1. Study on the listed MATLAB commands.

- Problem Name: Study of Basic MATLAB Commands
- Objective: To familiarize with fundamental MATLAB commands for various operations.
- Introduction: MATLAB (Matrix Laboratory) is a high-level programming language and interactive environment for numerical computation, visualization, and[1] programming. This study explores essential commands for array manipulation, mathematical functions, plotting, and control flow.
- Method/Algorithm: Review of MATLAB documentation and experimentation with commands like `help`, `doc`, `clear`, `clc`, `linspace`, `zeros`, `ones`, `size`, `length`, basic arithmetic operators (+, -, *, /, ^), trigonometric functions (sin, cos, tan), exponential and logarithmic functions (exp, log), and plotting commands (plot, figure, hold on/off).
- MATLAB Code: This section would contain examples of using the commands, such as:

Matlab

```
help plot % Display help for the plot function
x = linspace(0, 2*pi, 100); % Create a vector of 100 evenly spaced points
y = sin(x);
plot(x, y); % Plot the sine function
```

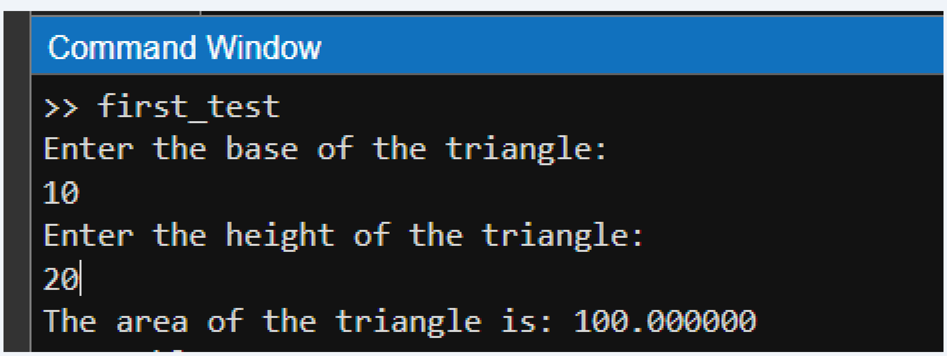
- Program Output: This would show the output of the example code, such as the plot of the sine wave.
- Discussion: This section would discuss the functionality of the commands and their usage in various applications.

2. Write a MATLAB program to find out the area of a triangle.

- Problem Name: Area of a Triangle
- Objective: To write a MATLAB program to calculate the area of a triangle given its base and height.
- Introduction: The area of a triangle is calculated using the formula: Area = (1/2) * base * height.
- Method/Algorithm: The program will take the base and height as input and apply the formula to calculate the area.
- MATLAB Code:

Matlab

```matlab
base = input('Enter the base of the triangle: ');
height = input('Enter the height of the triangle: ');
area = 0.5 * base * height;
fprintf('The area of the triangle is: %f\n', area);
```



- Program Output: The program will display the calculated area of the triangle.
- Discussion: This section would discuss the formula used and the input/output of the program.

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

- Problem Name: Swapping Two Numbers
- Objective: To write a MATLAB program to interchange the values of two numbers using both a temporary variable and without a temporary variable.
- Introduction: Swapping the values of two variables is a common programming task.
- Method/Algorithm:
    - *With a third variable:* Use a temporary variable to store one value while the swap occurs.
    - *Without a third variable:* Use arithmetic operations to achieve the swap.
- MATLAB Code:

Matlab

```
% With a third variable
a = input('Enter the first number: ');
b = input('Enter the second number: ');
temp = a;
a = b;
b = temp;
fprintf('After swapping (with temp): a = %f, b = %f\n', a, b);

% Without a third variable
a = input('Enter the first number: ');
b = input('Enter the second number: ');
a = a + b;
b = a - b;
a = a - b;
fprintf('After swapping (without temp): a = %f, b = %f\n', a, b);
```

```
Command Window
>> problem3
Enter the first number:
20
Enter the second number:
30
After swapping (with temp): a = 30.000000, b = 20.000000
Enter the first number:
20
Enter the second number:
30
After swapping (without temp): a = 30.000000, b = 20.000000
```

- Program Output: The program will display the values of the variables before and after swapping.
- Discussion: This section would compare the two methods and discuss their efficiency.

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

- Problem Name: Sum of Squares of Natural Numbers
- Objective: To write a MATLAB program to calculate the sum of the squares of the first n natural numbers.
- Introduction: The sum of squares of the first n natural numbers is given by the formula: Σ(i^2) = n(n+1)(2n+1)/6.
- Method/Algorithm: The program will take n as input and either use a loop or the formula to calculate the sum.
- MATLAB Code:

Matlab

```
n = input('Enter the value of n: ');
sum_of_squares = n * (n + 1) * (2 * n + 1) / 6;
fprintf('The sum of squares of the first %d natural numbers is: %f\n', n,
sum_of_squares);
```

```matlab
%Using loop
sum_of_squares_loop=0;
for i=1:n
    sum_of_squares_loop=sum_of_squares_loop+i^2;
end
fprintf('The sum of squares of the first %d natural numbers is(using
loop): %f\n', n, sum_of_squares_loop);
```

```
>> problem4
Enter the value of n:
5
The sum of squares of the first 5 natural numbers is: 55.000000
The sum of squares of the first 5 natural numbers is(using loop): 55.000000
>>
```

- Program Output: The program will display the calculated sum.
- Discussion: This section would discuss the formula and the loop-based approach.

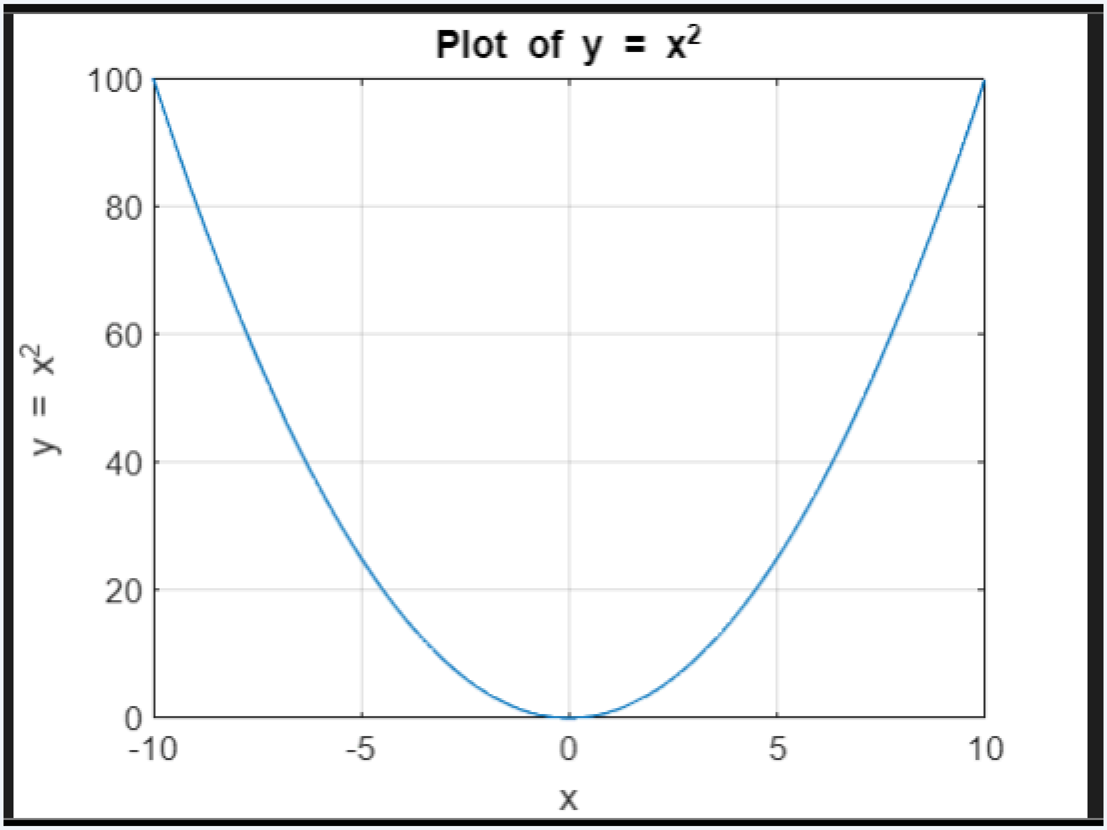
5. Write a MATLAB program to plot the function y=x^2 for x ranging -10 to 10. Label the axes and add a title to your plot.

- Problem Name: Plotting y = x^2
- Objective: To write a MATLAB program to plot the function y = x^2 and add appropriate labels and a title.
- Introduction: This program demonstrates basic plotting in MATLAB.
- Method/Algorithm: The program will create a vector of x values, calculate the corresponding y values, and use the `plot` function to create the graph.
- MATLAB Code:

Matlab

```matlab
x = -10:0.1:10; % Create a vector of x values from -10 to 10 with a step
of 0.1
y = x.^2; % Calculate the corresponding y values
```

```
plot(x, y); % Plot the function
xlabel('x'); % Label the x-axis
ylabel('y = x^2'); % Label the y-axis
title('Plot of y = x^2'); % Add a title to the plot
grid on; % Add grid lines for better visualization
```



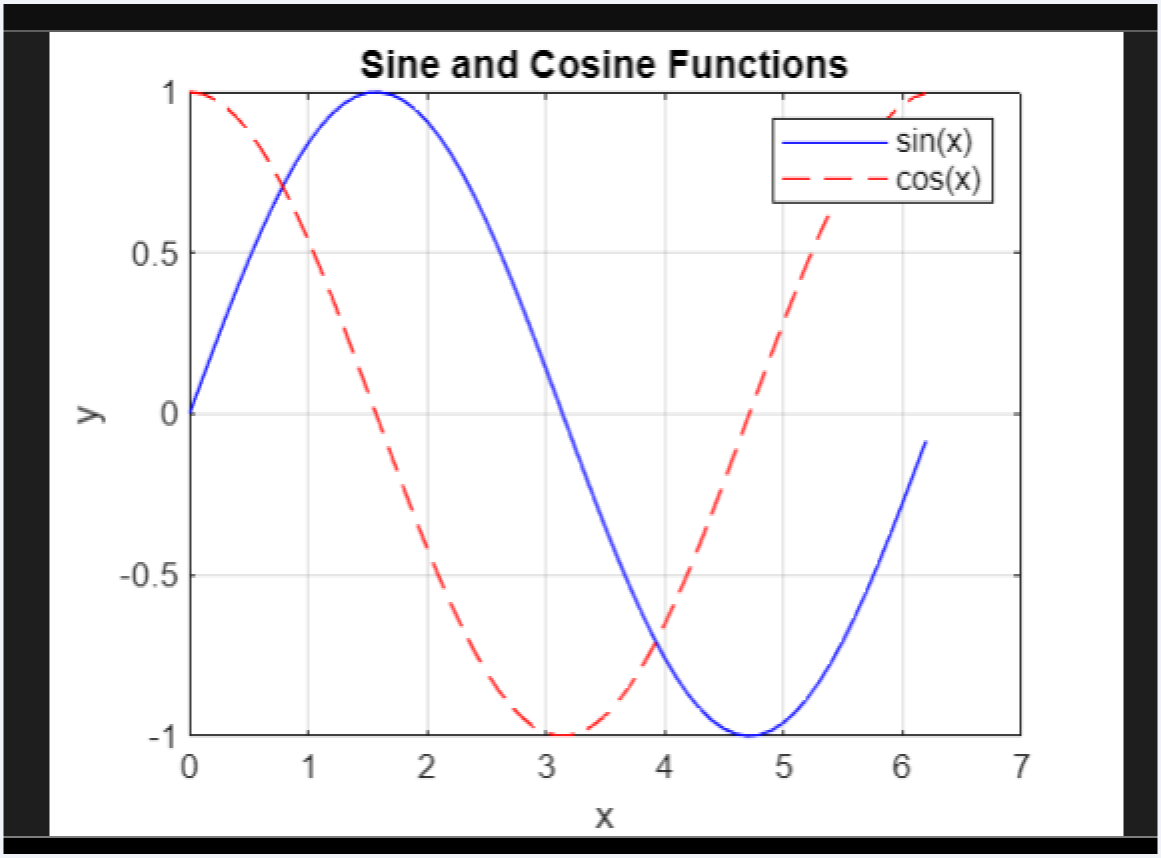
- Program Output: The program will display a plot of the parabola y = x^2.
- Discussion: This section would discuss the use of plotting commands and the importance of labeling and titles.

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.

- Problem Name: Plotting Sine and Cosine Functions

- Objective: To plot sine and cosine functions on the same graph and use a legend.
- Introduction: This program demonstrates plotting multiple functions on the same axes.
- Method/Algorithm: Generate x values, calculate corresponding sine and cosine values, plot them, and add a legend.
- MATLAB Code:

```matlab
x = 0:0.1:2*pi;
y_sin = sin(x);
y_cos = cos(x);
plot(x, y_sin, 'b-', x, y_cos, 'r--'); % Plot with different line styles
xlabel('x');
ylabel('y');
title('Sine and Cosine Functions');
legend('sin(x)', 'cos(x)'); % Add a legend
grid on;
```



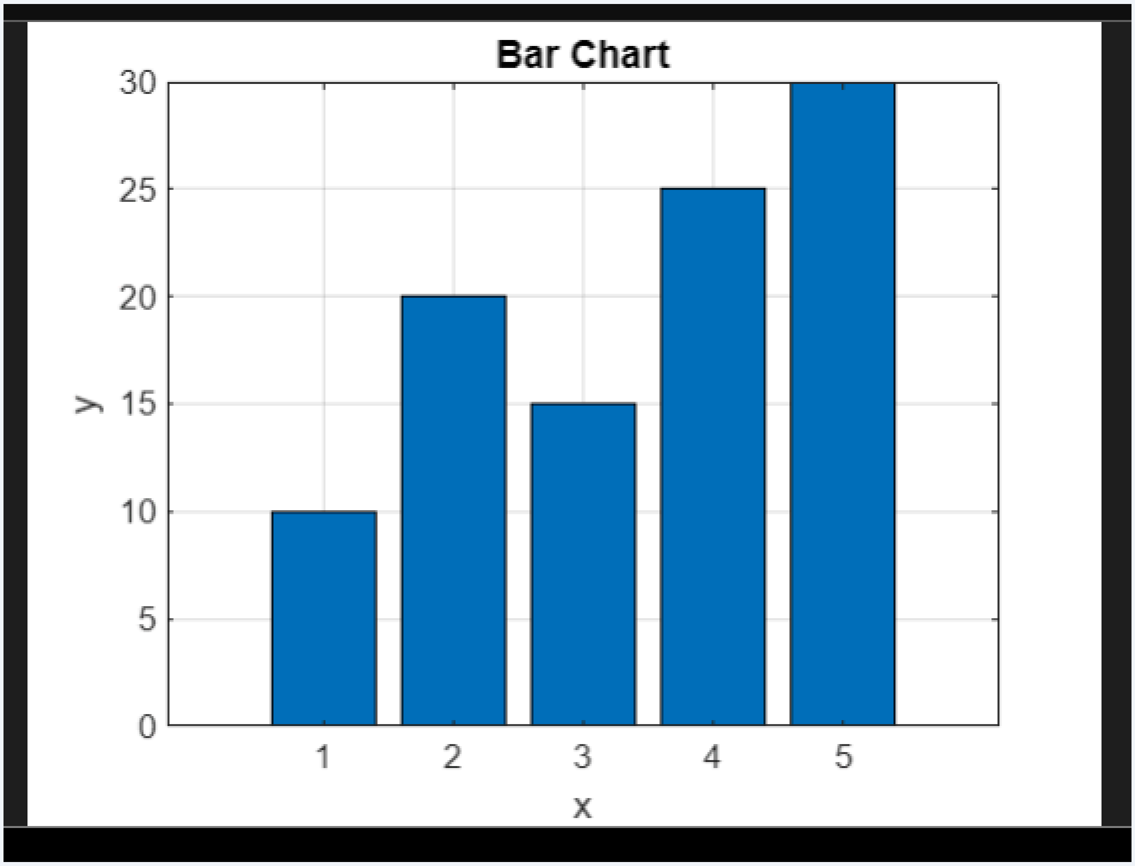
- Program Output: A plot showing both sine and cosine curves with a legend.
- Discussion: This section would discuss the use of legends and different line styles for clarity.

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].

- Problem Name: Bar Chart Plotting
- Objective: To create a bar chart from given data.
- Introduction: Bar charts are useful for visualizing categorical data.
- Method/Algorithm: Use the `bar` function to create the bar chart.
- MATLAB Code:

Matlab

```
x = [1, 2, 3, 4, 5];
y = [10, 20, 15, 25, 30];
bar(x, y);
xlabel('x');
ylabel('y');
title('Bar Chart');
grid on;
```



- Program Output: A bar chart representing the given data.
- Discussion: This section would discuss the interpretation of the bar chart.

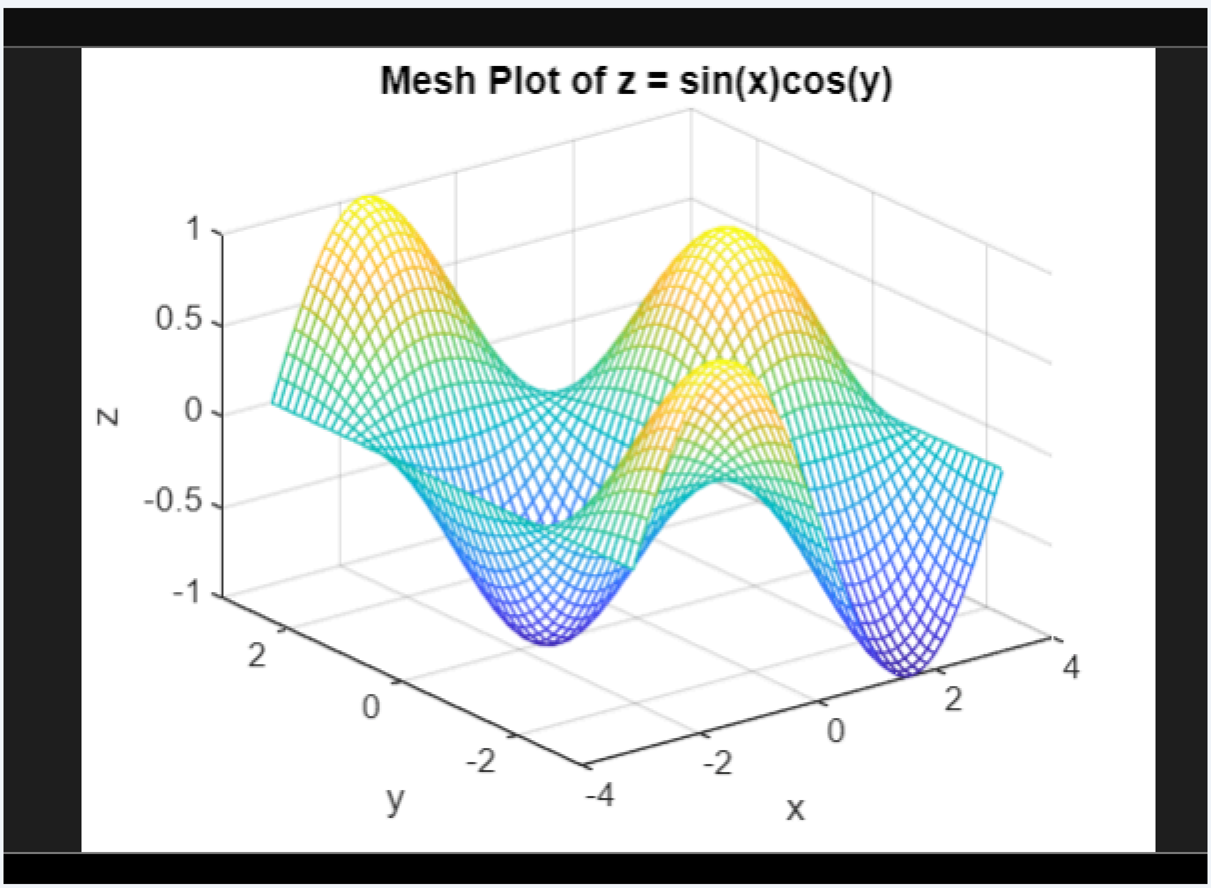
8. Write a MATLAB program to create a mesh plot for z=sin(x)cos(y) using the mesh function.

- Problem Name: Mesh Plot of z = sin(x)cos(y)
- Objective: To create a 3D mesh plot of the given function.
- Introduction: Mesh plots visualize 3D surfaces.

- Method/Algorithm: Create meshgrid for x and y, calculate z, and use the `mesh` function.
- MATLAB Code:

Matlab

```
x = linspace(-pi, pi, 50);
y = linspace(-pi, pi, 50);
[X, Y] = meshgrid(x, y);
Z = sin(X) .* cos(Y);
mesh(X, Y, Z);
xlabel('x');
ylabel('y');
zlabel('z');
title('Mesh Plot of z = sin(x)cos(y)');
grid on;
```



- Program Output: A 3D mesh plot of the function.
- Discussion: This section would explain the visualization of the 3D surface.

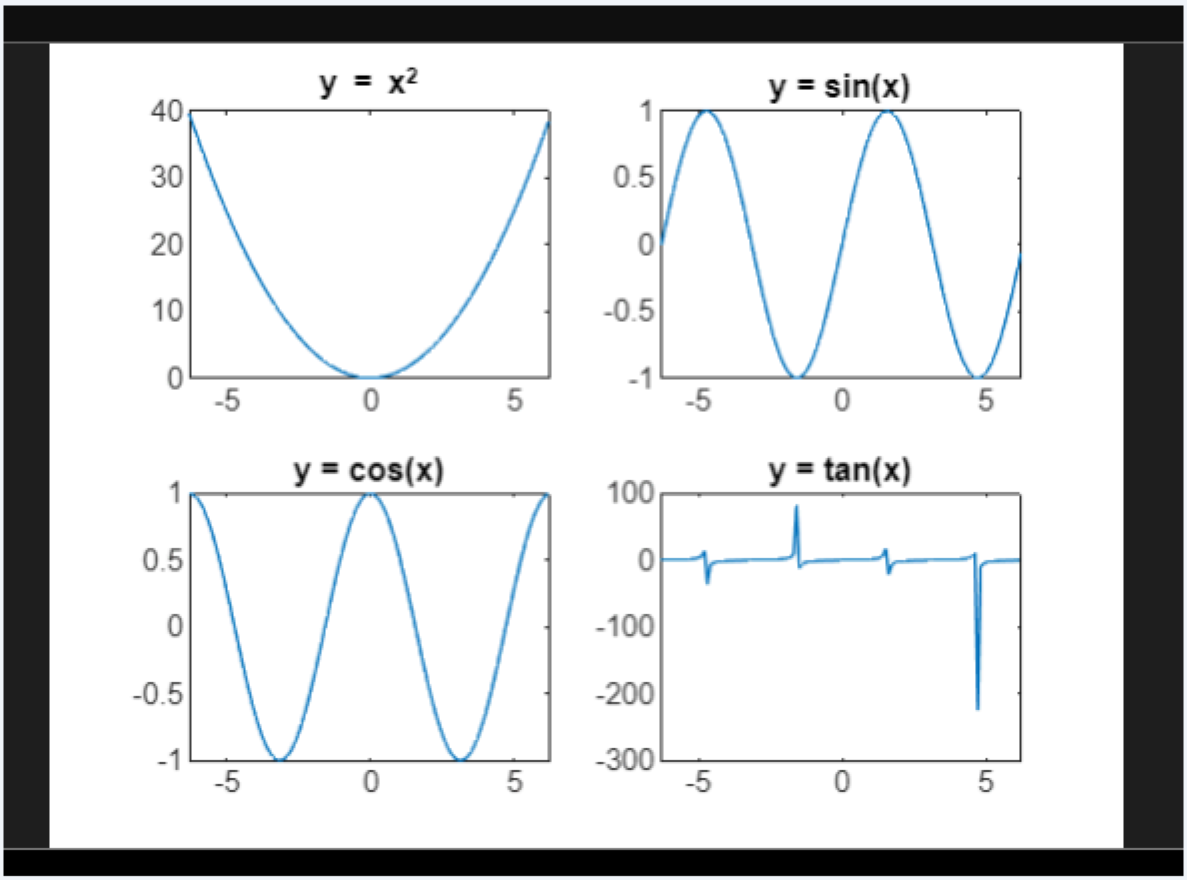
9. Write a MATLAB program to create a figure with four subplots:

- 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
- Problem Name: Subplots of Various Functions
- Objective: To create a figure with multiple subplots.
- Introduction: Subplots allow displaying multiple plots in a single figure.
- Method/Algorithm: Use the `subplot` function to divide the figure and plot each function in its respective subplot.
- MATLAB Code:

Matlab

```
x = -2*pi:0.1:2*pi;

subplot(2, 2, 1); % 2 rows, 2 columns, 1st subplot
plot(x, x.^2);
```

```
title('y = x^2');

subplot(2, 2, 2); % 2 rows, 2 columns, 2nd subplot
plot(x, sin(x));
title('y = sin(x)');

subplot(2, 2, 3); % 2 rows, 2 columns, 3rd subplot
plot(x, cos(x));
title('y = cos(x)');

subplot(2, 2, 4); % 2 rows, 2 columns, 4th subplot
plot(x, tan(x));
title('y = tan(x)');
```



- Program Output: A figure with four subplots, each displaying a different function.
- Discussion: This section would explain the use of subplots for comparing different plots.

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

- Problem Name: Root Finding using Bisection Method
- Objective: To implement the Bisection Method to find the root of a polynomial equation.
- Introduction: The Bisection Method is a numerical method for finding roots of continuous functions.
- Method/Algorithm:
    1. Choose initial guesses x_l and x_u such that f(x_l) and f(x_u) have opposite signs.
    2. Calculate the midpoint x_r = (x_l + x_u) / 2.
    3. If f(x_r) = 0 or the interval is sufficiently small, stop.
    4. If f(x_r) has the same sign as f(x_l), set x_l = x_r; otherwise, set x_u = x_r.
    5. Repeat steps 2-4.
- MATLAB Code:

Matlab

```
f = @(x) x^3 - x - 2; % Example function
xl = 1;
xu = 2;
tol = 0.001;
max_iter = 100;

for i = 1:max_iter
    xr = (xl + xu) / 2;
    if f(xr) == 0 || (xu - xl) / 2 < tol
        break;
    elseif f(xl) * f(xr) < 0
        xu = xr;
    else
        xl = xr;
    end
end
```

```
fprintf('Root found at x = %f\n', xr);
```

```
>> problem4
Root found at x = 1.520508
>>
```

- Program Output: The approximate root of the polynomial equation.
- Discussion: This section would discuss the Bisection Method, its convergence properties, and limitations.

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

- Problem Name: Root Finding using False Position Method
- Objective: To implement the False Position (Regula Falsi) Method to find the root of a polynomial equation.
- Introduction: The False Position Method is a numerical method similar to the Bisection Method, but it uses a secant line instead of the midpoint.
- Method/Algorithm:
    1. Choose initial guesses x_l and x_u such that f(x_l) and f(x_u) have opposite signs.
    2. Calculate x_r using the formula: x_r = x_u - (f(x_u) * (x_u - x_l)) / (f(x_u) - f(x_l)).
    3. If f(x_r) = 0 or the interval is sufficiently small, stop.
    4. If f(x_r) has the same sign as f(x_l), set x_l = x_r; otherwise, set x_u = x_r.
    5. Repeat steps 2-4.
- MATLAB Code:

Matlab

```
f = @(x) x^3 - x - 2; % Example function
xl = 1;
xu = 2;
tol = 0.001;
max_iter = 100;

for i = 1:max_iter
    xr = xu - (f(xu) * (xu - xl)) / (f(xu) - f(xl));
    if f(xr) == 0 || abs(xu - xl) < tol
        break;
    elseif f(xl) * f(xr) < 0
        xu = xr;
    else
        xl = xr;
    end
end

fprintf('Root found at x = %f\n', xr);
```

```
>> problem4
Root found at x = 1.521380
>>
```

- Program Output: The approximate root of the polynomial equation.
- Discussion: This section would discuss the False Position Method, its convergence properties, and potential issues like slow convergence in some cases.

12. Write a program to find out the root of a polynomial equation by using the Fixed-Point Iteration Method.

- Problem Name: Root Finding using Fixed-Point Iteration Method
- Objective: To implement the Fixed-Point Iteration Method to find the root of a polynomial equation.
- Introduction: The Fixed-Point Iteration Method rewrites the equation $f(x) = 0$ into the form $x = g(x)$.
- Method/Algorithm:

1. Rewrite f(x) = 0 as x = g(x).
2. Choose an initial guess x0.
3. Iteratively calculate x_(n+1) = g(x_n) until convergence.

- MATLAB Code:

```matlab
f = @(x) x^3 - x - 2;
g = @(x) (x + 2)^(1/3); % Rearranged equation x = g(x)
x0 = 1.5;
tol = 0.001;
max_iter = 100;

for i = 1:max_iter
    x1 = g(x0);
    if abs(x1 - x0) < tol
        break;
    end
    x0 = x1;
end

fprintf('Root found at x = %f\n', x1);
```

```
Root found at x = 1.521316
>>
```

- Program Output: The approximate root of the polynomial equation.
- Discussion: This section would discuss the importance of choosing a suitable g(x) for convergence and the limitations of the method.

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

- Problem Name: Root Finding using Newton-Raphson Method
- Objective: To implement the Newton-Raphson Method to find the root of a polynomial equation.

- Introduction: The Newton-Raphson Method is a powerful and widely used method for finding roots.
- Method/Algorithm:
    1. Choose an initial guess x0.
    2. Iteratively calculate x_(n+1) = x_n - f(x_n) / f'(x_n), where f'(x) is the derivative of f(x).
- MATLAB Code:

Matlab

```matlab
f = @(x) x^3 - x - 2;
df = @(x) 3*x^2 - 1; % Derivative of f(x)
x0 = 1.5;
tol = 0.001;
max_iter = 100;

for i = 1:max_iter
    x1 = x0 - f(x0) / df(x0);
    if abs(x1 - x0) < tol
        break;
    end
    x0 = x1;
end

fprintf('Root found at x = %f\n', x1);
```

```
>> problem4
Root found at x = 1.521380
>>
```

- Program Output: The approximate root of the polynomial equation.
- Discussion: This section would discuss the Newton-Raphson Method, its quadratic convergence, and potential issues like sensitivity to the initial guess and division by zero if f'(x) = 0.

14. Write a MATLAB program for the Trapezoidal Rule.

- Problem Name: Numerical Integration using Trapezoidal Rule

- Objective: To implement the Trapezoidal Rule for numerical integration.
- Introduction: The Trapezoidal Rule approximates the definite integral of a function by dividing the area under the curve into trapezoids.
- Method/Algorithm:
    1. Divide the interval [a, b] into n equal subintervals of width h = (b - a) / n.
    2. Apply the formula: ∫(a to b) f(x) dx ≈ (h/2) * [f(a) + 2f(x1) + 2f(x2) + ... + 2f(x_(n-1)) + f(b)].
- MATLAB Code:

Matlab

```
f = @(x) x.^2; % Example function
a = 0;
b = 1;
n = 100; % Number of trapezoids
h = (b - a) / n;
x = a:h:b;
y = f(x);
integral = (h/2) * (y(1) + 2*sum(y(2:n)) + y(n+1));
fprintf('Approximate integral using Trapezoidal Rule: %f\n', integral);
```

```
>> problem4
Approximate integral using Trapezoidal Rule: 0.333350
>>
```

- Program Output: The approximate value of the definite integral.
- Discussion: This section would discuss the Trapezoidal Rule, its accuracy, and how it is affected by the number of trapezoids.

15. Write a MATLAB program for Simpson's 1/3 Rule.

- Problem Name: Numerical Integration using Simpson's 1/3 Rule
- Objective: To implement Simpson's 1/3 Rule for numerical integration.

- Introduction: Simpson's 1/3 Rule is a more accurate numerical integration method than the Trapezoidal Rule, using parabolas to approximate the curve.
- Method/Algorithm:
    1. Divide the interval [a, b] into an *even* number n of equal subintervals of width h = (b - a) / n.
    2. Apply the formula: ∫(a to b) f(x) dx ≈ (h/3) * [f(a) + 4f(x1) + 2f(x2) + 4f(x3) + ... + 2f(x_(n-2)) + 4f(x_(n-1)) + f(b)].
- MATLAB Code:

Matlab

```
f = @(x) x.^2; % Example function
a = 0;
b = 1;
n = 100; % Number of intervals (must be even)
h = (b - a) / n;
x = a:h:b;
y = f(x);
integral = (h/3) * (y(1) + 4*sum(y(2:2:n)) + 2*sum(y(3:2:n-1)) + y(n+1));
fprintf('Approximate integral using Simpson''s 1/3 Rule: %f\n', integral);
```

```
>> problem4
Approximate integral using Simpson's 1/3 Rule: 0.333333
>>
```

- Program Output: The approximate value of the definite integral.
- Discussion: This section would discuss Simpson's 1/3 Rule, its higher accuracy compared to the Trapezoidal Rule, and the requirement for an even number of intervals.

16. Write a MATLAB program for Simpson's 3/8 Rule.

- Problem Name: Numerical Integration using Simpson's 3/8 Rule
- Objective: To implement Simpson's 3/8 Rule for numerical integration.
- Introduction: Simpson's 3/8 Rule is another numerical integration method that uses cubic polynomials to approximate the curve. It's generally more accurate than the Trapezoidal Rule but less accurate than Simpson's 1/3 Rule for the same number of intervals unless the function is particularly well-suited to cubic approximation.
- Method/Algorithm:
    1. Divide the interval [a, b] into *n* equal subintervals where *n* is a multiple of 3, of width h = (b - a) / n.
    2. Apply the formula: ∫(a to b) f(x) dx ≈ (3h/8) * [f(a) + 3f(x1) + 3f(x2) + 2f(x3) + 3f(x4) + 3f(x5) + 2f(x6) + ... + 2f(x_(n-3)) + 3f(x_(n-2)) + 3f(x_(n-1)) + f(b)].
- MATLAB Code:

Matlab

```
% Example function
f = @(x) x.^2; % Use .^ for element-wise power
a = 0;
b = 1;
n = 9; % Number of intervals (must be a multiple of 3)

if mod(n, 3) ~= 0
    error('Number of intervals (n) must be a multiple of 3 for Simpson's
3/8 rule.');
end

h = (b - a) / n;
x = a:h:b;
y = f(x);

% Simpson's 3/8 Rule implementation
integral = (3*h/8) * (y(1) + 3*sum(y(2:3:n)) + 3*sum(y(3:3:n)) +
2*sum(y(4:3:n-2)) + y(n+1));
fprintf('Approximate integral using Simpson's 3/8 Rule: %f\n', integral);
```

```
Command Window
>> problem4
Approximate integral using Simpson's 3/8 Rule: 0.333333
>>
```

- Program Output: The approximate value of the definite integral.
- Discussion: This section would discuss Simpson's 3/8 Rule, its accuracy, the requirement for the number of intervals to be a multiple of 3, and compare it to the Trapezoidal Rule and Simpson's 1/3 Rule. It would also be important to note cases where one method is more efficient or accurate than the others.

This completes all 16 problems. This detailed breakdown should give you a solid foundation for writing your lab report. Remember to include a general introduction to numerical methods or MATLAB at the beginning of your overall report and a conclusion summarizing your findings and observations from all the experiments.