficient}x'
            else:
                if coefficient == 1:
                    polynomial += f'x^{n}'
                elif coefficient == -1:
                    polynomial += f'-x^{n}'
                elif coefficient != 0:
                    polynomial += f'{coefficient}x^{n}'
        elif idx == n-1:
            if coefficient == 1:
                polynomial += f' + x'
            elif coefficient == -1:
                polynomial += f' - x'
            elif coefficient > 0:
                polynomial += f' + {coefficient}x'
            elif coefficient < 0:
                polynomial += f' - {-coefficient}x'
        elif idx == n:
            if coefficient > 0:
                polynomial += f' + {coefficient}'
            elif coefficient < 0:
                polynomial += f' - {-coefficient}'
        else:
            if coefficient == 1:
                polynomial += f' + x^{n - idx}'
            elif coefficient == -1:
                polynomial += f' - x^{n - idx}'
            elif coefficient > 0:
                polynomial += f' + {coefficient}x^{n - idx}'
            elif coefficient < 0:
                polynomial += f' - {-coefficient}x^{n - idx}'
    return polynomial

#Step 1
n = int(input('n = '))
#Step 2
if n <= 2:
    raise ValueError('The degree of the polynomial must be an integer greater tha
#Step 3
coefficients = []
#Step 4
for i in range(n, -1, -1):
    coefficient = int(input(f'a{i} = '))
    #Step 5
    if i == n and coefficient == 0:
        raise ValueError('Coefficient of x^n cannot be zero.')
    #Step 6
    coefficients.append(coefficient)
#Step 7
a = int(input('a = '))
#Step 8
quotient = [coefficients[0]]
#Step 9
for i in range(1, n):
    quotient.append(coefficients[i] + a*quotient[i - 1])
#Step 10
remainder = coefficients[-1] + a*quotient[-1]

#Step 11
px = polynomialStr(coefficients)
qx = polynomialStr(quotient)
dx = polynomialStr([1, a])
print(f'p(x) = {px}')
print(f‘d(x) = {dx}’)
print(f‘q(x) = {qx}’)
print(f‘r = {remainder}’)
```

Output

The output of the program for various inputs is as follows:



```
n = 5
a5 = 2
a4 = -9
a3 = 0
a2 = -3
a1 = 0
a0 = 23
a = -2
p(x) = 2x^5 - 9x^4 - 3x^2 + 23
d(x) = x - 2
q(x) = 2x^4 - 13x^3 + 26x^2 - 55x + 110
r = -197
```

Exercise Problem

Write a program which implements remainder theorem and prints if $d(x) = x - a$ is a factor of a given polynomial $p(x)$ of degree more than 1. Run the program for $p(x) = 2x^3 - 9x^2 + x + 12$ and $d(x) = x - 4$.

Chapter 8

Complex Analysis

Practical No. 1

Aim

To find the sum of two complex numbers using the MATLAB code.

Problem

Find the sum of two complex numbers $z_1 = 2 + 3i$ and $z_2 = 4 + 8i$.

Algorithm

Step1: Write $z_1 = 2 + 1i$*3;
Step2: Press Enter
Step3: Write $z_2 = 4 + 1i$*8;
Step4: Press Enter
Step5: $z = z_1 + z_2$.

Matlab Code

```
z1=2+1i*3;
z2=4+1i*8;
z=z1+z2
```

Output

```
z =
   6.0000 +11.0000i
```

Exercise Problems

1. Find the sum of two complex numbers $z_1 = 5 + 6i$ and $z_2 = 7 + 9i$.
2. Find the sum of two complex numbers $z_1 = 15 + 61i$ and $z_2 = 17 + 90i$.

Practical No. 2

Aim

To find the product of two complex numbers using MATLAB code.

Problem

Find the product of $z_1 = 3 + 5i$ and $z_2 = 4 + 6i$.

Algorithm

Step1: $z_1 = 3 + 1i^*5$;
Step2: Press Enter
Step3: $z_2 = 4 + 1i^*6$;
Step4: Press Enter
Step5: $z = z_1^* z_2$
Step6: Press Enter
Step7: To display output, type fprintf('The product of two complex number is: %s $\backslash n'$, num2str(z)).

Matlab Code

```
z1=2+1i*3;
z2=3+1i*4;
z=z1*z2;
fprintf('The product of two complex number is:%s\n',num2str(z))
```

Output

```
The product of two complex number is:-6+17i
```

Exercise Problems

1. Find the product of $z_1 = 13 + 50i$ and $z_2 = 40 + 16i$.
2. Find the product of $z_1 = 25 + 20i$ and $z_2 = 30 + 6i$.

Practical No. 3

Aim

Use MATLAB code to find the Angle and Modulus of the complex Number.

Problem

Find the modulus and argument of the complex number $z = 3 + 4i$.

Algorithm

Step1: Write $x = 3$;
Step2: Press Enter
Step3: Write $y = 4$;
Step4: Write $z = x = 1i^{*}y$,
Step5: Press Enter
Step6: Write $r = abs(z)$;
Step7: Press Enter
Step8: Write $theta = atan2(x, y)$.

Matlab Code

```
x=3;
y=4;
z=x+1i*y;
r=abs(z)
theta=atan2(x,y)
```

Output

```
r =

     5
```

```
theta =

    0.6435
```

Exercise Problems

1. Find the modulus and argument of the complex number $z = 20 + 10i$.
2. Find the modulus and argument of the complex number $z = 10 + 2i$.

Practical No. 4

Aim

Convert the cartesian coordinate to a polar coordinate of a complex-valued function Using MATLAB code.

Problem

Convert the complex-valued function $f(z) = 2 + 5i$ to a polar form.

Algorithm

Step1: Write $x = 2$;
Step2: Press Enter
Step3: Write $y = 5$;
Step4: Press Enter
Step5: Write $[theta, r] = cart2pol(x, y)$.

Matlab Code

```
 x=2;
 y=5;
[theta, r]=cart2pol(x,y)
```

Output

```
 theta =

    1.1903
 r =

    5.3852
```

Exercise Problems

1. Convert the complex-valued function $f(z) = 4 + 8i$ to a polar form.
2. Convert the complex-valued function $f(z) = 16 + 12i$ to a polar form.

Practical No. 5

Aim

Using the MATLAB code to find the complex conjugate of a complex number.

Problem

Find the complex conjugate of $z = 3 + 5i$.

Algorithm

Step1: Write $z = 3 + 1i$*5;
Step2: Press Enter
Step3: Write $zc = conj(z)$;
Step4: Press Enter
Step5: To display the output, type fprintf('The conjugate of z is: %s$\backslash n'$, num2str(zc)).

Matlab Code

```
z=3+1i*5;
zc=conj(z);
fprintf('The conjugate of z is: %s\n',num2str(zc))
```

Output

```
The conjugate of z is: 3-5i
```

Exercise Problems

1. Find the complex conjugate of $z = 13 + 15i$.

2. Find the complex conjugate of $z = 23 + 25i$.

Practical No. 6

Aim

Find the roots of unity and plot the graph of a complex function by using MATLAB code.

Problem

To find the fifth root of unity.

Algorithm

Step1: Write $p = [1, 0, 0, 0, 0, -1]$;
Step2: Pres Enter
Step3: $z = roots(p)$;
Step4: Press Enter
Step5: To plot the graph by using the pre-defined MATLAB code i.e.$plot(z, "*")$;
Step6: Write axis equal to plot the graph on the same length of the x-axis and y-axis.
Step7: Write grid on to show the grid line on the graph.
Step8: Press Enter
Step9: Write xlabel(Re(z))
Step10: Press Enter
Step11: Write ylabel(Im(z))

Matlab Code

```
p=[1,0,0,0,0,-1];
z=roots(p);
plot(z,"*")
axis equal
grid on
xlabel("Re(z)")
ylabel("Im(z)")
```

Output

```
z =
  -0.8090 + 0.5878i
  -0.8090 - 0.5878i
   0.3090 + 0.9511i
   0.3090 - 0.9511i
   1.0000 + 0.0000i
```

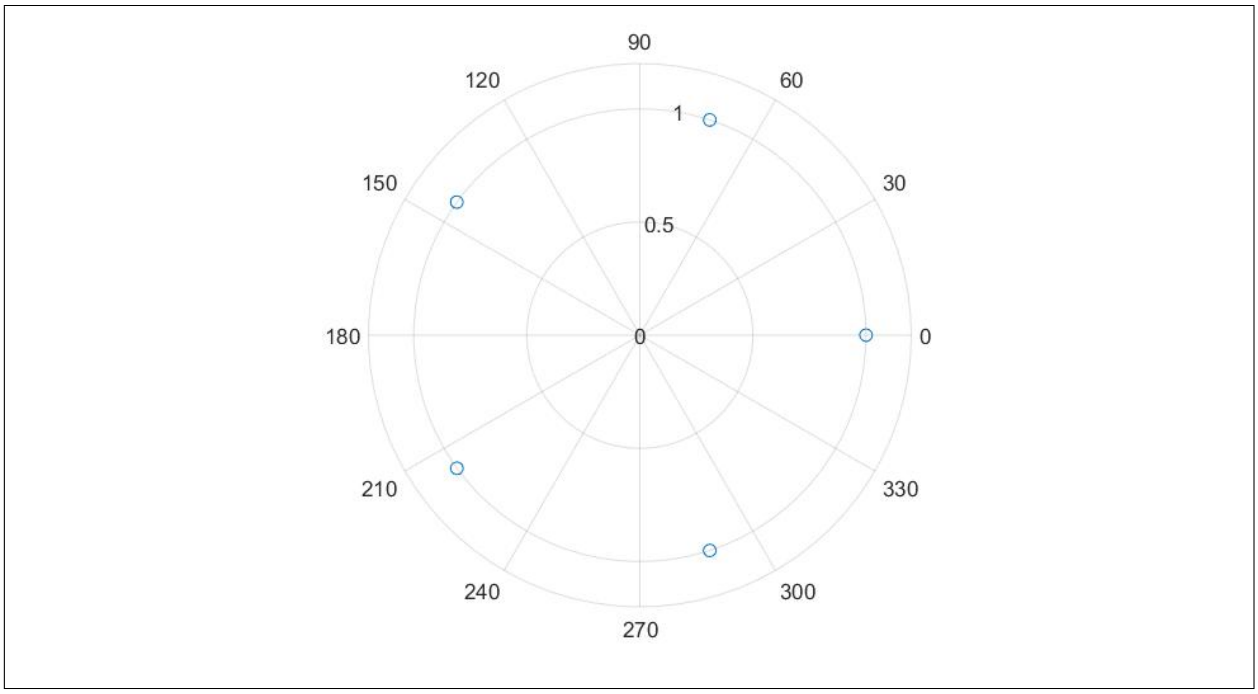


Figure 8.1: Graph of fifth root of unity

Exercise Problems

1. Find the sixth root of unity.

2. Find the third root of unity.

Practical No. 7

Aim

Plot the graph of the complex-valued function, using MATLAB code.

Problem

Plot the graph of the complex-valued function $f(z) = ze^z$.

Algorithm

Step1: Write $z = linspace(0, 4{*}pi, 200)$;
Step2: Press Enter
Step3: Write $w = z^*exp(1i^*z)$;
Step4: Press Enter
Step5: To plot the graph by using the pre-defined MATLAB code i.e. $plot(z, "{*}")$;
Step6: Write axis equal to plot the graph on the same length of the x-axis and y-axis.
Step7: Write grid on to show the grid line on the graph.
Step8: Press Enter
Step9: Write xlabel(Re(w));
Step10: Press Enter
Step11: Write ylabel(Im(w)).

Matlab Code

```
z = linspace(0,4*pi,200);
w = z.*exp(1i*z);
plot(w,"*")
axis equal
grid on
xlabel("Re(w)")
ylabel("Im(w)")
```

Output

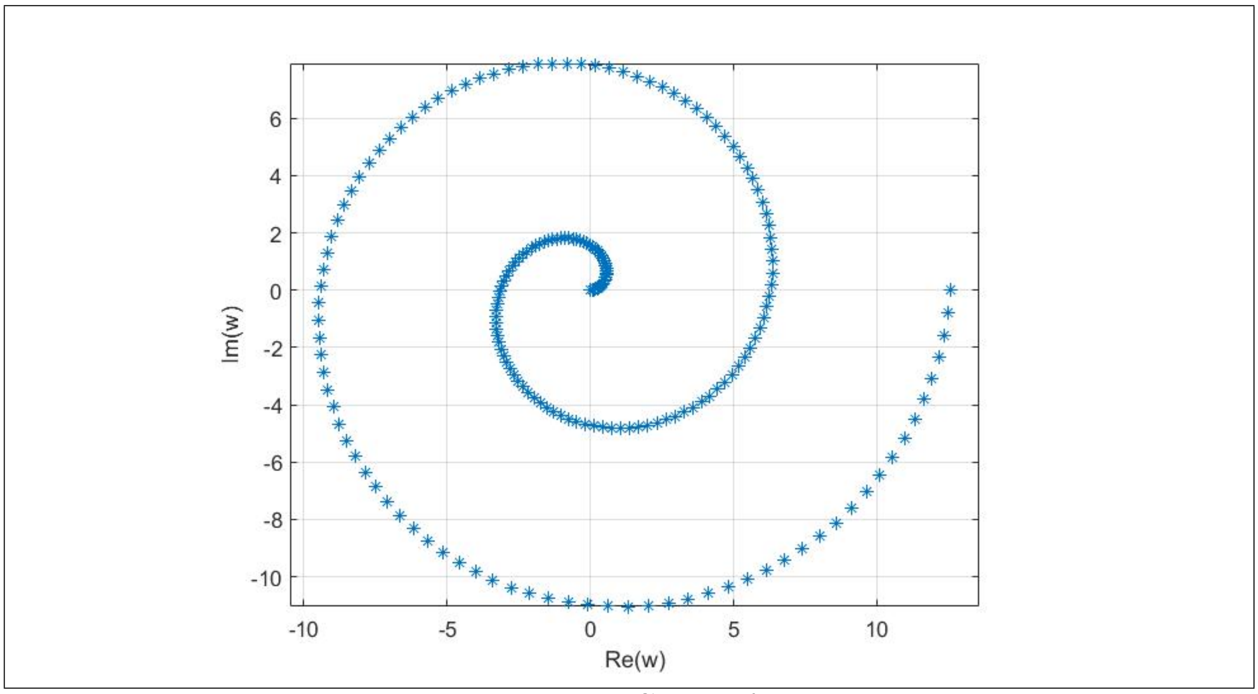


Figure 8.2: Graph of ze^z

Exercise Problems

1. Plot the graph of the complex-valued function $f(z) = z^2e^z$.
2. Plot the graph of the complex-valued function $f(z) = e^z$.

Practical No. 8

Aim

To find the poles and residues of the complex-valued function by using MATLAB code.

Problem

Find the poles and corresponding residues of the function $f(z) = \frac{4z+3}{2z^3-3.4z^2+1.98z-0.406}$.

Algorithm

Step1: In the complex-valued function $f(z), p(z) = 4z+3, q(z) = 2z^3-3.4z^2+1.98z-0.406$
Step2: Write $p = [4, 3]$; and press the enter key. Next, write $q = [2, -3.4, 1.98, -0.406]$;

and press the enter key.
Step3: To find the residue and poles by using the pre-defined residue MATLAB command i.e. $[r, a, k] = residue(p, q)$; and then press Enter. Here, r, a, and k denote the residue, poles, and the constant term of the function $f(z)$.

Matlab Code

```
p=[4 3];
q=[2 -3.4 1.98 -0.406];
[r,a,k]=residue(p,q)
```

Output

```
r =

  36.2500 + 0.0000i
 -18.1250 +13.1250i
 -18.1250 -13.1250i


a =

   0.7000 + 0.0000i
   0.5000 + 0.2000i
   0.5000 - 0.2000i


k =

     []
```

Exercise Problems

1. Find the poles and corresponding residues of the function $f(z) = \frac{2z+1}{z^3-3.4z^2+1.98z-0.406}$.
2. Find the poles and corresponding residues of the function $f(z) = \frac{z}{5z^3-3.4z^2+z-0.406}$.

Practical No. 9

Aim

Find the residue at infinity by using MATLAB code.

Problem

Find the residue at the infinity of the function $f(z) = \frac{z^2+3}{z^2-z+2}$.

Algorithm

Step1: Write $numerator = [1, 0, 3]$;
Step2: Press Enter
Step3: Write $denominator = [1, -1, 2]$;
Step4: Press Enter
Step5: Write $[r, p, k] = residue(numerator, denominator)$;
Step6: Press Enter
Step7: Write $residue\_sum = sum(r)$;
Step8: Press Enter
Step9: To display the output, type fprintf('The residue at infinity is: %s$\backslash n'$, num2str (residue_at_infinity)).

Matlab Code

```
numerator = [1 0 3];
denominator = [1 -1 2];
[r, p, k] = residue(numerator, denominator);
residue_sum = sum(r);
residue_at_infinity = -residue_sum;
fprintf('The residue at infinity is: %s\n', num2str(residue_at_infinity))
```

Output

```
The residue at infinity is: -1
```

Exercise Problems

1. Find the residue at the infinity of the function $f(z) = \frac{z^2+1}{z^2-z+1}$.

2. Find the residue at the infinity of the function $f(z) = \frac{z^2+2}{z^2-z+5}$.

Practical No. 10

Aim

Verify Cauchy-Riemann equations, using MATLAB.

Problem

Verify Cauchy-Riemann equations for the function $f(z) = (x^2 + y^2) + i(2xy)$ at the origin.

Algorithm

Step1: Define all the variables as symbolic variables i.e.

- Type syms x y real;
- Press Enter
- Type syms u(x,y) v(x,y);

Step2: Type the real and imaginary parts of the function as

- $u(x, y) = x^2 + y^2;$
- Press Enter
- $v(x, y) = 2 * x * y,$

Step3: Compute all partial derivatives

- $ux = diff(u, x);$
- $uy = diff(u, y);$
- $vx = diff(v, x);$
- $vy = diff(v, y);$

Step4: Check Cauchy-Riemann equations

- cr1=simplify(ux-vy);
- cr2=simplify(uy+vx);

Step5: Use the if-else statement

```
if isequal(cr1, 0) && isequal(cr2, 0)
disp('Cauchy-Riemann equations are satisfied');
else
disp('\Cauchy-Riemann equations are NOT  satisfied.');
end
```

Matlab Code

```
syms x y real;
syms u(x, y) v(x, y);
u(x, y) = x^2;
v(x, y) = 2*x*y;
% Compute partial derivatives
ux = diff(u, x);
uy = diff(u, y);
vx = diff(v, x);
vy = diff(v, y);
cr1 = simplify(ux - vy);
cr2 = simplify(uy + vx);
if isequal(cr1, 0) && isequal(cr2, 0)
    disp('Cauchy-Riemann equations are satisfied.');
else
    disp('Cauchy-Riemann equations are NOT satisfied.');
end
```

Output

```
Cauchy-Riemann equations are NOT satisfied.
```

Exercise Problems

1. Verify Cauchy-Riemann equations for the function $f(z) = (x^2 + y^3) + i(2xy)$ at the origin.

2. Verify Cauchy-Riemann equations for the function $f(z) = (x^2 - y^2) + i(2x)$ at the origin.

Practical No. 11

Aim

To find the singularity of the complex-valued function, using MATLAB.

Problem

Find the singularity of $f(z) = \frac{1}{z^2+5z+6}$.

Algorithm

Step1: Type syms z;
Step2: Press Enter
Step3: Type $f(z) = 1/(z^2 + 5 * z + 6)$;
Step4: Press Enter
Step5: Type singularity = solve ((z^2+5*z+6)==0,z);
Step6: To display the output use the disp (‘The singularity of the function are:’); disp(singularity);

Matlab Code

```
syms z;
f(z) =1/(z^2+5*z+6);
singularity = solve((z^2+5*z+6) == 0, z);
disp(‘The singularity of the function are:’);
disp(singularity);
```

Output

```
The singularity of the function are:
-3
-2
```

Exercise Problems

1. Find the singularity of $f(z) = \frac{1}{z^3+15z+60}$.
2. Find the singularity of $f(z) = \frac{1}{z^4+10z+50}$.

Practical No. 12

Aim

Check the continuity of a complex-valued function, using MATLAB code.

Problem

Check the continuity of the function $f(z) = \cos(z)$, at $z_0 = 2$.

Algorithm

Step1: Define the complex variable z as a symbolic variable;
Type syms z;
Step2: Type $f(z) = \cos(z)$;
Step3: Type $z_0 = 2$;
Step4: Type `f_z0` $= subs(f(z), z, z0)$; for symbolic substitution
Step5: Type `limit_f_z0` $= limit(f(z), z, z0)$;
Step6: Check the continuity condition using the if-else statement

Matlab Code

```
syms z;
f(z) = cos(z);
z0 = 2;
f_z0 = subs(f(z), z, z0);
limit_f_z0 = limit(f(z), z, z0);
disp([‘f(z0) = ’, char(f_z0)]);
disp([‘Limit of f(z) as z approaches z0 = ‘, char(limit_f_z0)]);
if isequal(f_z0, limit_f_z0)
    fprintf(‘The function is continuous at z = %s\n’,num2str(z0) );
else
    fprintf(‘The function is not continuous at z =%s\n’,num2str(z0));
end
```

Output

```
f(z0) = cos(2)
Limit of f(z) as z approaches z0 = cos(2)
The function is continuous at z = 2
```

Exercise Problems

1. Check the continuity of the function $f(z) = \sin(z)$, at $z_0 = 0$.
2. Check the continuity of the function $f(z) = \exp(z)$, at $z_0 = 0$.

Practical No. 13

Aim

To find the order of zeros of a complex function by using MATLAB code.

Problem

Find the order of zeros of $f(z) = z^3 - 1$

Algorithm

Step1: Initialize multiplicity=zeros(size(zeros))
Step2: Define z as a symbolic variable
Syms z;
Step3: Define the complex function `f_sym` $=$ `z^3` $- 1$;
Step4: Type `df_sym` $= diff$(`f_sym, z`);
Step5: Use for loop to find the order of zeros
Step6: To display the output by using the "disp command

Matlab Code

```
multiplicity = zeros(size(zeros));
syms z
f_sym = z^3 - 1;
df_sym = diff(f_sym, z);

for k = 1:length(zeros)
    order = 1;
    while abs(subs(diff(f_sym, z, order), z, zeros(k))) < 1e-10
        order = order + 1;
    end
    multiplicity(k) = order - 1;
end
disp('The order of zeros of function is:');
disp(order);
```

Output

```
The order of zeros of function is:
     3
```

Exercise Problems

1. Find the order of zeros of $f(z) = z^4 - 5$.
2. Find the order of zeros of $f(z) = z^6 - 1$.

Practical No. 14

Aim

Check whether a complex-valued function function satisfies Cauchy-Riemann equations or not, using MATLAB.

Problem

Verify Cauchy-Riemann equations for the function $f(z) = \log(z)$ at $z = 0$.

Algorithm

Step1: Define the symbolic variable as syms x y real;
Step2: Type $z = x + 1i * y$;
Step3: Define a complex function $f(z) = log(z)$;
Step4: Define the Real part of the function to type u=real(f);
Step5: Define the imaginary part of the function to type v=img(f);
Step6: Compute all partial derivatives to type the following command
$du\_dx = diff(u, x)$;
$du\_dy = diff(u, y)$;
$dv\_dx = diff(v, x)$;
$dv\_dy = diff(v, y)$;
Step7: Display the all partial derivatives to use the "fprintf command
fprintf('Partial derivatives:$\backslash n'$;)
fprintf($'du/dx = \%s\backslash n', du\_dx$);
fprintf($'du/dy = \%s\backslash n', du\_dy$);
fprintf($'dv/dx = \%s\backslash n', dv\_dx$);
fprintf($'dv/dy = \%s\backslash n', dv\_dy$);
Step8: To check the Cauchy Riemann equations use using the if-else statement

```
if isequal(simplify(du_dx - dv_dy), 0) && isequal(simplify(du_dy + dv_dx), 0)
    fprintf('The function satisfies the Cauchy-Riemann equations.\n');
else
    fprintf('The function does not satisfy the Cauchy-Riemann equations. \n');
end
```

Matlab Code

```
syms x y real;
z = x + 1i*y;
f = log(z); % Define the complex function
```

```
u = real(f); % Real part of the function
v = imag(f); % Imaginary part of the function
% Compute the partial derivatives
du_dx = diff(u, x);
du_dy = diff(u, y);
dv_dx = diff(v, x);
dv_dy = diff(v, y);
% Display the partial derivatives
fprintf('Partial derivatives:\n');
fprintf('du/dx = %s\n', du_dx);
fprintf('du/dy = %s\n', du_dy);
fprintf('dv/dx = %s\n', dv_dx);
fprintf('dv/dy = %s\n', dv_dy);
% Check the Cauchy-Riemann equations
fprintf('\nChecking the Cauchy-Riemann equations:\n');
if isequal(simplify(du_dx - dv_dy), 0) && isequal(simplify(du_dy + dv_dx), 0)
    fprintf('The function satisfies the Cauchy-Riemann equations.\n');
else
    fprintf('The function does not satisfy the Cauchy-Riemann equations
    and is not differentiable.\n');
end
```

Output

```
Partial derivatives:
du/dx = x/(x^2 + y^2)
du/dy = y/(x^2 + y^2)
dv/dx = -y/(x^2 + y^2)
dv/dy = x/(x^2 + y^2)

Checking the Cauchy-Riemann equations:
The function does not satisfy the Cauchy-Riemann equations and is not differentiable.
```

Exercise Problems

1. Verify Cauchy-Riemann equations for the function $f(z) = \sin(z)$ at $z = 0$.
2. Verify Cauchy-Riemann equations for the function $f(z) = \cos(z)$ at $z = 0$.

Practical No. 15

Aim

Find the Laurent series expansion of complex function using the MATLAB code.

Problem

Find the Laurent series of $f(z) = \frac{1}{z^2+z-2}$.

Algorithm

Step1: Define the complex variable z as symbolic variable Syms z;
Step2: Type the complex function
$f = \frac{1}{z^2+z-2}$
Step3: Define a=0;
Step4: Convert the function into its partial sum by use the command
f_partial = partfrac(f, z);
Step5: Split the function into positive power term of z and negative power term of z as
positive_powers = taylor(f_partial, z, 'Order', 10);
negative_powers = taylor(1/f_partial, z, 'Order', 10);
Step6: Add the positive power term and negative power term of z
L = positive_powers + negative_powers;
Step7: To display the output use the disp command
disp('Laurent series expansion:');
disp(L).

Matlab Code

```
syms z;
f = 1/(z^2 + z - 2);
a = 0;
f_partial = partfrac(f, z);
positive_powers = taylor(f_partial, z, 'Order', 10);
negative_powers = taylor(1/f_partial, z, 'Order', 10);
L = positive_powers + negative_powers;
disp('Laurent series expansion:')
disp(L)
```

Output

```
Laurent series expansion:
- (341*z^9)/1024 - (171*z^8)/512 - (85*z^7)/256 - (43*z^6)/128 - (21*z^5)/64 -
(11*z^4)/32 - (5*z^3)/16 + (5*z^2)/8 + (3*z)/4 - 5/2
```

Exercise Problems

1. Find the Laurent series of $f(z) = \frac{1}{z^3+z-5}$.

2. Find the Laurent series of $f(z) = \frac{1}{z^2+2z-6}$.

Practical No. 16

Aim

Find the poles of the complex function using MATLAB code .

Problem

Find the poles of $f(z) = \frac{1}{(z-1)(z-2)}$.

Matlab Code

```
syms z
f = 1/((z-1)*(z-2));
pole = solve(1/f == 0, z);
disp('Poles of the function:')
disp(pole)
```

Output

```
Poles of the function:
1
2
```

Exercise Problems

1. Find the poles of $f(z) = \frac{1}{(z-2)(z-5)}$.
2. Find the poles of $f(z) = \frac{1}{(z-6)(z-7)}$.

Practical No. 17

Aim

Find the integration of complex-valued function using MATLAB code.

Problem

Find the integral of $f(z) = \frac{1}{(z-2)}$ on $|z| < 3$.

Matlab Code

```
f = @(z) 1./(z-2);
theta = linspace(0, 2*pi, 100);
r = 3;
z = r * exp(1i * theta);
fz = f(z);
integral = trapz(theta, fz .* (1i * r * exp(1i * theta)));
disp([‘Integral over the closed contour: ’, num2str(integral)]);
```

Output

```
Integral over the closed contour: -4.0593e-16+6.2832i
```

Exercise Problems

1. Find the integral of $f(z) = \frac{1}{(z-3)}$ on $|z| < 4$.
2. Find the integral of $f(z) = \frac{1}{(z-5)}$ on $|z| < 5$.

Practical No. 18

Aim

Find the integration of complex-valued functions with the help of Cauchy integral formula for derivatives using MATLAB code.

Problem

Find the integral of $f(z) = \frac{1}{z^2+z-2}$ on $\mid z - 1 \mid < 1$.

Algorithm

Step1: Define z as a symbolic variable Syms z;
Step2: Type the complex function f=1/(z^2+z-2);
Step3: Define the point, radius and the integrand
z0 = 1;
n = 1;
integrand = f/(z-z0)^(n+1);
r=1;
Step4: Type theta and contour as
theta = linspace(0, 2*pi, 10);

contour = z0 + r*exp(1i*theta);
Step5: Compute the integral using numerical integration
integral_value = trapz(contour, subs(integrand, z, contour)) * (contour(2) - contour(1));
Step6: Compute the derivative using the Cauchy Integral Formula
f_derivative = factorial(n) / (2*pi*1i) * integral_value;
Step7: Display the result using the disp command
disp(['The ', num2str(n), '-th derivative of f(z) at z0 = ', num2str(z0), ' is:']);
disp(vpa(f_derivative, 6)); % Display the result with 6 significant digits

Matlab Code

```
syms z;
f = 1/(z^2 + z - 2);
z0 = 1;
n = 1;
integrand = f / (z - z0)^(n + 1);
r = 1;
theta = linspace(0, 2*pi, 10);
contour = z0 + r*exp(1i*theta);
% Compute the integral using numerical integration (trapezoidal rule)
integral_value = trapz(contour, subs(integrand,z,contour))*(contour(2)-contour(1));
% Compute the derivative using the Cauchy Integral Formula
f_derivative = factorial(n) / (2*pi*1i) * integral_value;
% Display the result
disp([‘The ’, num2str(n), ’-th derivative of f(z) at z0 = ’, num2str(z0), ’ is:’]);
disp(vpa(f_derivative, 6)); % Display the result with 6 significant digits
```

Output

```
The 1-st derivative of f(z) at z0 = 1 is:
- 0.0079777 + 0.0219185i
```

Exercise Problems

1. Find the integral of $f(z) = \frac{1}{z^2+z-4}$ on $\mid z-1 \mid < 1$.

2. Find the integral of $f(z) = \frac{1}{z^2+z+5}$ on $\mid z-1 \mid < 2$.

Practical No. 19

Aim

Check the Singularity type of the complex-valued function using MATLAB code.

Problem

To check the singularity types of $f(z) = \frac{1}{(z-1)(z-2)}$.

Algorithm

Step1: Define z as a symbolic variable Syms z;
Step2: Define the complex function f = 1/((z-1)*(z-2));
Step3: For singularities of the function
singularities = solve(1/f == 0, z);
Step4: To determine the type of singularity use for loop

```
disp('Singularities and their types:');
for i = 1:length(singularities)
    singularity = singularities(i);
% Check if it is a pole by examining the limit
  order = limit((z - singularity)^2 * f, z, singularity);
  if isfinite(order)
    disp(['Pole at z = ', char(singularity), ' of order 2']);
  else
    order = limit((z - singularity) * f, z, singularity);
    if isfinite(order)
      disp(['Pole at z = ', char(singularity), ' of order 1']);
    else
      disp(['Essential singularity or other type at z = ', char(singularity)]);
     end
  end
end
```

Matlab Code

```
% Import the symbolic math toolbox
syms z
% Define the complex function
f = 1/((z-1)*(z-2));
% Find the singularities of the function (poles)
singularities = solve(1/f == 0, z);
% Determine the type of singularity
disp('Singularities and their types:');
for i = 1:length(singularities)
```

```
    singularity = singularities(i);

    % Check if it is a pole by examining the limit
    order = limit((z - singularity)^2 * f, z, singularity);
    if isfinite(order)
        disp(['Pole at z = ', char(singularity), ' of order 2']);
    else
        order = limit((z - singularity) * f, z, singularity);
        if isfinite(order)
            disp(['Pole at z = ', char(singularity), ' of order 1']);
        else
            disp(['Essential singularity or other type at z = ', char(singularity)]);
        end
    end
end
```

Output

```
Singularities and their types:
Pole at z = 1 of order 2
Pole at z = 2 of order 2
```

Exercise Problems

1. To check the singularity types of $f(z) = \frac{1}{(z-2)(z-5)}$.
2. To check the singularity types of $f(z) = \frac{1}{(z-7)(z-8)}$.

Chapter 9

Linear Algebra

Practical No. 1

Aim

To define the matrices in MATLAB.

Problem

Write a MATLAB code for defining the following matrices in the MATLAB

$$A = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \qquad and \qquad B = \begin{bmatrix} 2 & 3 & 4 \\ 3 & 2 & 1 \\ 6 & 5 & 8 \end{bmatrix}.$$

Algorithm

Step 1: Open the command window of the MATLAB;
Step 2: Define the matrix in the array form. The elements are given row-wise and rows are separated with semicolons;
Step 3: To enter the elements of the row, put space between the elements of the row or separate them with commas.

Program

```
>> A = [5,6; 7,8]

A =

     5     6
```

```
    7     8

>> A = [2 3 4; 3 2 1; 6 5 8]

A =

    2    3    4
    3    2    1
    6    5    8
```

Practical No. 2

Aim

To solve the problem of performing addition of the matrices using MATLAB.

Problem

Write a program for finding A+B, A+3I for the following matrix

$$A = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} \qquad and \qquad B = \begin{bmatrix} 6 & 7 \\ 8 & 8 \end{bmatrix}.$$

Algorithm

Step1: Define a function, *matrix operations()* in the function file of the Matlab program. This function takes input of the two matrices from the user and then perform the required operations;
Step2: Inside the function give command to input the matrices;
Step3: The addition of matrices can be done if their rows and columns are equal. So, put a condition to check the index of both the matrices;
Step4: Perform the required operations and display the result;
Step5: Now, run the function *matrix operations()* in the command window;
Step6: Enter the matrix A in the array form and press Enter;
Step7: Enter the matrix B in the array form and again press Enter to obtain the results.

Program

```
function matrix_operations()
    % Ask the user to input the matrix A
    A = input('Enter the matrix A: ');
    % Ask the user to input the matrix B
    B = input('Enter the matrix B: ');

    % Check if the matrices A and B are of the same size
    [m, n] = size(A);
    [p, q] = size(B);

    if m ~= p || n ~= q
        error('Matrices A and B must be of the same size');
    end

    % Compute A + B
    A_plus_B = A + B;

    % Compute the identity matrix I of the same size as A
    I = eye(m);

    % Compute A + 3I
    A_plus_3I = A + 3 * I;

    % Display the results
    disp('Matrix A + B:');
    disp(A_plus_B);
    disp('Matrix A + 3I:');
    disp(A_plus_3I);
end
```

Output

```
 matrix_operations
Enter the matrix A: [5 6; 7 8]
Enter the matrix B: [6 7; 8 8]
Matrix A + B:
    11    13
    15    16

Matrix A + 3I:
     8     6
     7    11
```

Aim

To solve the problem of performing multiplication of the matrices using MATLAB.

Problem

Write a program for finding AB, A^2B^3 for the following matrix

$$A = \begin{bmatrix} 1 & 3 & 4 \\ 6 & 1 & 2 \\ 2 & 4 & 8 \end{bmatrix} \quad \mathit{and} \quad B = \begin{bmatrix} 2 & 1 & 1 \\ 3 & 2 & 7 \\ 2 & 4 & 8 \end{bmatrix}.$$

Algorithm

Step1: Define a function, *matrix multiplications()* in the function file of the Matlab program. This function takes input of the two matrices from the user and then perform the required operations;
Step2: Inside the function give command to input the matrices;
Step3: The multiplication of the matrices can be done if the number of columns in A are equal to number of rows in the B. So, put a command to check this condition;
Step4: Perform the required operations and display the result;
Step5: Now, run the function matrix multiplications() in the command window;

Program

```
function matrix_multiplications()
    % Ask the user to input the matrix A
    A = input('Enter the matrix A: ');
    % Ask the user to input the matrix B
    B = input('Enter the matrix B: ');
    % Check if the matrices A and B can be multiplied (for AB)
    [m, n] = size(A);
    [p, q] = size(B);
    if n ~= p
        error('Number of columns in A must be equal to the number of rows in B for AB');
    end
    % Compute AB
    AB = A * B;
    % Display the results
    disp('Matrix AB:');
    disp(AB);
    if m ~= n
        error('Matrix A must be square to find A^2');
    end
    if p ~= q
        error('Matrix B must be square to find B^3');
    end
    % Compute A^2
    A2 = A * A;
    % Compute B^3
    B3 = B * B * B;
    % Check if the matrices A^2 and B^3 can be multiplied (for A^2B^3)
    [r, s] = size(A2);
    [t, u] = size(B3);
    if s ~= t
        error('Number of columns in A^2 must be equal to the number of rows in B^
    end
    % Compute A^2B^3
    A2B3 = A2 * B3;
    disp('Matrix A^2B^3:');
    disp(A2B3);
end
```

Output

```
matrix_multiplications
Enter the matrix A: [1 3 4;6 1 2;2 4 8]
Enter the matrix B: [2 1 1;3 2 7;9 4 2]
Matrix AB:
    47    23    30
    33    16    17
    88    42    46

Matrix A^2B^3:
       38613        18974        23752
       39578        19483        24628
       72138        35450        44392
```

Practical No. 4

Aim

To solve the problem of finding the determinant and inverse of the matrices using MATLAB.

Problem

Write a program for finding the determinant and inverse for the following matrix

$$A = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}.$$

Algorithm

Step1: Define a function, *det inverse()* in the function file of the MATLAB program. This function takes input of the matrix from the user and then perform the required operations;
Step2: Inside the function give command to input the matrix;
Step3: To find determinant of the matrix give command *det(A)*;
Step4: To find the inverse of the matrix check whether the matrix is square or not;
Step5: Check whether the matrix is singular or not;
Step6: Compute the inverse of the matrix using command *inv(A)*;
Step7: Display the results of both operations;
Step8: Run the function *det inverse()* in the command window;

Program

```
function det_inverse()
    % Ask the user to input the matrix A
    A = input('Enter the matrix A: ');
    d = det(A);
    % Check if the matrix A is square
    [m, n] = size(A);
    if m ~= n
        error('Matrix A must be square to find its inverse');
    end

    % Check if the matrix A is invertible
    if d == 0
        error('Matrix A is singular and cannot be inverted');
    end

    % Compute the inverse of A
    A_inv = inv(A);

    % Display the results
    disp('Determinant of matrix A:');
    disp(d);
    disp('Inverse of matrix A:');
    disp(A_inv);
```

```
end
```

Output

```
Enter the matrix A: [5 6;7 8]
Determinant of matrix A:
   -2.0000

Inverse of matrix A:
   -4.0000    3.0000
    3.5000   -2.5000
```

Practical No. 5

Aim

To solve the problem of finding the transpose of the matrices using MATLAB.

Problem

Write a program for finding the transpose for the following matrix

$$A = \begin{bmatrix} 2 & 1 & 1 \\ 3 & 2 & 7 \\ 9 & 4 & 2 \end{bmatrix}.$$

Algorithm

Step1: Define a function, *trans()* in the function file of the Matlab program. This function takes input of the matrix from the user and then perform the transpose of matrix;
Step2: Inside the function give command to input the matrix;
Step3: To find transpose of the matrix give command **A'** ;
Step4: Display the results of transpose operation;
Step5: Run the function *trans()* in the command window;

Program

```
function Trans()
    % Ask the user to input the matrix A
    A = input('Enter the matrix A: ');
    % Compute the transpose of A
    A_trans = A';
    % Display the results
    disp('Transpose of matrix A:');
    disp(A_trans);
end
```

Output

```
Trans
Enter the matrix A: [2 1 1;3 2 7;9 4 2]
Transpose of matrix A:
     2     3     9
     1     2     4
     1     7     2
```

Practical No. 6

Aim

To solve the problem of finding trace of the matrices using MATLAB.

Problem

Write a program for finding trace for the following matrix

$$A = \begin{bmatrix} 4 & 0 & 2 \\ 0 & 5 & 2 \\ 5 & 4 & 10 \end{bmatrix}.$$

Algorithm

Step1: Define a function, *find trace of matrix()* in the function file of the MATLAB program. This function takes input of the matrix from the user and then find the trace of the matrix;
Step2: Inside the function give command to input the matrix;
Step3: To find trace of the matrix, first check whether the matrix is square or not;
Step4: Then compute the trace of the matrix by command, *trace(A)"*;
Step5: Display the result by *fprintf* command;

Program

```
function find_trace_of_matrix()
    % Ask the user to input the matrix A
    A = input('Enter the matrix A: ');

    % Check if the matrix A is square
    [m, n] = size(A);
    if m ~= n
        error('Matrix A must be square to find its trace');
    end

    % Compute the trace of A
    A_trace = trace(A);

    % Display the result
    fprintf('The trace of the matrix A is: %.2f\n', A_trace);
end
```

Output

```
find_trace_of_matrix
Enter the matrix A: [4 0 2;0 5 2;5 4 10]
The trace of the matrix A is: 19.00
```

Practical No. 7

Aim

To solve the problem of finding the eigenvalues of the matrices using MATLAB.

Problem

Write a program for finding the eigenvalue for the following matrix

$$\begin{bmatrix} 2 & -3 & 0 \\ 2 & -5 & 0 \\ 0 & 0 & 3 \end{bmatrix}.$$

Algorithm

Step1: Define a function, *find eigenvalues()* in the function file of the MATLAB program. This function takes input of the matrix from the user and then find the eigenvalues of the

matrix;
Step2: Inside the function give command to input the matrix;
Step3: To find eigenvalues of the matrix, first check whether the matrix is square or not;
Step4: Compute the eigenvalues of the matrix by command, *eig(A)*;
Step5: Display the result by *disp* command;

Program

```
function find_eigenvalues()
    % Ask the user to input the matrix A
    A = input('Enter the matrix A: ');

    % Check if the matrix A is square
    [m, n] = size(A);
    if m ~= n
        error('Matrix A must be square to find its eigenvalues');
    end

    % Compute the eigenvalues of A
    eigenvalues = eig(A);

    % Display the result
    disp('The eigenvalues of the matrix A are:');
    disp(eigenvalues);
end
```

Output

```
find_eigenvalues
Enter the matrix A: [2 -3 0;2 -5 0;0 0 3]
The eigenvalues of the matrix A are:
     1
    -4
     3
```

Practical No. 8

Aim

To solve the problem of finding the characteristic polynomial of the matrices using MATLAB.

Problem

Write a program for finding the characteristic polynomial for the following matrix

$$\begin{bmatrix} 0 & 6 & 8 \\ 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \end{bmatrix}.$$

Algorithm

Step1: Define a function, *find characteristic polynomials()* in the function file of the MATLAB program. This function takes input of the matrix from the user and then find the characteristic polynomial of the matrix;
Step2: Inside the function give command to input the matrix;
Step3: To find the characteristic polynomial of the matrix, first check whether the matrix is square or not;
Step4: Then compute the characteristic polynomial of the matrix by command, *poly(A)*;
Step5: Display the result by *fprintf* command;

Program

```
function find_characteristic_polynomial()
    % Ask the user to input the matrix A
    A = input('Enter the matrix A: ');

    % Check if the matrix A is square
    [m, n] = size(A);
    if m ~= n
        error('Matrix A must be square to find its characteristic polynomial');
    end

    % Compute the characteristic polynomial coefficients of A
    char_poly = poly(A);

    % Display the result
    fprintf('The characteristic polynomial of the matrix A is:\n');
    disp(char_poly);
end
```

Output

```
find_characteristic_polynomial
Enter the matrix A: [0 6 8;0.5 0 0;0 0.5 00]
```

```
The characteristic polynomial of the matrix A is:
    1.0000   -0.0000   -3.0000   -2.0000
```

Practical No. 9

Aim

To solve the problem of finding the rank of the matrices using MATLAB.

Problem

Write a program for finding the rank for the following matrix

$$\begin{bmatrix} 10 & 2 \\ 4 & 5 \\ 5 & 10 \end{bmatrix}.$$

Algorithm

Step1: Define a function, *find matrix rank()* in the function file of the MATLAB program. This function takes input of the matrix from the user and then find characteristic polynomial of the matrix;
Step2: Inside the function give command to input the matrix;
Step3: Then compute the rank of the matrix by command, *rank(A)*;
Step4: Display the result by *fprintf* command;

Program

```
function find_matrix_rank()
    % Ask the user to input the matrix A
    A = input('Enter the matrix A: ');

    % Compute the rank of the matrix A
    matrix_rank = rank(A);

    % Display the result
    fprintf('The rank of the matrix A is: %d\n', matrix_rank);
end
```

Output

```
>> find_matrix_rank
Enter the matrix A: [10,2;4 5;5 10]
The rank of the matrix A is: 2
```

Practical No. 10

Aim

To solve the problem of linear independence of the column of the matrices using MATLAB.

Problem

Write a MATLAB code to show the linear independence of the column of the following matrix

$$\begin{bmatrix} 2 & 4 & 10 \\ 3 & -7 & 11 \\ -1 & 4 & 10 \end{bmatrix}.$$

Program

Algorithm

Step1: Define a function, *column linear independence()* in the function file of the Matlab program. This function takes input of the matrix from the user and then show whether the columns of the matrix are linearly independent or dependent;
Step2: Inside the function give command to input the matrix;
Step3: Then compute the rank of the matrix by command, *rank(A)*;
Step4: The columns of matrix are linearly independent if the number of columns is equal to rank of matrix, so put a test condition using *if* command to check if it is equal or not.
Step5: Display the result by *disp* command.

```
function column_linear_independence()
   % Ask the user to input the matrix A
   A = input('Enter the matrix A: ');

   % Get the number of rows and columns
   [m, n] = size(A); % m = rows, n = columns

   % Compute the rank of the matrix A
   r = rank(A);
```

```
    % Check if the rank is equal to the number of columns
    if r == n
        disp('The columns of the matrix A are linearly independent.');
    else
        disp('The columns of the matrix A are linearly dependent.');
    end
end
```

Output

```
Enter the matrix A: [2 4 10; 3 -7 11;-1 4 10]
The columns of the matrix A are linearly independent.
```

Practical No. 11

Aim

To solve the problem of linear independence of the row of the matrices using MATLAB.

Problem

Write a program to show the linear independence of the row of the following matrix

$$\begin{bmatrix} 4 & 2 & 3 \\ 2 & 1 & 4 \\ 5 & 8 & 2 \end{bmatrix}.$$

Algorithm

Step1: Define a function, *row linear independence()* in the function file of the MATLAB program. This function takes input of the matrix from the user and then show whether the rows of the matrix are linearly independent or dependent;
Step2: Inside the function give command to input the matrix;
Step3: Then compute the rank of the matrix by command, *rank(A)*;
Step4: The rows of matrix are linearly independent if the number of rows is equal to rank of matrix, so put a test condition using *if* command to check if it is equal or not;
Step5: Display the result by *disp* command;

Program

```
function row_linear_independence()
    % Ask the user to input the matrix A
    A = input('Enter the matrix A: ');

    % Get the number of rows and columns
    [m, n] = size(A); % m = rows, n = columns

    % Compute the rank of the matrix A
    r = rank(A);

    % Check if the rank is equal to the number of rows
    if r == m
        disp('The rows of the matrix A are linearly independent.');
    else
        disp('The rows of the matrix A are linearly dependent.');
    end
end
```

Output

```
Enter the matrix A: [4 2 3;2 1 4;5 8 2]
The rows of the matrix A are linearly independent.
```

Practical No. 12

Aim

To solve the problem to identify the type of the matrices using MATLAB.

Problem

Write a program to check whether the matrix is symmetric or skew-symmetric

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 5 & 4 \\ 3 & 4 & 6 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 2 & -3 \\ -2 & 0 & -4 \\ 3 & 4 & 0 \end{bmatrix} \qquad C = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}.$$

Algorithm

Step1: Define a function, *check symmetric skew-symmetric ()* in the function file of the Matlab program. This function takes input of the matrix from the user and then perform the required operations;
Step2: Inside the function give command to input the matrix.
Step3: To check the type of matrices, first we check whether it is square matrix or not;
Step4: Perform the required operations and display the result;

Program

```
function check_symmetric_skew_symmetric()
    % Ask the user to input the matrix A
    A = input('Enter the matrix A: ');

    % Get the number of rows and columns
    [m, n] = size(A);

    % Check if the matrix A is square
    if m ~= n
        error('Matrix A must be square to check if it is symmetric or skew-symmetric');
    end

    % Check if the matrix is symmetric
    if isequal(A, A')
        disp('The matrix A is symmetric.');
    elseif isequal(A, -A')
        disp('The matrix A is skew-symmetric.');
    else
        disp('The matrix A is neither symmetric nor skew-symmetric.');
    end
end
```

Output

```
 check_symmetric_skew_symmetric
Enter the matrix A: [1 2 3;2 5 4;3 4 6]
The matrix A is symmetric.
 check_symmetric_skew_symmetric
Enter the matrix A: [0 2 -3;-2 0 -4;3 4 0]
The matrix A is skew-symmetric.
 check_symmetric_skew_symmetric
Enter the matrix A: [1 2 3;4 5 6;7 8 9]
```

```
The matrix A is neither symmetric nor skew-symmetric.
```

Practical No. 13

Aim

To solve the problem to identify the type of the matrices using MATLAB.

Problem

Write a program to check whether the matrix is hermitian or skew-hermitian

$$A = \begin{bmatrix} -i & -1 & 1-i \\ 1 & -i & -1 \\ -1-i & 1 & -i \end{bmatrix} \qquad B = \begin{bmatrix} 1 & 1+i & 2-3i \\ -1+i & 2i & 1 \\ -2-3i & -1 & 0 \end{bmatrix} \qquad C = \begin{bmatrix} 1 & 2+i & 3 \\ 2-i & 5 & 4 \\ 3 & 4 & 6 \end{bmatrix}.$$

Algorithm

Step1: Define a function, *check hermitian skew- hermitian ()* in the function file of the MATLAB program. This function takes input of the matrix from the user and then perform the required operations;
Step2: Inside the function give command to input the matrix;
Step3: To check the type of matrices, first we check whether it is square matrix or not;
Step4: Perform the required operations using the built-in function, *ishermitian* and display the result;

Program

```
function check_hermitian_skew_hermitian()
    % Ask the user to input the matrix A
    A = input('Enter the matrix A: ');
    % Get the number of rows and columns
    [m, n] = size(A);
    % Check if the matrix A is square
    if m ~= n
        error('Matrix A must be square to check if it is Hermitian or skew-Hermit
    end
    % Check if the matrix is Hermitian
    if ishermitian(A)
        disp('The matrix A is Hermitian.');
    elseif ishermitian(A,"skew")
        disp('The matrix A is skew-Hermitian.');
    else
        disp('The matrix A is neither Hermitian nor skew-Hermitian.');
    end
end
```

Output

```
Enter the matrix A: [-i -1 1-i;1 -i -1;-1-i 1 -i]
The matrix A is skew-Hermitian.
check_hermitian_skew_hermitian
Enter the matrix A: [1 1+i 2-3i;-1+i 2i 1;-2-3i -1 0]
The matrix A is neither Hermitian nor skew-Hermitian.
check_hermitian_skew_hermitian
Enter the matrix A: [1 2+1i 3;2-1i 5 4;3 4 6]
The matrix A is Hermitian.
```

Practical No. 14

Aim

To solve the problem to find the basis of column space of the matrices using MATLAB.

Problem

Write a program creating a script file to find the basis of column space of the following matrix

$$\begin{bmatrix} 2 & 0 \\ 3 & 4 \\ 0 & 5 \end{bmatrix}.$$

Algorithm

Step1: Create script file named *COLSPACE*;
Step2: Specify the matrix for which we want to find the basis of column space;
Step3: Convert the matrix to symbolic form by command *sym(A)*;
Step4: Use command *colspace(A)* to find the basis of column space;
Step5: Display the basis for the basis of column space;

Program

```
% Ask the user to input the matrix A
A = [2 0; 3 4; 0 5];
disp('The matrix A is:');
disp(A);

% Compute the symbolic form of A
A = sym(A);

% Extract the pivot columns from the original matrix A
col_space_basis = colspace(A);

% Display the result
disp('The basis for the column space of the matrix A is:');
disp(col_space_basis);
```

Output

```
The matrix A is:
     2     0
     3     4
     0     5

The basis for the column space of the matrix A is:
[    1,   0]
[    0,   1]
[-15/8, 5/4]
```

Practical No. 15

Aim

To solve the problem to find the row space of the matrices using MATHEMATICA.

Problem

Write a program to find row space of the following matrix

$$\begin{bmatrix} 2 & -1 & 3 \\ 1 & 0 & 1 \\ 0 & 2 & -1 \\ 1 & 1 & 4 \end{bmatrix}.$$

Algorithm

Step1: Specify the matrix for which we want to find the row space;
Step2: Use command *RowReduce* to transform the matrix into its row echelon form;
Step3: Select the rows from the reduced matrix that are not all zeros, as these rows form the basis for the row space;
Step4: Display the basis for the row space.

Program

```
matrix = {{2, -1, 3}, {1, 0, 1}, {0, 2, -1}, {1, 1, 4}};
reduced matrix = RowReduce[matrix];
rowSpaceBasis = Select[reducedMatrix, # != {0, 0, 0} &];

Print["Row Reduced Echelon Form of the matrix:"];
Print[MatrixForm[reducedMatrix]];

Print["The row space of the matrix is:"];
Print[MatrixForm[rowSpaceBasis]];
```

Output

```
Row Reduced Echelon Form of the matrix:
1   0   0
0   1   0
0   0   1
0   0   0
```

```
The row space of the matrix is:
1   0   0
0   1   0
0   0   1
```

Practical No. 16

Aim

To solve the problem to verify the Cayley-Hamiltons Theorem using MATLAB.

Problem

Write a program by creating a script file to verify Cayley-Hamiltons Theorem for the following matrix

$$\begin{bmatrix} 2 & -1 & 3 \\ 1 & 0 & 1 \\ 0 & 2 & -1 \end{bmatrix}.$$

Algorithm

Step1: Create script file named **cayley check**;
Step2: Specify the matrix for which we want to verify the Cayley-Hamiltons Theorem;
Step3: Compute dimension and characteristic polynomial of matrix;
Step4: Use *for loop* to replace the matrix in verify the Cayley-Hamiltons Theorem;
Step5: Display the result;

Program

```
clear all
clc
disp("Verify the Cayley-Hamiltons theorem in MATLAB")

A = [2 -1 3; 1 0 1; 0 2 -1];

% DimA(1) = no. of Columns & DimA(2) = no. of Rows
DimA = size(A);
charp = poly(A);

% Substitute the matrix A in the characteristic polynomial
P = zeros(DimA);
```

```
for i = 1:(DimA(1) + 1)
    P = P + charp(i) * (A^(DimA(1) + 1 - i));
end

disp("Result of the Characteristic equation after substituting the Matrix itself = ")
disp(round(P))

if round(P) == 0
    disp("Therefore, Cayley-Hamilton theorem is verified")
end
```

Output

```
 Verify the Cayley-Hamiltons theorem in MATLAB

DimA =

     3     3


charp =

    1.0000   -1.0000   -3.0000   -1.0000

Result of the Characteristic equation after substituting the Matrix itself =
     0     0     0
     0     0     0
     0     0     0

Therefore, Caylay-Hamilton theorem is verified
```

Practical No. 17

Aim

To solve the problem to find the inverse by Cayley-Hamiltons Theorem using MATLAB.

Problem

Write a program by creating a script file to find the inverse of following matrix by the Cayley-Hamiltons Theorem

$$\begin{bmatrix} -5 & -4 & 2 \\ 4 & -5 & 2 \\ 2 & 2 & -8 \end{bmatrix} \qquad \begin{bmatrix} -5 & 1 & 2 \\ 4 & -3 & 2 \\ 2 & 1 & -4 \end{bmatrix}.$$

Algorithm

Step1: Create script file named *cayley inverse*;
Step2: Ask the user to enter the matrix for which we want to find the inverse by the Cayley-Hamiltons Theorem;
Step3: Compute the characteristic polynomial of the matrix and find the number of coefficients;
Step4: Use *for loop* to find inverse by the Cayley-Hamiltons Theorem;
Step5: Display the result with a condition to check if the matrix is singular or not;

Program

```
disp("Finding Inverse of a Square Matrix using Cayley Hamilton theorem in MATLAB"
A = sym(input('Enter the Matrix A: '));

% To find Coefficients of Characteristic Equation of Matrix 'A'
cf = charpoly(A);

% To find the Number of Coefficients in
% the Characteristic Equation of Matrix 'A'
n = length(cf);

% To find the Inverse of A
inverse = cf(1) * A^(n-2);
for i = 2:n-1
    inverse = inverse + cf(i) * A^(n-i-1);
end

% Checking whether |A| = 0 or not
if round(cf(n)) == 0
    disp('Inverse of A does not exist as it is a singular matrix.');
else
    inverse = inverse / (-cf(n));
    disp('Inverse of A: ');
    disp(inverse);
```

```
end
```

Output

```
Finding Inverse of a Square Matrix using Cayley Hamilton theorem in MATLAB
Enter the Matrix A: [-5 4 2;4 -5 2;2 2 -8]
Inverse of A does not exist as it is a singular matrix.
Finding Inverse of a Square Matrix using Cayley Hamilton theorem in MATLAB
Enter the Matrix A: [-5 1 2;4 -3 2;2 1 -4]
Inverse of A:
[-1,  -3/5,   -4/5]
[-2,  -8/5,   -9/5]
[-1, -7/10, -11/10]
```

Practical No. 18

Aim

To solve the problem to identify the type of system of equations using MATLAB.

Problem

Write a program to identify the consistency of following system of equation

$$2x - y + 3z = 0$$

$$x + z = 0$$

$$y - z = 0.$$

Algorithm

Step1: Create function file named *check consistency*;
Step2: Ask the user to enter the system of equation in matrix and vector form;
Step3: Form the augmented matrix, compute its row reduced echelon form using *reff*;
Step4: Use *for loop* to check for the consistency;
Step5: Display the result with a condition to check if the matrix is consistent or not.

Program

```
function check_consistency()
    % Ask the user to input the coefficient matrix A
    A = input('Enter the coefficient matrix A: ');
    % Ask the user to input the constant vector b
    b = input('Enter the constant vector b: ');
    % Form the augmented matrix
    augmented_matrix = [A b];
    % Compute the reduced row echelon form of the augmented matrix
    R = rref(augmented_matrix);
    % Get the number of rows and columns of the augmented matrix
    [m, n] = size(R);
    % Check for inconsistency
    consistent = true;
    for i = 1:m
        if all(R(i, 1:end-1) == 0) && R(i, end) ~= 0
            consistent = false;
            break;
        end
    end
    % Display the result
    if consistent
        disp('The system of equations is consistent.');
    else
        disp('The system of equations is inconsistent.');
    end
end
```

Output

```
Enter the coefficient matrix A: [2 -1 3;1 0 1;0 1 -1]
Enter the constant vector b: [1; 2; 3]
The system of equations is consistent.
```

Practical No. 19

Aim

To solve the problem to identify the type of system of equations using MATLAB.

Problem

Write a program to identify the consistency and type of solution of the following system of equation

$$x + 2y + 3z = 1$$

$$4x + 5y + 6z = 2$$

$$7x + 8y + 9z = 3.$$

Algorithm

Step1: Create function file named *check system consistency*;
Step2: Ask the user to enter the system of equation in matrix and vector form;
Step3: Form the augmented matrix and compute its row reduced echelon form using *reff*;
Step4: Use *for loop* to check for the consistency;
Step5: Finally, display the result with a condition to check if the matrix is consistent or not and type of solution it posses;
Step6: Now in command window type *check system consistency* and press Enter.

Program

```
function check_system_consistency()
    % Ask the user to input the coefficient matrix A
    A = input('Enter the coefficient matrix A: ');
    % Ask the user to input the constant vector b
    b = input('Enter the constant vector b: ');
    % Check if b is a column vector
    if isrow(b)
        b = b'; % Convert row vector to column vector if necessary
    end
    % Get the number of rows and columns of A
    [m, n] = size(A);
    % Check if the number of rows of A matches the number of elements in b
    if length(b) ~= m
        error('The number of rows in A must match the number of elements in b');
    end
    % Form the augmented matrix
    augmented_matrix = [A b];
    % Compute the reduced row echelon form of the augmented matrix
    R = rref(augmented_matrix);

    % Compute the rank of A and the augmented matrix
    rank_A = rank(A);
    rank_augmented = rank(augmented_matrix);

    % Determine consistency
    consistent = (rank_A == rank_augmented);

    % Display the result
    if consistent
        if rank_A == n
            disp('The system of equations is consistent and has a unique solution
        else
            disp('The system of equations is consistent and has infinitely many s
        end
    else
        disp('The system of equations is inconsistent.');
    end
end
```

Output

```
Enter the coefficient matrix A: [1 2 3;4 5 6;7 8 9]
Enter the constant vector b: [1 2 3]
The system of equations is consistent and has infinitely many solutions.
```

Practical No. 20

Aim

To solve the problem to determine the roots of equations using MATLAB.

Problem

Write a program to determine the positive and negative roots of the following equation

$$x^3 + 2y + 3z = 1.$$

Algorithm

Step1: Create function file named *descartes rule of signs*;
Step2: Ask the user to input the coefficients of the polynomial as a vector;
Step3: The function *count sign changes* counts the number of sign changes in the coefficients of the polynomial. This determines the possible number of positive real roots;
Step4: The coefficients are modified to represent the polynomial P(-x);
Step5: The function *count sign changes*, count the number of sign changes in the modified coefficients and determine the possible number of negative real roots;
Step6: The function *display possibilities*, displays the possible number of real roots based on the count of sign changes and decreasing by 2 each time;

Program

```
function descartes_rule_of_signs()
    % Ask the user to input the coefficients of the polynomial
    coeffs = input('Enter the coefficients of the polynomial as a vector: ');

    % Find the number of positive real roots
    pos_changes = count_sign_changes(coeffs);

    % Find the number of negative real roots
    neg_coeffs = coeffs .* ((-1).^(length(coeffs)-1:-1:0));
    neg_changes = count_sign_changes(neg_coeffs);

    % Display the results
    disp('Possible number of positive real roots:');
    display_possibilities(pos_changes);

    disp('Possible number of negative real roots:');
    display_possibilities(neg_changes);
end

function changes = count_sign_changes(coeffs)
    % Initialize the count of sign changes
    changes = 0;
    for i = 1:length(coeffs)-1
        if coeffs(i) * coeffs(i+1) < 0
            changes = changes + 1;
        end
    end
end

function display_possibilities(changes)
    % Display the possible number of real roots
    while changes >= 0
        disp(changes);
        changes = changes - 2;
    end
end
```

Output

```
Enter the coefficients of the polynomial as a vector: [1 -6 11 -6]
Possible number of positive real roots:
     3

     1

Possible number of negative real roots:
     0
```

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:

`Sqrt[16]` (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.

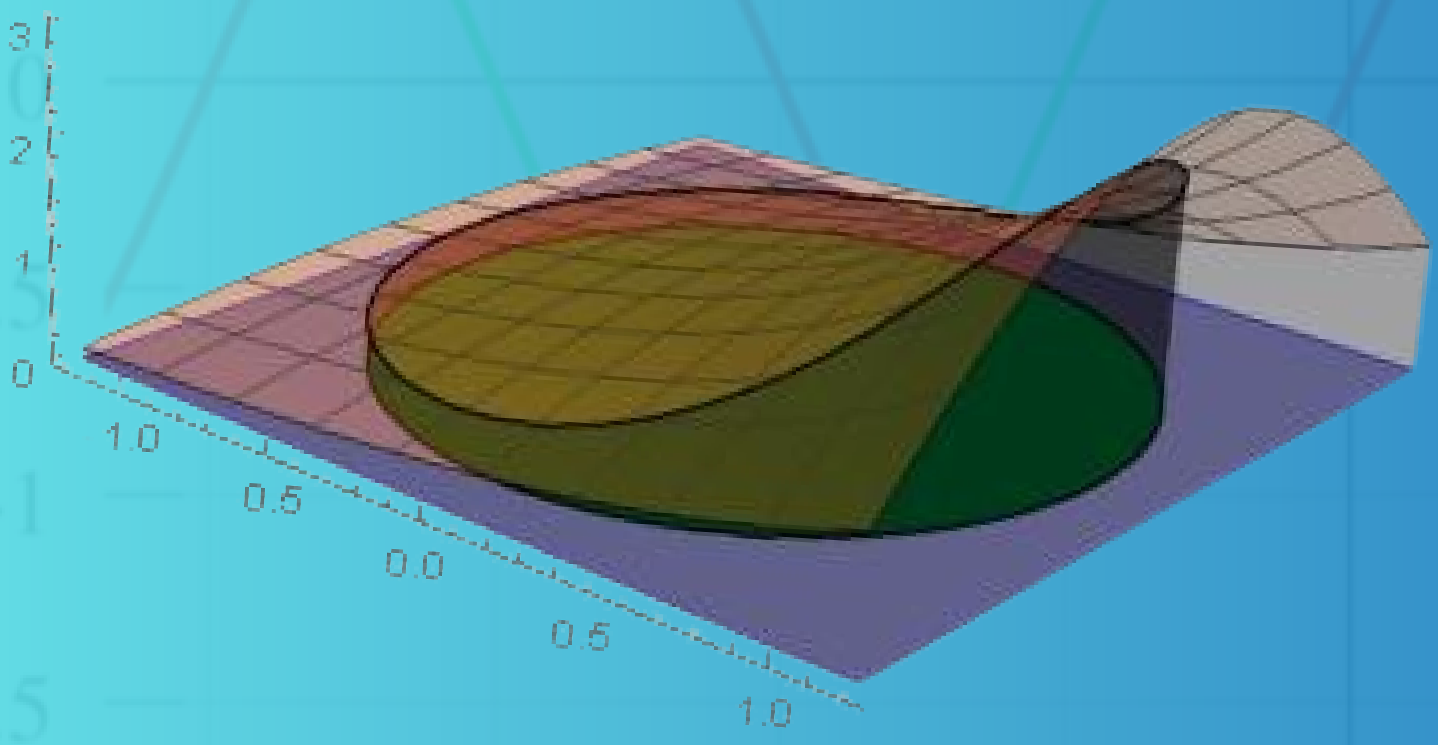
[8] Gilat, A. *MATLAB: An introduction with Applications.* John Wiley & Sons, 2017.

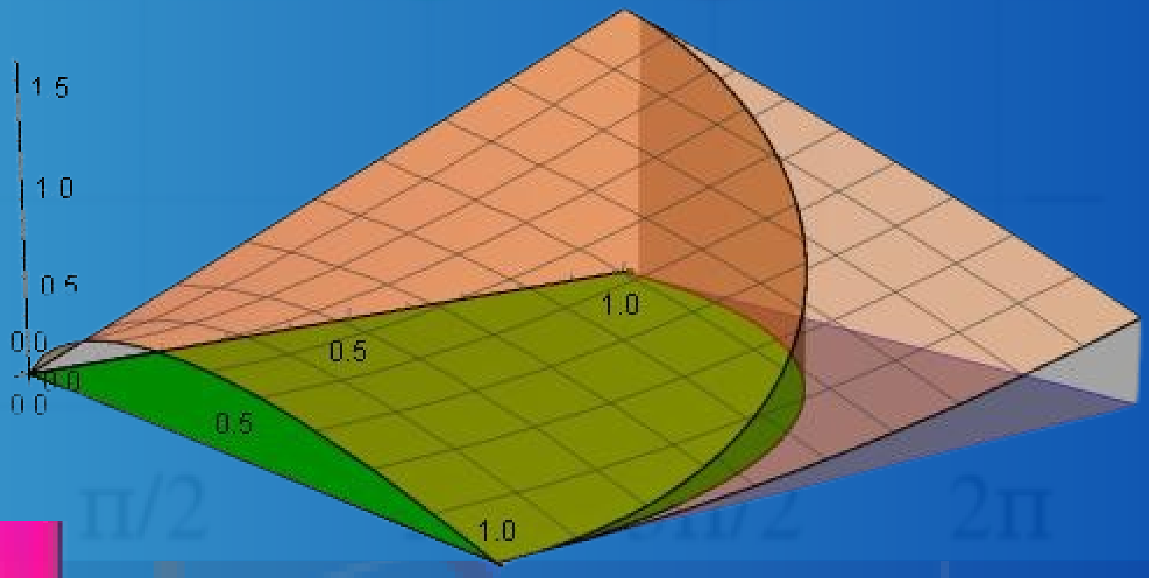
[9] Gowrishankar, S., and Veena, A. *Introduction to Python programming.* Chapman and Hall/CRC, 2018.

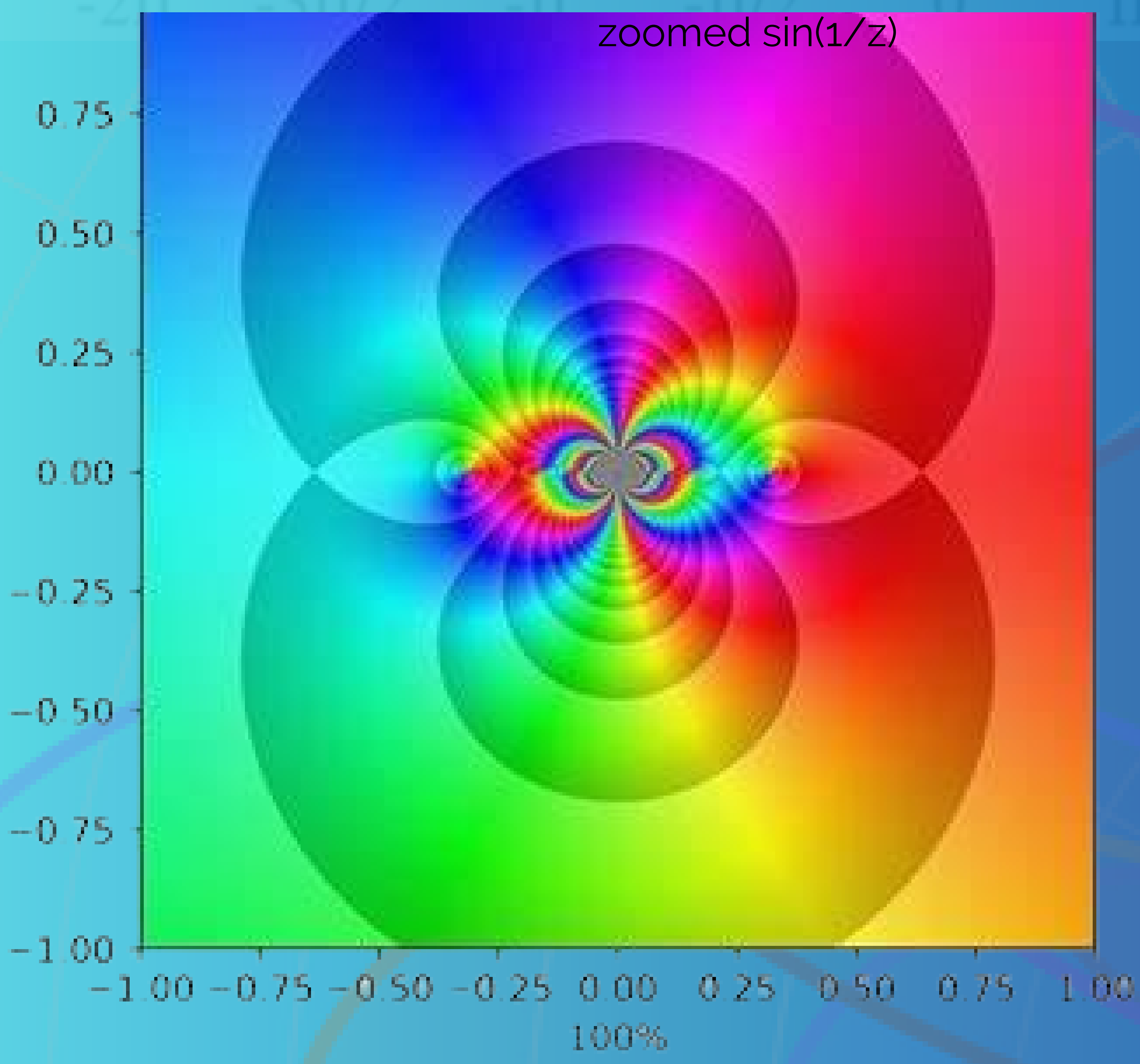
[10] Kattan, P. *Matlab for beginners*, vol. 1. Petra books, 2022.

[11] Lutz, M. *Programming python.* O'Reilly Media, Inc., 2001.

[12] Trott, M. *The Mathematica guidebook for programming.* Springer, 2013.







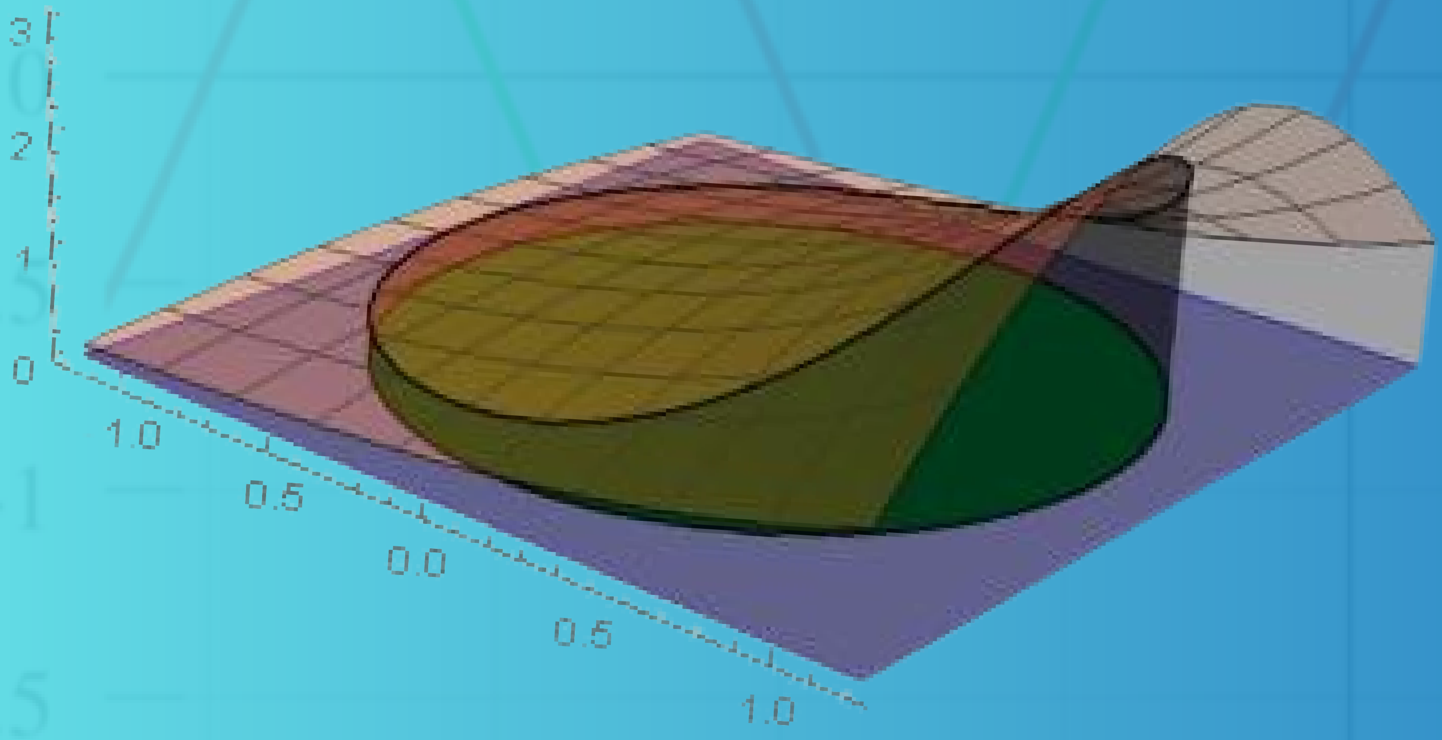
विद्यया ऽमृतमश्नुते

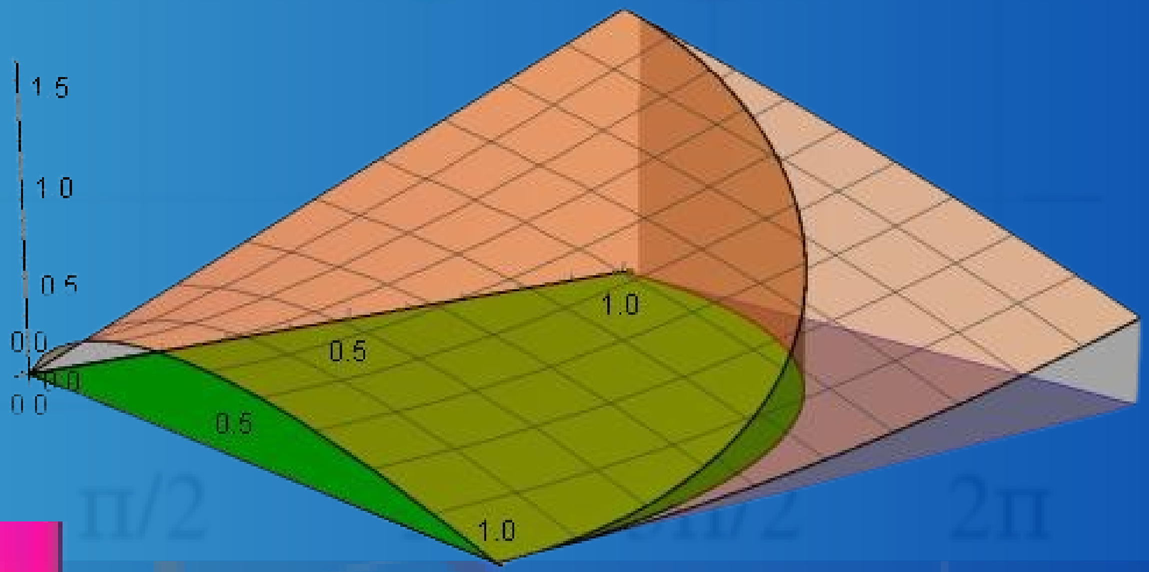
एन सी ई आर टी
NCERT

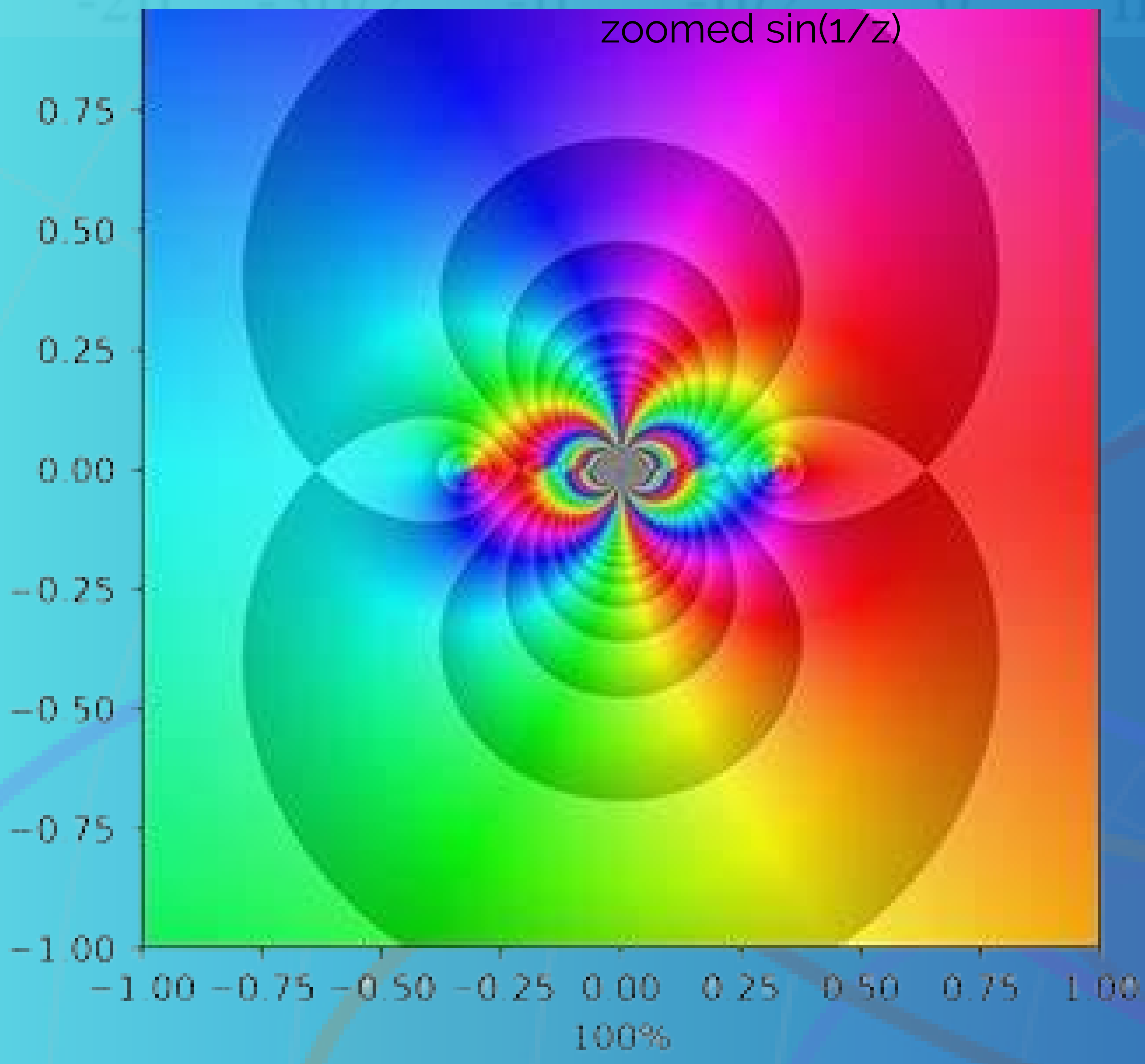
REGIONAL INSTITUTE OF EDUCATION

National Council of Educational Research and Training

Shyamala Hills, Bhopal, 102002







विद्यया ऽमृतमश्नुते

एन सी ई आर टी
NCERT

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