

1. Study on the listed MATLAB commands.

Command	Description
clc	Clears Command window.
clear	Removes variables from memory.
quit	Stops MATLAB.
delete	Deletes a file.
disp	Displays contents of an array or string.
fprintf	Performs formatted writes to screen or file.
input	Displays prompts and waits for input.
linspace	Creates regularly spaced vector.
grid	Displays gridlines.
plot	Generates xy plot.
title	Puts text at top of plot.
xlabel	Adds text label to x-axis.
ylabel	Adds text label to y-axis.
zlabel	Adds text label to z-axis.
close all	Closes all plots.
legend	Legend placement by mouse.
subplot	Creates plots in subwindows.
bar	Creates bar chart.
contour	Creates contour plot.
mesh	Creates three-dimensional mesh surface plot.
meshgrid	Creates rectangular grid.
syms	Creates one or more symbolic variables
close	Closes the current plot.
hold	Freezes current plot.
abs(x)	Absolute value; $ x $.

2. Write a MATLAB program to Area of a triangle where three sides are given.

Objective: Write a MATLAB program that calculates the area of a triangle given the lengths of its three sides using Herons Formula.

Introduction: Heron's formula is a mathematical formula used to calculate the area of a triangle when the lengths of all three sides are known.

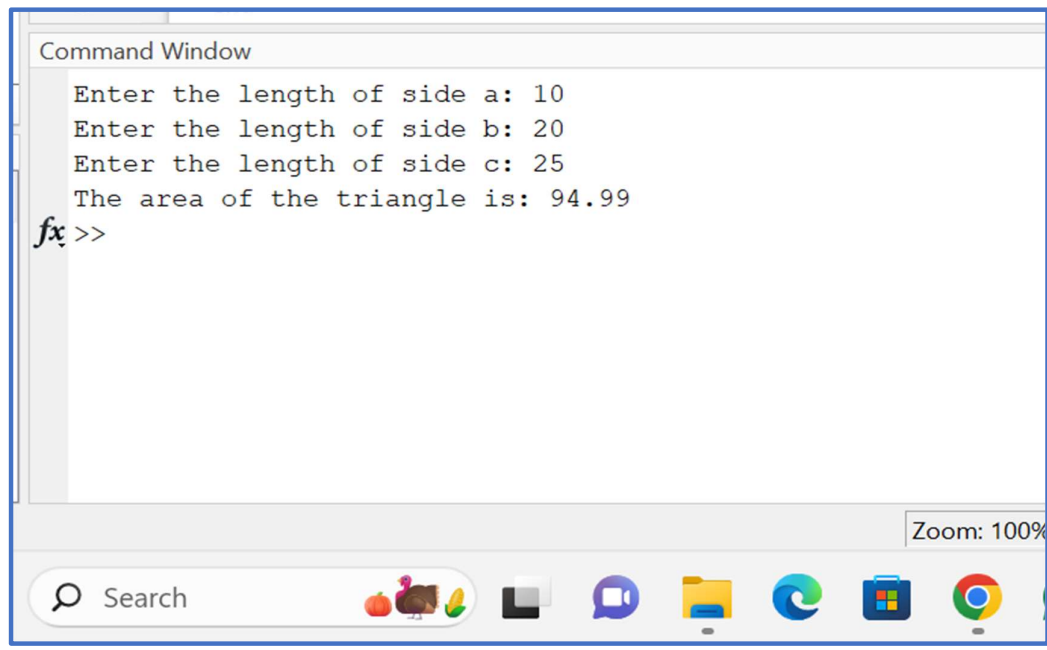
Algorithm:

1. Enter the lengths of the three sides of the triangle: a, b, and c.
2. Calculate Semi-perimeter using the formula: $s = (a + b + c) / 2$
3. Calculate Area $A = \sqrt{s(s-a)(s-b)(s-c)}$
4. Display the area of the triangle.

MatLab Code:

```
clc;
clear;
close all;
a = input('Enter the length of side a: ');
b = input('Enter the length of side b: ');
c = input('Enter the length of side c: ');
if (a + b > c) && (a + c > b) && (b + c > a)
s = (a + b + c) / 2;
area = sqrt(s * (s - a) * (s - b) * (s - c));
fprintf('The area of the triangle is: %.2f\n', area);
else
fprintf('The given sides do not form a valid triangle.\n');
end
```

Output:



```
Command Window

Enter the length of side a: 10
Enter the length of side b: 20
Enter the length of side c: 25
The area of the triangle is: 94.99
fx >>
```

Discussion:

The program waits for the user to input the three side lengths. s variable holds the semi-perimeter that is half of the sum of side lengths. Heron's formula is applied to calculate the area and store it in the area variable. Area is displayed on the console using the `fprintf` function.

3.(a) Write a MATLAB program to interchange the value of two numbers (with using third variable).

Objective: To write a MATLAB program to swap the values of two variables using a temporary variable.

Introduction:

Swapping variables is a common programming task. In this case, we'll use a third variable to temporarily store the value of one variable while assigning the value of the other to it.

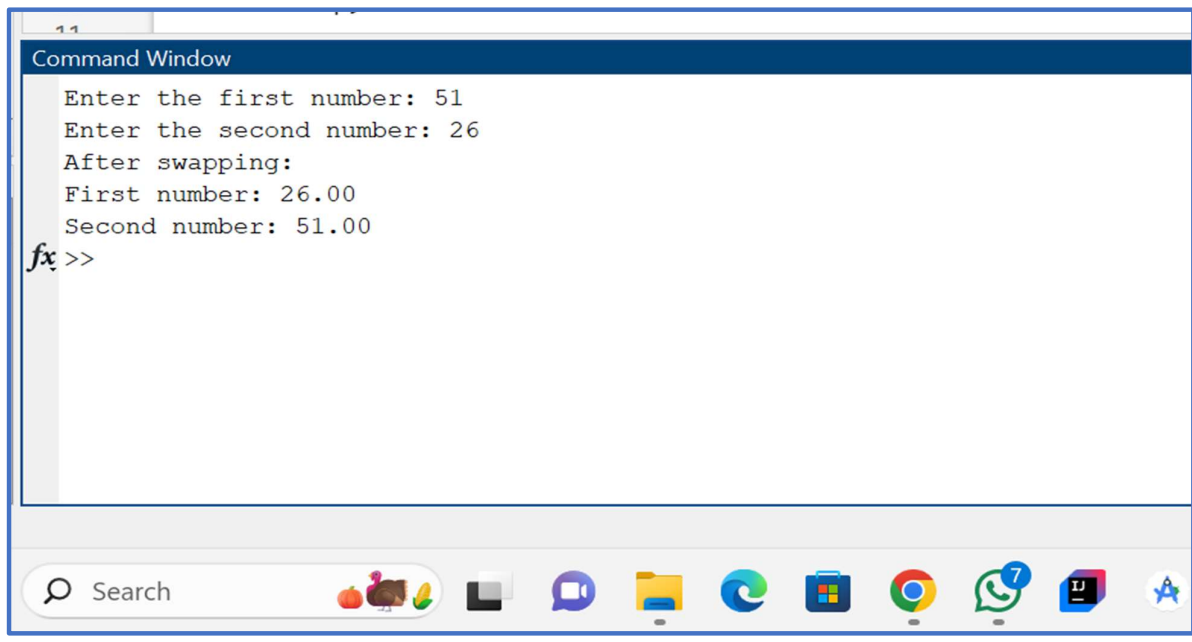
Algorithm:

- 1.input two numbers x and y.
- 2.Declare a temporary variable, temp to store values during the swapping process.
- 3.Swap values
- 4.Display the swapped values of x and y.

MatLab Code:

```
clc;
clear;
close all;
a= input('Enter the first number: ');
b = input('Enter the second number: ');
temp = a;
a = b;
b = temp;
fprintf('After swapping:\n');
fprintf('First number: %.2f\n',a);
fprintf('Second number: %.2f\n', b);
```

Output:



```
Command Window
Enter the first number: 51
Enter the second number: 26
After swapping:
First number: 26.00
Second number: 51.00
fx >>
```

The screenshot shows a Windows Command Window with a dark blue title bar. The text inside the window displays the execution of a program that swaps two numbers. The user inputs 51 and 26. The program outputs 'After swapping:', 'First number: 26.00', and 'Second number: 51.00'. Below the window, a portion of the Windows taskbar is visible, showing a search bar and several application icons including File Explorer, Microsoft Edge, and Google Chrome.

Discussion:

The program prompts the user to input two numbers, a and b. A temporary variable, temp is declared to store the value of a temporarily.

Swapping:

- Assign the value of a to temp.
- Assign the value of b to a.
- Assign the value of temp to b.

The final values of a and b after swapping are displayed on the console.

3.(b) Write a MATLAB program to interchange the value of two numbers (without using third variable).

Objective:

The objective of this program is to swap the values of two numbers without using a third variable in MATLAB.

Introduction: In many programming scenarios, it becomes necessary to swap the values of two variables. Usually, a third variable is used as a temporary storage location to accomplish the swap. But in this case we will swap two numbers without using any third variable by using basic arithmetic operations.

Algorithm:

1. Start
2. Input two numbers a and b.
3. Use arithmetic operations to swap the values:

$$a = a + b.$$

$$b = a - b$$

$$a = a - b$$

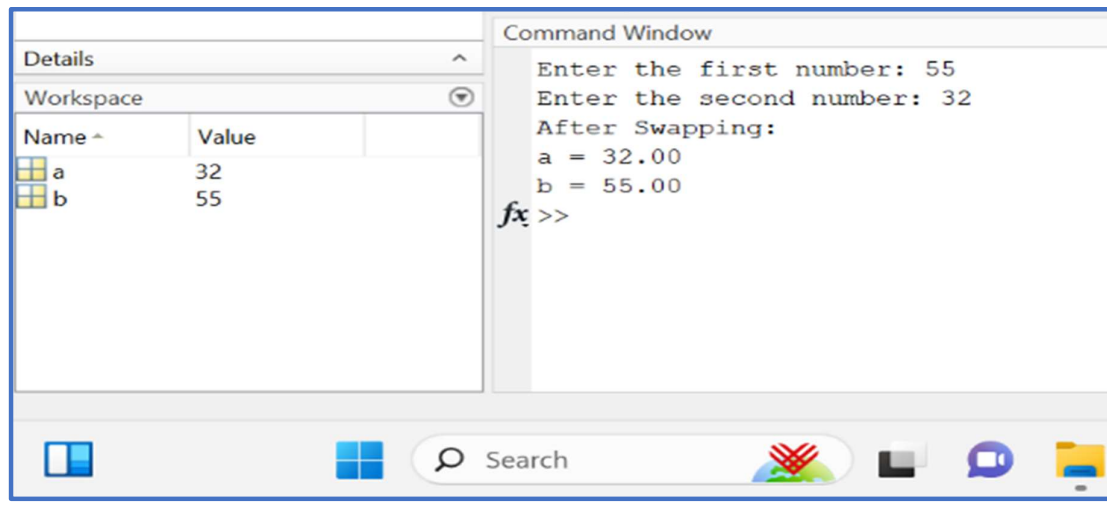
4. Display the swapped values of a and b.
5. End

MatLab Code:

```
clc;  
  
clear;  
  
close all;  
  
a = input('Enter the first number: ');  
b = input('Enter the second number: ');  
  
a = a + b;  
  
b = a - b;
```

```
a = a - b;  
  
fprintf('After Swapping:\n');  
  
fprintf('a = %.2f\n', a);  
  
fprintf('b = %.2f\n', b);
```

Output:



Discussion:

Without using a third variable This program successfully swaps the values of a and b without using an additional variable. It uses arithmetic operations (addition and subtraction) to store the sum and then retrieves the original values through subtraction. This method is space-efficient because it avoids the need for a third temporary variable.

4. Write a MATLAB program to sum of squares of n natural numbers.

Objective:

To compute the sum of the squares of the first n natural numbers using a MATLAB program

Introduction:

The sum of squares of the first n natural numbers is a common mathematical problem that arises in various fields like physics, statistics, and computer science. The formula for the sum of squares is:

$$\text{Sum}=1^2+2^2+3^2+\dots+n^2$$

In this experiment, MATLAB is used to calculate this sum iteratively and display the result.

Algorithm:

1. Start
2. Enter Input the number n.
3. Use a for loop to iterate through numbers from 1 to n.
4. Display the result.
5. End

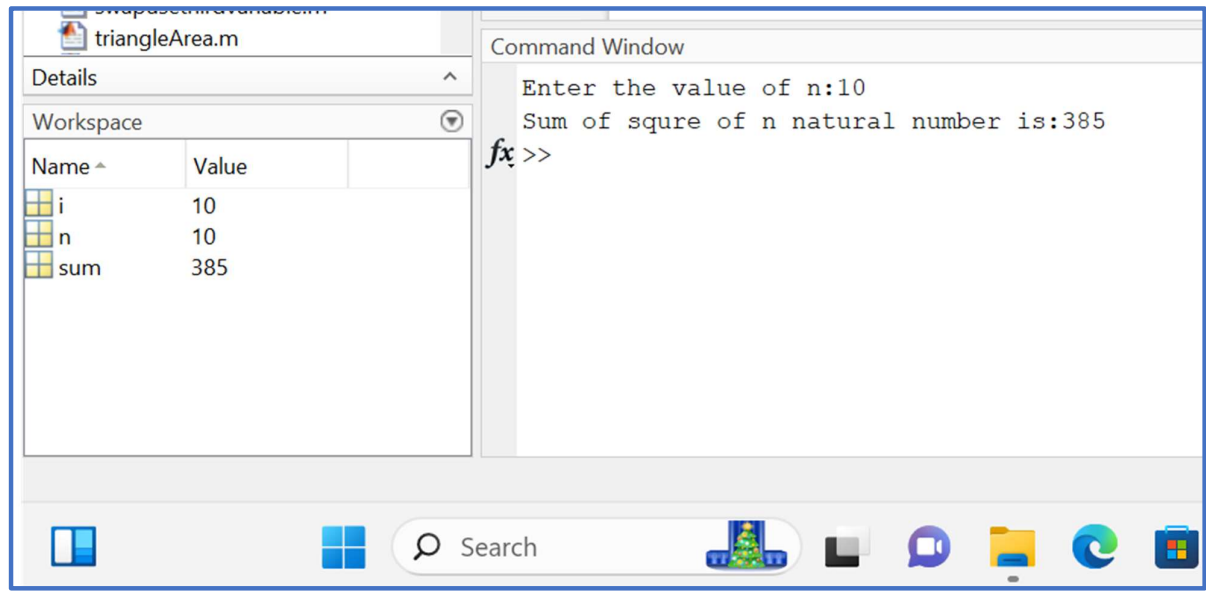
MatLab Code:

```
clc;  
clear;  
close all;  
n=input('Enter the value of n:');  
sum=0;  
for i=1:n  
    sum=sum+i*i;
```


end

```
fprintf("Sum of squire of n natural number is:%d\n",sum);
```

Output:



Discussion:

The code calculates the sum of squares of the first n natural numbers, useful in mathematics and applied sciences. The loop method is simple but can be replaced by a direct formula for faster computation. The code lacks input validation for negative. Works well for small to moderate n, but loops may slow down for large n.

5. Write a MATLAB program to plot the function $y=x^2$ for x ranging -10 to 10 label the axes and add a title to your plot.

Objective:

The goal of this program is to plot the quadratic function $y=x^2$ for x values ranging from -10 to 10, ensuring that the axes are labeled and the plot has a meaningful title.

Introduction:

Quadratic functions, such as $y=x^2$, are fundamental in mathematics and engineering. MATLAB provides powerful tools for plotting such functions. This program will calculate y values for x in the range of -10 to 10 and create a plot with properly labeled axes and a descriptive title. Graphs like these help to understand the behavior and properties of mathematical functions in a visual manner.

Algorithm:

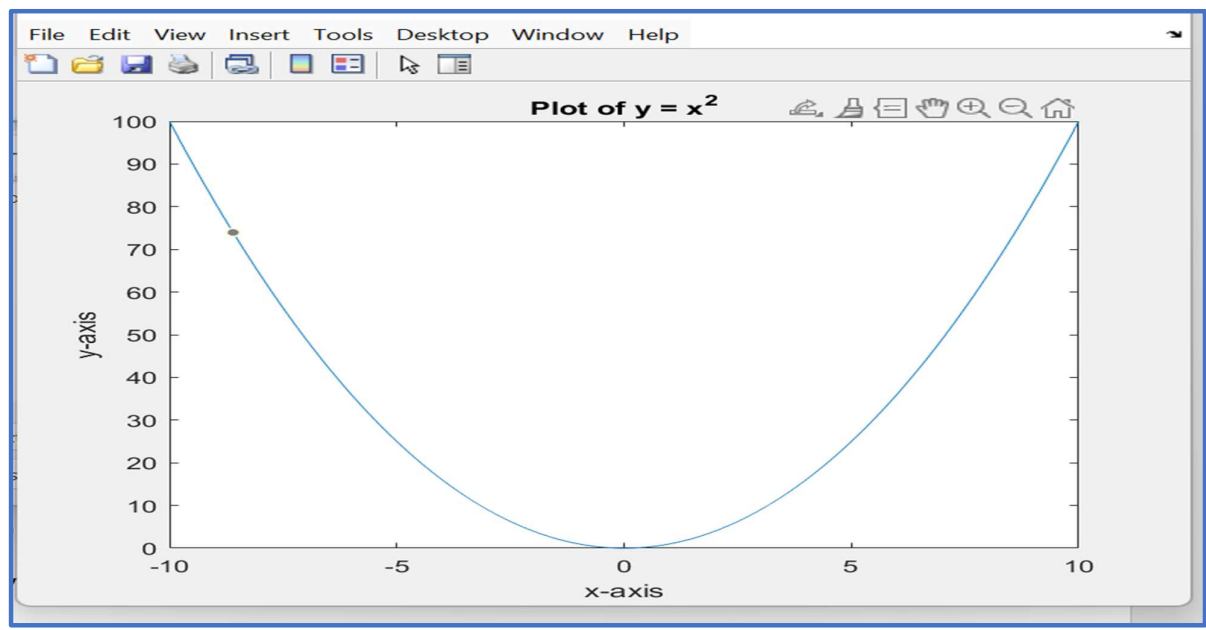
1. Start.
2. Define the range of x from -10 to 10, with small intervals for a smooth curve.
3. Compute the corresponding y-values using the formula $y=x^2$.
4. Use the plot function to create a 2D graph.
5. Add labels for the x-axis and y-axis using the xlabel and ylabel functions.
6. Add a title to the plot using the title function.
7. End.

MatLab Code:

```
clc;  
clear;  
close all;
```

```
x = -10:0.1:10;  
y = x.^2;  
plot(x, y);  
xlabel('x-axis');  
ylabel('y-axis');  
title('Plot of y = x^2');
```

Output:



Discussion:

This program effectively demonstrates how to create a basic plot in MATLAB. By plotting $y=x^2$, it illustrates the characteristic parabolic curve of a quadratic function. The range of x values and the smooth increments ensure that the graph is detailed and continuous. Labeling and adding a title make the graph more informative and suitable for interpretation in both academic and practical applications.

6. Write a MATLAB program to plot the sine and cosine functions on the same graph for x ranging from 0 to 2π use a legend to distinguish the two curves.

Objective:

To plot the sine and cosine functions on the same graph for x ranging from 0 to 2π and use a legend to distinguish between the two curves.

Introduction:

The sine and cosine functions are fundamental in trigonometry and appear frequently in mathematics, physics, and engineering. Visualizing these periodic functions helps in understanding their behavior, phase differences, and amplitude. In this experiment MATLAB is used to generate plots for these functions over a specific range. By using plotting tools we can visually compare the sine and cosine curves to explore their relationships and differences.

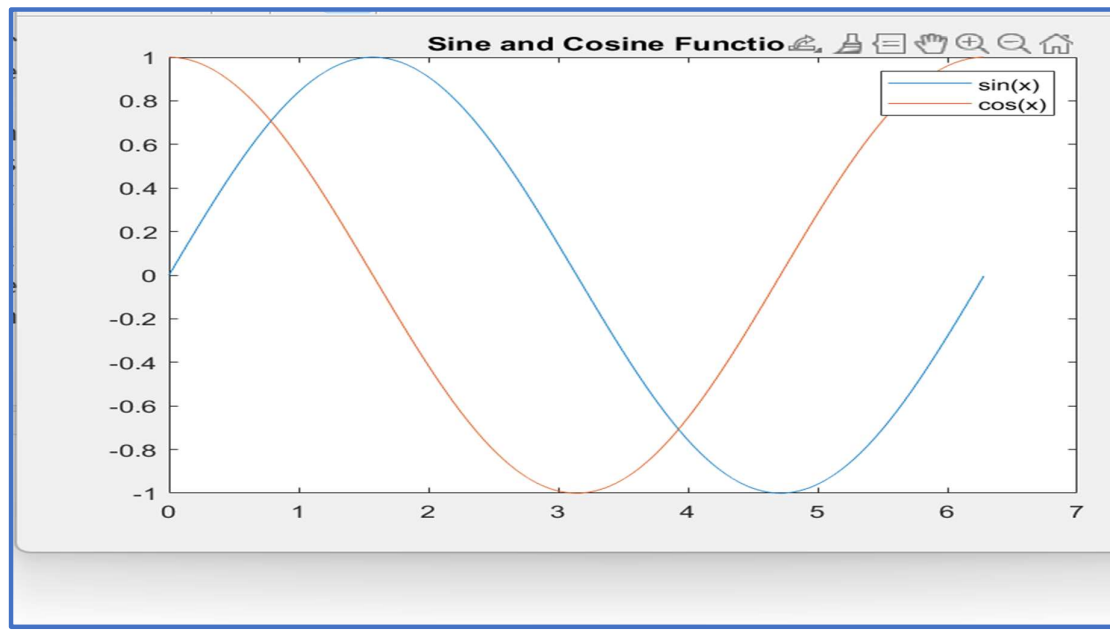
Algorithm:

1. Start
2. Define the Range of x as a vector ranging from 0 to 2π with increments of 0.01 for smooth curves.
3. Calculate $y=\sin(x)$ and $g=\cos(x)$
4. Plot the sine function
5. Use hold on to overlay the cosine plot on the same graph.
6. Plot the cosine function with a distinct color or style.
7. Add a title, label and use a legend to differentiate between sine and cosine functions.
8. End

MatLab Code:

```
clc;  
clear;  
close all;  
x=(0:0.01:2*pi);  
y=sin(x);  
g=cos(x);  
plot(x,y);  
hold on;  
plot(x,g);  
title('Sine and Cosine Functions');  
legend('sin(x)', 'cos(x)');
```

Output:



Discussion:

Using MATLAB, the sine and cosine functions were successfully plotted for x ranging from 0 to 2π . The resulting graph clearly demonstrates the periodic nature of these functions and their phase difference. The legend and labels enhance the clarity of the graph making it easier to analyze and interpret the behavior of the sine and cosine functions.

7. Write a MATLAB program to plot a bar chart for the following data $x=[1,2,3,4,5]$, $y=[10,20,15,25,30]$.

Objective:

To create a bar chart for given data $x=[1,2,3,4,5]$ and $y=[10,20,15,25,30]$ using MATLAB.

Introduction:

Bar charts are a common way to represent discrete data visually, where each bar's height corresponds to the value of the data point. we use MATLAB `bar()` function to plot the data x and y where x represents categories and y represents their corresponding values.

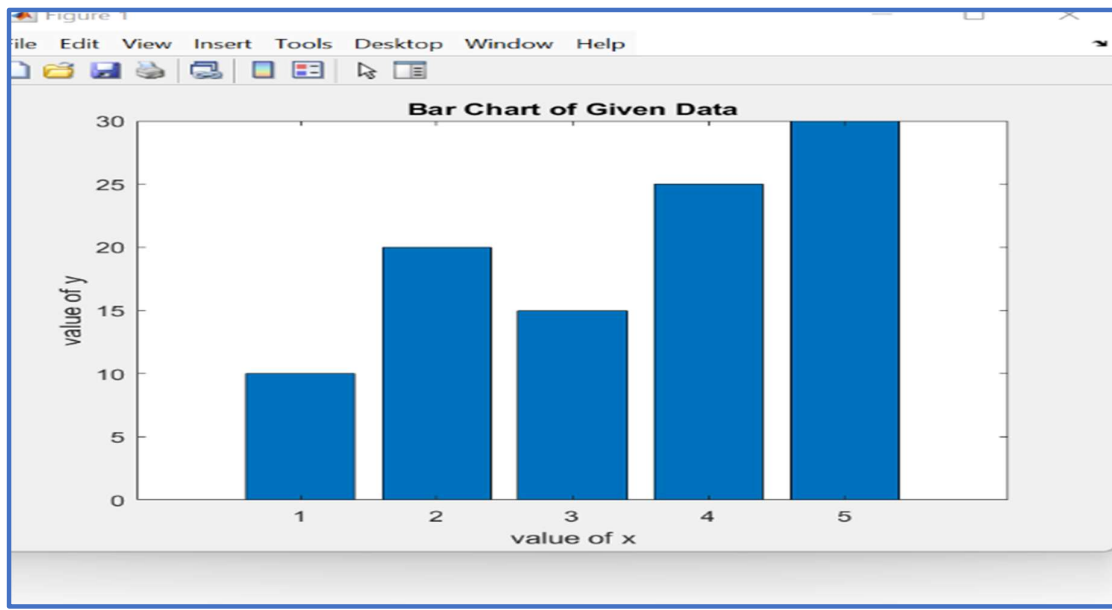
Algorithm:

- 1.Input the arrays $x= [1,2,3,4,5]$ and $y= [10,20,15,25,30]$
- 2.Use MATLAB `bar()` function to create the bar chart.
- 3.Add Labels and Title to the x axis and y axis and a title.
- 4.Display the final chart.

MatLab Code:

```
clc;  
clear;  
close all;  
x = [1, 2, 3, 4, 5];  
y = [10, 20, 15, 25, 30];  
bar(x, y);  
xlabel('value of x');  
ylabel('value of y');  
title('Bar Chart of Given Data');
```

Output:



Discussion: The bar chart effectively represents discrete data for x and y. Each bar's height corresponds to the value of y, making trends and comparisons clear. Adding labels, a title, and gridlines improves readability and interpretation. Bar charts are widely used in data analysis, statistics, and presentations.

8. Write a MATLAB program to create a mesh plot for $z=\sin(x)\cos(y)$ using mesh function.

Objective: To create a mesh plot for the function $z=\sin(x)\cos(y)$ using MATLAB mesh() function.

Introduction: A mesh plot is a 3D surface plot that represents a function of two variables $z=f(x, y)$. It uses a grid of x and y values to compute corresponding z values. In this task, we will use MATLAB mesh() function to visualize the behavior of $z=\sin(x)\cos(y)$ which is a periodic function.

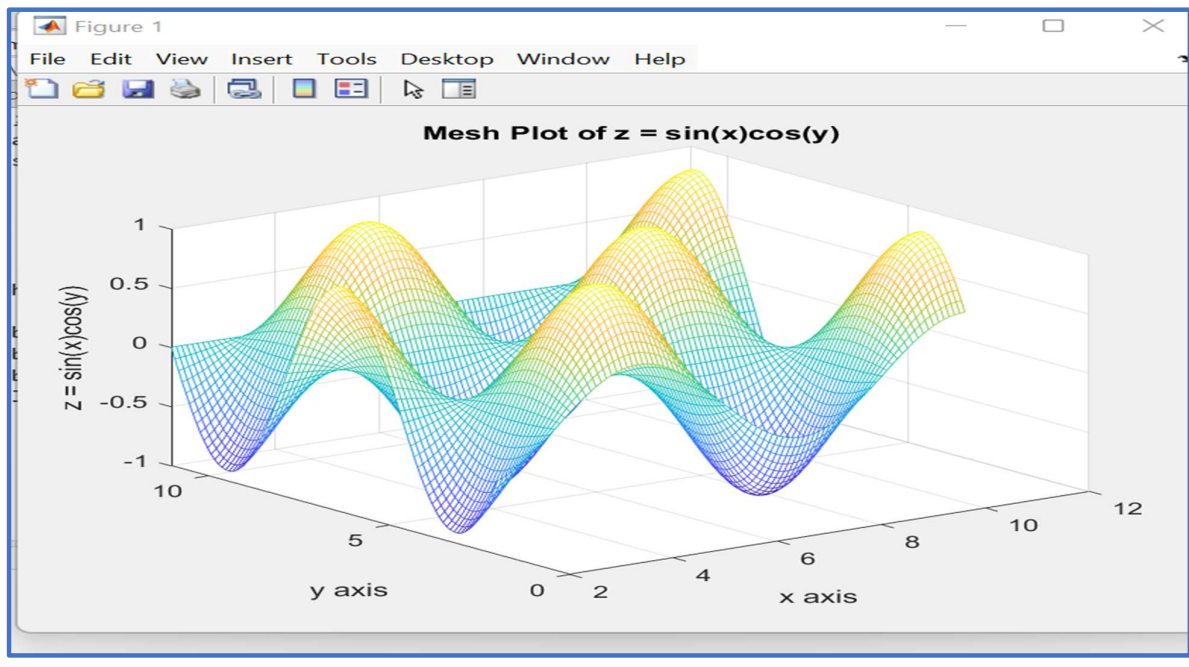
Algorithm:

1. Define the Ranges for x and y use meshgrid
2. Calculate $z=\sin(x)\cos(y)$
3. Create Mesh Plot Use the mesh() function.
4. Add title, axis labels.

MatLab Code:

```
clc;
clear;
close all;
[x, y] = meshgrid(2:.100:11);
z = sin(x) .* cos(y);
mesh(x, y, z);
xlabel('x axis');
ylabel('y axis');
zlabel('z = sin(x)cos(y)');
title('Mesh Plot of z = sin(x)cos(y)');
```


Output:



Discussion: A mesh plot provides a 3D visualization of the function $z = \sin(x)\cos(y)$ helping to analyze its periodic nature in two variables. The meshgrid function generates a grid of x and y values enabling efficient computation of z. The mesh() function displays the relationship between x, y and z as a 3D frame.

9. Write a MATLAB program to create a figure with four subplots:

- i. plot $y=x^2$ in the first subplot
- ii. plot $y=\sin(x)$ in the second subplot
- iii. plot $y=\cos(x)$ in the third subplot
- iv. plot $y=\tan(x)$ in the fourth subplot

Objective:

create a figure with four subplots in MATLAB plotting the following functions:

1. $y=x^2$

2. $y=\sin(x)$

3. $y=\cos(x)$

4. $y=\tan(x)$

Introduction:

Subplots in MATLAB allow multiple plots to be displayed in a single figure window. This is useful for comparing different functions visually. we will create a figure with four subplots each show a different mathematical function.

Algorithm:

- 1. Define the Range for x use meshgrid
- 2. Compute $y=x^2$, $y=\sin(x)$, $y=\cos(x)$ and $y=\tan(x)$.
- 3. Use subplot() to divide the figure into 4 sections and plot each function in its respective section.
- 4. Add titles, axis labels.

MatLab Code:

```
clc;

clear;

close all;

x=(-10:0.01:10);

subplot(2, 2, 1);

y1 = x.^2;

plot(x, y1);

title('y = x^2');

xlabel('x axis');

ylabel('y axis');

subplot(2, 2, 2);

y2 = sin(x);

plot(x, y2);

title('y = sin(x)');

xlabel('x axis');

ylabel('y axis');

subplot(2, 2, 3);

y3 = cos(x);

plot(x, y3);

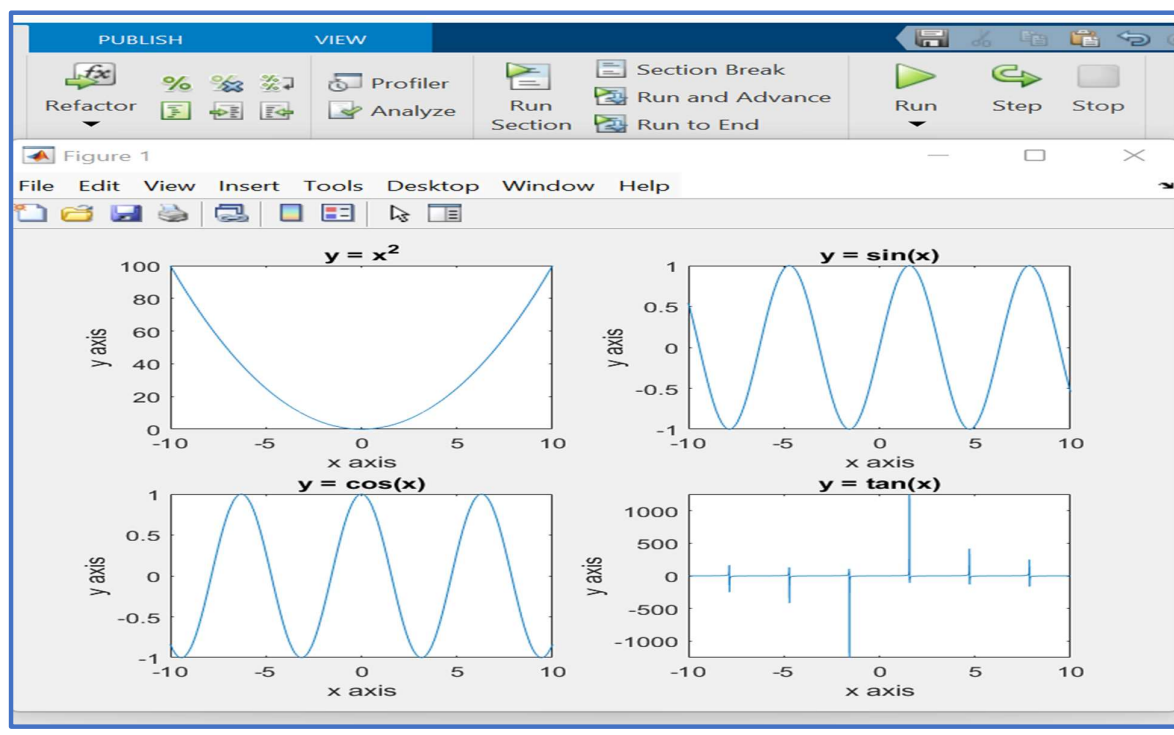
title('y = cos(x)');

xlabel('x axis');

ylabel('y axis');
```

```
subplot(2, 2, 4);  
  
y4 = tan(x);  
  
plot(x, y4);  
  
title('y = tan(x)');  
  
xlabel('x axis');  
  
ylabel('y axis');
```

Output:



Discussion: The program demonstrates how to use subplots in MATLAB to visualize multiple functions in a single figure.

*** Function Behavior:

1. $y=x^2$ Parabolic.
2. $y=\sin(x)$ Periodic.
3. $y=\cos(x)$ Periodic.
3. $y=\tan(x)$

Subplots enable easy comparison of the functions behavior. Useful for analyzing and presenting multiple datasets simultaneously in one view.

10. Write a MATLAB program to find out the root of a polynomial equation by using Bisection Method.

Objective: To find a root of a polynomial equation using the Bisection Method in MATLAB.

Introduction:

The Bisection Method is a numerical approach to find a root of a function $f(x)=0$. It works by iteratively narrowing the interval $[a,b]$ that contains the root based on the Intermediate Value Theorem. This method is simple and reliable.

Algorithm:

1. Choose two real number a and b such that $f(a)*f(b)<0$
2. Define Root $c=(a+b)/2$
3. Find $f(c)$
4. if $f(a)*f(c)<0$, Then $b=c$, else $a=c$

Return to step -1 untill find the root method twice.

MatLab Code:

```
clc;
clear;
close all;
f = @(x) x^3 -4*x-9;
```

```

a = 2;
b = 3;
tol = 1e-6;
max_iter = 100;
if f(a) * f(b) > 0
error('f(a) and f(b) must have opposite signs. please provide a valid interval.');
```

```

end

fprintf('Bisection Method:\n');
fprintf('Iteration\t a\t\t\t b\t\t\t c\t\t\t f(c)\n');
for i = 1:max_iter
c = (a + b) / 2;
f_c = f(c);

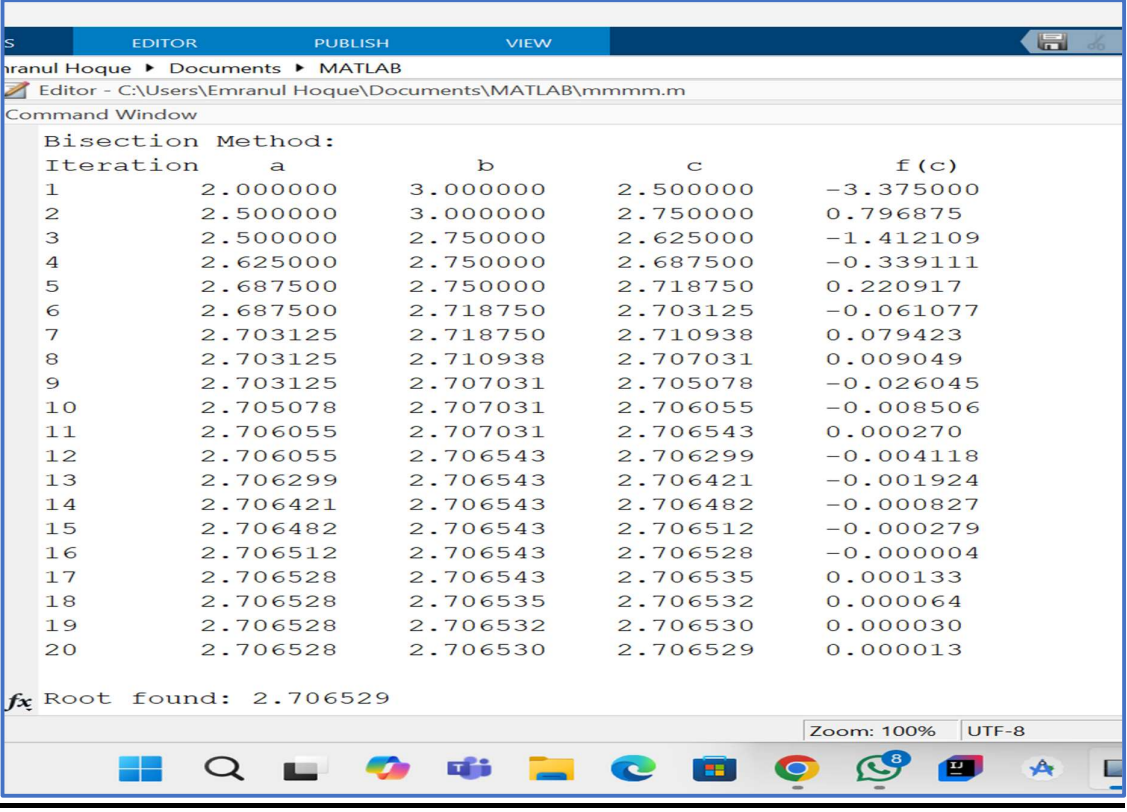
fprintf('%d\t\t %.6f\t %.6f\t %.6f\t %.6f\n', i, a, b, c, f_c);
if abs(f_c) < tol || (b - a) / 2 < tol
fprintf('\nRoot found: %.6f\n', c);
break;
end

if f(a) * f_c < 0
b = c;
else
a = c;
end

end
if i == max_iter
fprintf('\nMethod did not convergence within the maximum number of
iteration\n');
end

```

Output:



The image shows a MATLAB Command Window with the following output:

```
Bisection Method:
Iteration    a          b          c          f(c)
1           2.000000    3.000000    2.500000   -3.375000
2           2.500000    3.000000    2.750000    0.796875
3           2.500000    2.750000    2.625000   -1.412109
4           2.625000    2.750000    2.687500   -0.339111
5           2.687500    2.750000    2.718750    0.220917
6           2.687500    2.718750    2.703125   -0.061077
7           2.703125    2.718750    2.710938    0.079423
8           2.703125    2.710938    2.707031    0.009049
9           2.703125    2.707031    2.705078   -0.026045
10          2.705078    2.707031    2.706055   -0.008506
11          2.706055    2.707031    2.706543    0.000270
12          2.706055    2.706543    2.706299   -0.004118
13          2.706299    2.706543    2.706421   -0.001924
14          2.706421    2.706543    2.706482   -0.000827
15          2.706482    2.706543    2.706512   -0.000279
16          2.706512    2.706543    2.706528   -0.000004
17          2.706528    2.706543    2.706535    0.000133
18          2.706528    2.706535    2.706532    0.000064
19          2.706528    2.706532    2.706530    0.000030
20          2.706528    2.706530    2.706529    0.000013

fx Root found: 2.706529
```

The window also shows the file path: C:\Users\Emranul Hoque\Documents\MATLAB\mmmm.m and the zoom level: 100% UTF-8.

Discussion:

The program finds the root of a polynomial by iteratively reducing the interval containing the root. The interval $[a,b]$ must be chosen carefully to ensure $f(a)*f(b)<0$. The root is displayed with a specified tolerance along with the number of iterations.

11. Write a MATLAB program to find out the root of a polynomial equation by using False position Method.

Objective:

To determine the root of a polynomial equation using the False Position Method, a numerical technique based on linear interpolation.

Introduction:

The False Position Method is a numerical approach to find the root of an equation $f(x)=0$. It combines the benefits of the Bisection Method and linear interpolation. Unlike the Bisection Method, the False Position Method improves the interval by estimating the root using the equation of a straight line drawn between the two interval points where $f(a)*f(b)<0$.

Algorithm:

1. Choose two real number a and b such that $f(a)*f(b) \leq 0$
2. Define Root $c = a*f(b) - b*f(a) / f(b) - f(a)$
3. Find $f(c)$
4. if $f(a)*f(c) < 0$, Then $b=c$, else $a=c$

Return to step -1 until find the root method twice.

MatLab Code:

```
clc;  
clear;  
close all;  
f = @(x) x^3 - 2*x-5;  
a = 2;  
b = 3;  
tol = 1e-6;
```



```

max_iter = 100;
if f(a) * f(b) > 0
error('f(a) and f(b) must have opposite signs. Please provide a valid interval.');
```

end

```

fprintf('False-Position Method:\n');
fprintf('Iteration\t a\t\t\t b\t\t\t c\t\t\t f(c)\n');
for i = 1:max_iter
c=(a*f(b)-b*f(a))/(f(b) - f(a));
f_c = f(c);
fprintf('%d\t\t %.6f\t %.6f\t %.6f\t %.6f\n', i, a, b, c, f_c);
if abs(f_c) < tol
fprintf('\nRoot found: %.6f\n', c);
break;
end
if f(a) * f_c < 0
b = c;
else
a = c;
end
end
if i == max_iter
fprintf('\nMethod did not converge within the maximum number of
iterations.\n');
```


end

Output:

False-Position Method:				
Iteration	a	b	c	f(c)
1	2.000000	3.000000	2.058824	-0.390800
2	2.058824	3.000000	2.081264	-0.147204
3	2.081264	3.000000	2.089639	-0.054677
4	2.089639	3.000000	2.092740	-0.020203
5	2.092740	3.000000	2.093884	-0.007451
6	2.093884	3.000000	2.094305	-0.002746
7	2.094305	3.000000	2.094461	-0.001012
8	2.094461	3.000000	2.094518	-0.000373
9	2.094518	3.000000	2.094539	-0.000137
10	2.094539	3.000000	2.094547	-0.000051
11	2.094547	3.000000	2.094550	-0.000019
12	2.094550	3.000000	2.094551	-0.000007
13	2.094551	3.000000	2.094551	-0.000003
14	2.094551	3.000000	2.094551	-0.000001

Root found: 2.094551

fx >>



Discussion:

The program finds the root by iteratively refining the interval using linear interpolation. Each iteration calculates a new c based on the relative values of $f(a)$ and $f(b)$. The interval is updated to maintain $f(a)*f(b)<0$.

12. Write a MATLAB program to find out the root of a polynomial equation by using Fixed-point iteration Method.

Objective:

To find the root of a given polynomial equation $f(x) = x^3 - x - 1$ using the Fixed-Point Iteration Method.

Introduction:

The Fixed-Point Iteration Method is a numerical technique to approximate the root of an equation. The method involves rewriting $f(x)=0$ into the form $x=g(x)$ and iteratively solving for x . The iteration is governed by:

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

The method converges to a root if the function $g(x)$ satisfies certain conditions such as continuity and $|g'(x)| < 1$ near the root.

Algorithm:

1. Given an Equation $f(x)=0$
2. Convert an Equation $f(x)=0$ into the form of $x=g(x)$
3. Let the initial guess be 0.5 or 3.2 [(log, sin, cos, tan, e^x) in the equation then we use 3.2 otherwise 0.5]

4. Do

$$X_{i+1}=g(x_i)$$

While

(none of the convergence iteration is matched).

MatLab Code:

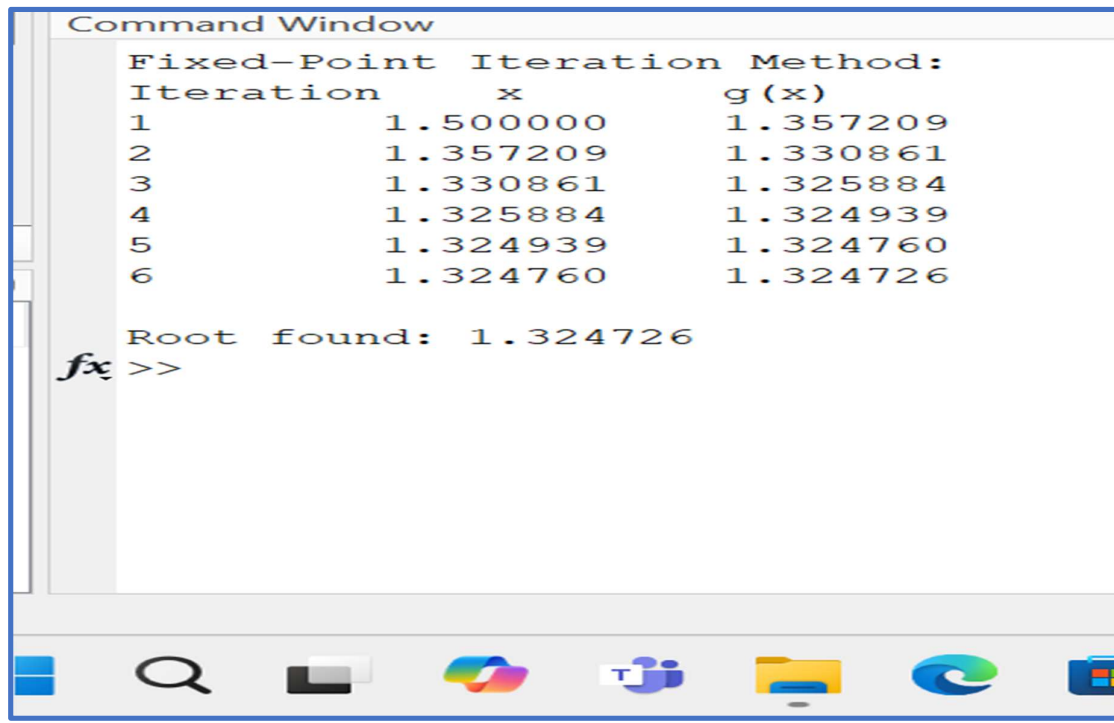
```
clc ;  
clear;  
close all;  
g = @(x) (x + 1)^(1/3);
```

```

x0 = 1.5;
tol = 0.0001;
max_iter = 100;
fprintf('Fixed-Point Iteration Method:\n');
fprintf('Iteration\t x\t\t g(x)\n');
for i = 1:max_iter
    x1 = g(x0);
    fprintf('%d\t\t %.6f\t %.6f\n', i, x0, x1);
    if abs(x1 - x0) < tol
        fprintf('\nRoot found: %.6f\n', x1);
        break;
    end
    x0 = x1;
end
if i == max_iter
    fprintf('\nMethod did not converge within the maximum number of
iterations.\n');
end

```

Output:



The image shows a MATLAB Command Window with the title 'Command Window'. It displays the output of a Fixed-Point Iteration Method. The output is a table with three columns: 'Iteration', 'x', and 'g(x)'. The table shows six iterations. The values of 'x' and 'g(x)' converge to approximately 1.324726. Below the table, it says 'Root found: 1.324726'. At the bottom left of the window, there is a prompt 'fx >>'.

```
Command Window

Fixed-Point Iteration Method:
Iteration      x          g(x)
1             1.500000    1.357209
2             1.357209    1.330861
3             1.330861    1.325884
4             1.325884    1.324939
5             1.324939    1.324760
6             1.324760    1.324726

Root found: 1.324726
fx >>
```

Discussion:

The Fixed-Point Iteration Method is a numerical technique to find roots of equations by rewriting $f(x)=0$ into $x=g(x)$. Starting with an initial guess x_0 successive approximations are calculated using $x_{n+1}=g(x_n)$ until the error is within a predefined tolerance. Simple to implement, requires fewer computations, and is intuitive.

13. Write a MATLAB program to find out the root of a polynomial equation by using Newton Raphson Method.

Objective:

To find the root of a given polynomial equation $f(x) = x^3 - x - 1$ using the Newton-Raphson Method.

Introduction: The Newton-Raphson Method is a numerical technique used to find roots of a nonlinear equation. It involves approximating the root iteratively based on the formula:

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

$f(x)$ is the given function and $f'(x)$ is its derivative. The process starts with an initial guess x_0 and continues until the error between successive approximations is smaller than a specified tolerance.

Algorithm:

1. Find $f'(x)$
2. find a & b so that $f(a) \cdot f(b) < 0$
3. Assume $x_0 = a$
4. Find out $x_{n+1} = x_n - f(x_n)/f'(x_n)$
5. Find x_1, x_2, \dots, x_n until any two successive value are equal.

MatLab Code:

```
clc;  
clear;  
close all;  
f = @(x) x^3-x-1;  
df = @(x) 3*x^2-1;  
x0 = 1.5;  
tol = 0.0001;
```

```

max_iter = 100;
fprintf('Newton-Raphson Method:\n');
fprintf('Iteration\t x\t\t f(x)\n');
for i = 1:max_iter
    f_x0 = f(x0);
    df_x0 = df(x0);
    if abs(df_x0) < 1e-12
        error('Derivative is too small. Method fails.');
```

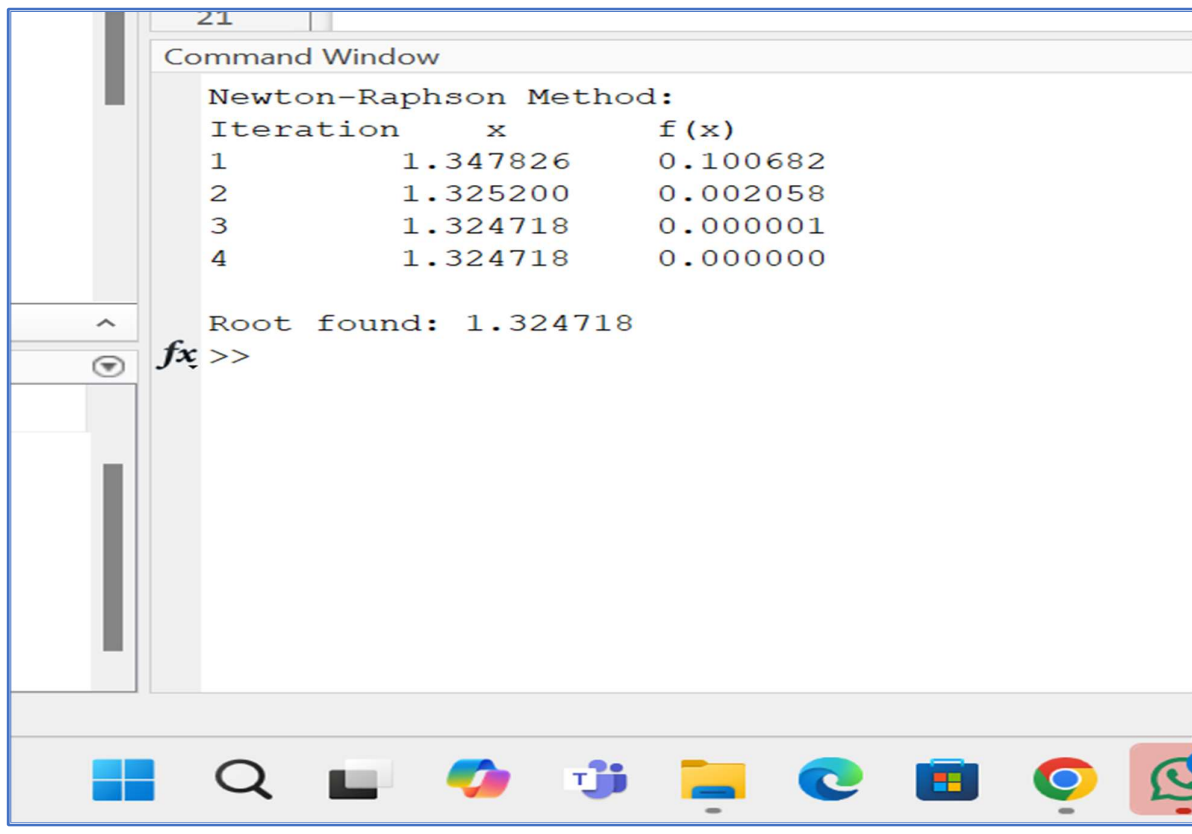
end

```

    x1 = x0 - f_x0 / df_x0;
    fprintf('%d\t\t %.6f\t %.6f\n', i, x1, f(x1));
    if abs(x1 - x0) < tol
        fprintf('\nRoot found: %.6f\n', x1);
        break;
    end
    x0 = x1;
end
if i == max_iter
    fprintf('\nMethod did not converge within the maximum number of
iterations.\n');
```

end

Output:



The screenshot shows a MATLAB Command Window titled "Command Window" with a tab labeled "21". The window displays the output of the Newton-Raphson method. The text "Newton-Raphson Method:" is followed by a table with three columns: "Iteration", "x", and "f(x)". The table contains four rows of data. Below the table, the text "Root found: 1.324718" is displayed. The prompt "fx >>" is visible at the bottom of the window. The Windows taskbar is visible at the bottom of the screen, showing icons for the Start menu, Search, File Explorer, Microsoft Edge, and other applications.

```
21
Command Window

Newton-Raphson Method:
Iteration      x      f(x)
1      1.347826  0.100682
2      1.325200  0.002058
3      1.324718  0.000001
4      1.324718  0.000000

Root found: 1.324718
fx >>
```

Discussion:

The Newton-Raphson method converges quadratically, making it faster than other methods. It requires the derivative $f'(x)$ to be known and continuous. A good initial guess is critical, as a poor guess may lead to divergence or convergence to the wrong root. The method may fail if $f'(x)=0$ at any iteration.

14. Write MATLAB Program for Numerical Integration using Trapezoidal Rule.

Objective:

To approximate the definite integral of a function using the Trapezoidal Rule.

Introduction:

The trapezoidal rule is a numerical integration method used to approximate the value of a definite integral. It works by dividing the integral into small subintervals and approximating the area under the curve for each subinterval using a trapezoid. This method is particularly useful when the function is complex.

Algorithm:

1. Define the function to be integrated.
2. Enter the limits of integration a and b and the number of subintervals n.
3. Calculate h using this formula $h = (b - a) / n$.
4. Compute the function values at the endpoints and at each subinterval.
5. Apply the trapezoidal rule formula:
$$(h/2) * (f(a) + 2 * \text{sum} + f(b))$$
6. Display the approximate integral value.

MatLab Code:

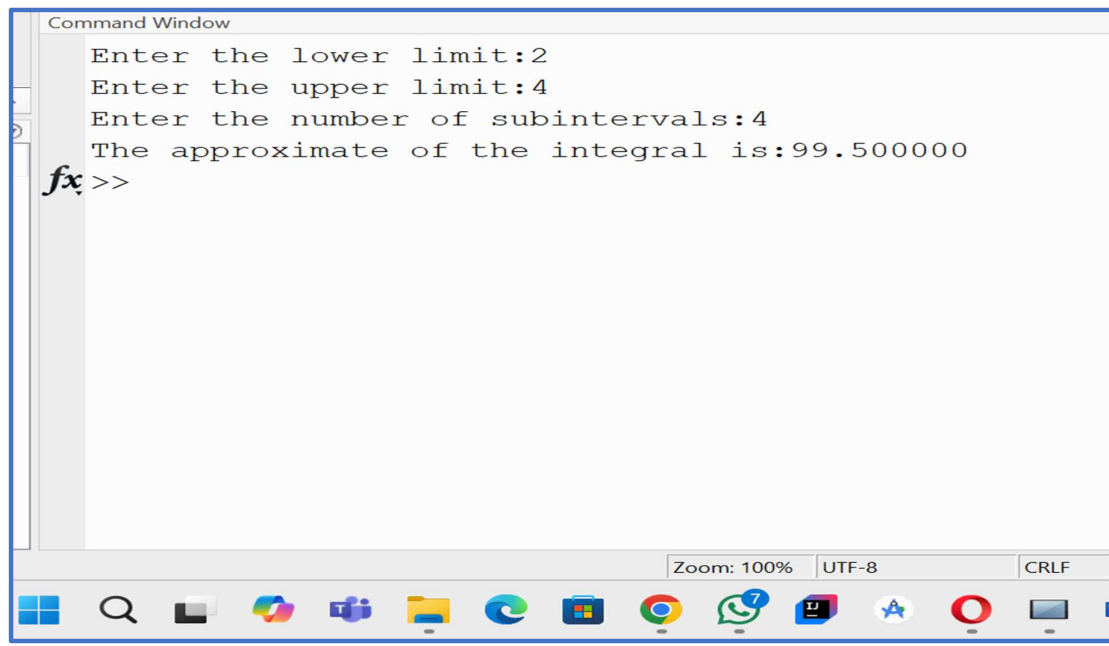
```
clc;
clear;
close all;
f=@(x) 2*x^3-4*x+1;
a=input('Enter the lower limit:');
b=input('Enter the upper limit:');
n=input('Enter the number of subintervals:');
```

```

if(n<0)
error('Number of subintervals must be greater than 0:');
end
h=(b-a)/n;
sum=0;
for i=1:n-1
x=a+i*h;
sum=sum+f(x);
end
result=(h/2)*(f(a)+2*sum+f(b));
fprintf('The approximate of the integral is:%.6f\n',result);

```

Output:



```

Command Window
Enter the lower limit:2
Enter the upper limit:4
Enter the number of subintervals:4
The approximate of the integral is:99.500000
fx >>

```

Zoom: 100% UTF-8 CRLF

Discussion:

- ❖ Limitations: Less accurate for functions with high discontinuities. Requires a larger number of subintervals for better accuracy compared to higher-order methods.
- ❖ Applications: The trapezoidal rule is commonly used in engineering, physics, and applied mathematics for problems where analytical integration is difficult.
- ❖ Accuracy: The accuracy of the trapezoidal rule depends on the number of subintervals n .

15. Write a MATLAB Program for Numerical Integration using Simpson's 1/3 Rule.

Objective:

To approximate the definite integral of a function using Simpson's 1/3 Rule.

Introduction:

Simpson's 1/3 Rule is a numerical integration method that provides a more accurate approximation of definite integrals than the trapezoidal rule. This method is particularly useful for smooth functions and requires the number of subintervals to be even.

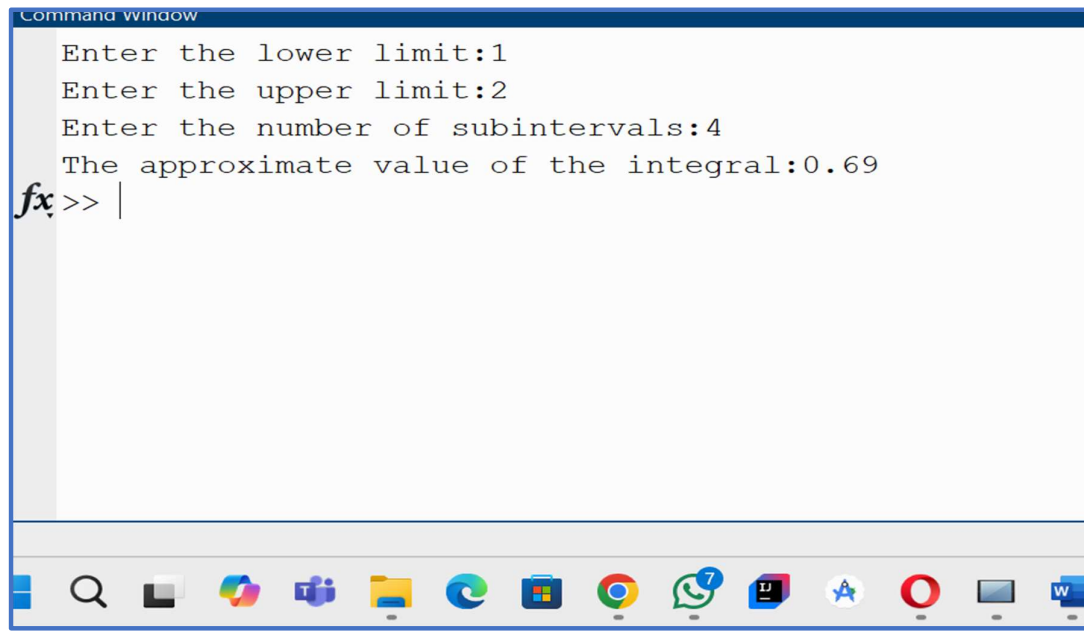
Algorithm:

1. Define the function to be integrated.
2. Specify the limits of integration a and b and the number of subintervals n is even.
3. Calculate h using this formula $h = (b - a) / n$
4. Compute the function values at the endpoints and subinterval points.
5. Apply Simpson's 1/3 Rule formula:
$$(h/3) * (f(a) + 4 * \text{sum_odd} + 2 * \text{sum_even} + f(b))$$
6. Display the approximate integral value.

MATLAB Code:

```
clc;
clear;
close all;
f=@(x) 1/x;
a=input("Enter the lower limit:");
b=input("Enter the upper limit:");
n=input("Enter the number of subintervals:");
if mod(n,2) ~=0
error('Number of subintervals must be even for simpsons 1/3 Rule');
end
h=(b-a)/n;
sum_odd=0;
sum_even=0;
for i=1:n-1
x=a+i*h;
if mod(i,2)==0
sum_even=sum_even+f(x);
else
sum_odd=sum_odd+f(x);
end
end
result=(h/3)*(f(a)+4*sum_odd+2*sum_even+f(b));
fprintf("The approximate value of the integral: %.2f\n",result);
```

Output:



```
Command window
Enter the lower limit:1
Enter the upper limit:2
Enter the number of subintervals:4
The approximate value of the integral:0.69
fx>> |
```

The screenshot shows a Windows Command Prompt window titled "Command window". The text inside the window displays the prompts and results of a program: "Enter the lower limit:1", "Enter the upper limit:2", "Enter the number of subintervals:4", and "The approximate value of the integral:0.69". Below the text, there is a prompt "fx>> |". The taskbar at the bottom of the window shows various application icons, including the Start menu, Search, File Explorer, Microsoft Edge, and several other programs.

Discussion:

- ❖ Accuracy: Simpson's 1/3 Rule is more accurate than the trapezoidal rule because it uses parabolic interpolation.
- ❖ Requirements: The method requires an even number of subintervals, which may not always be convenient.
- ❖ Applications: It is widely used in numerical integration for smooth functions
- ❖ Limitations: The method is less effective for functions with significant oscillations or discontinuities.

16. Write a MATLAB Program for Numerical Integration using Simpson's 3/8 Rule.

Objective:

To approximate the definite integral of a function using Simpson's 3/8 Rule.

Introduction:

Simpson's 3/8 Rule is a numerical integration method that approximates a definite integral by fitting a cubic polynomial over every three consecutive subintervals. It is an extension of Simpson's 1/3 Rule and is particularly useful when the number of subintervals n is a multiple of 3.

Algorithm:

1. Define the function to be integrated.
2. Enter the limits of integration a and b and ensure the number of subintervals n is a multiple of 3.
3. Calculate h using this formula $h = (b - a) / n$.
4. Compute the function values at the endpoints and subinterval points.
5. Apply Simpson's 3/8 Rule formula:
$$(3 \cdot h/8) \cdot (f(a) + 3 \cdot \text{sum_odd} + 2 \cdot \text{sum_even} + f(b))$$
6. Display the approximate integral value.

Matlab Code:

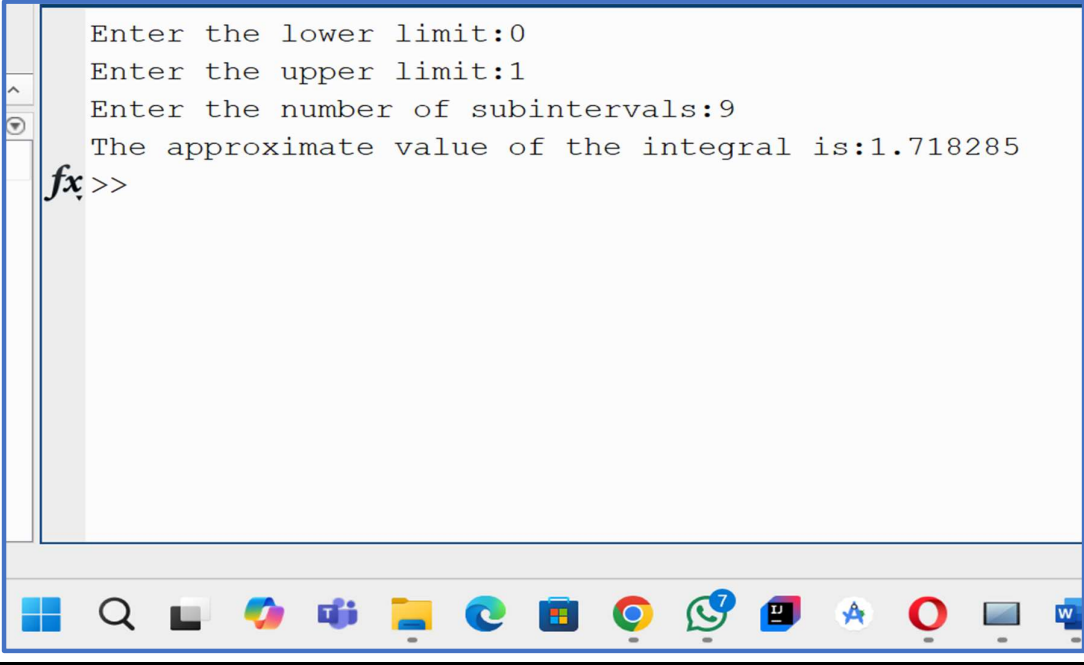
```
clc ;  
clear;  
close all;  
f=@(x) exp(x);  
a=input('Enter the lower limit:');  
b=input('Enter the upper limit:');
```

```

n = input('Enter the number of subintervals:');
if mod(n,3) ~=0
error('Number of subintervals must be a multiple of 3 for Simpsons 3/8 Rule');
end
h=(b-a)/n;
sum_odd = 0;
sum_even = 0;
for i= 1: n-1
x=a+i*h;
if mod(i,3) ==0
sum_even=sum_even+f(x);
else
sum_odd=sum_odd+f(x);
end
end
result=(3*h/8)*(f(a)+3*sum_odd+2* sum_even + f(b));
fprintf('The approximate value of the integral is:%.6f\n',result);

```

Output:



```

Enter the lower limit:0
Enter the upper limit:1
Enter the number of subintervals:9
The approximate value of the integral is:1.718285
fx >>

```

Discussion:

****Accuracy****: Simpson's 3/8 Rule provides a good approximation for definite integrals, especially for smooth functions.

****Requirements****: The method requires the number of subintervals to be a multiple of 3.

****Applications****: This rule is used when Simpson's 1/3 Rule cannot be applied due to the subinterval constraint or when slightly better accuracy is needed.

****Limitations****: Similar to other numerical integration methods, it may fail to provide accurate results for functions with discontinuities.