

LAB MANUAL

MATLAB APPLICATIONS IN ENGINEERING LAB

[23AES2]

Prepared by
Dr. Sreenadh Chevula, Ph. D
Associate Professor
Mr. G.V Surya Narayana
Sr. Assistant Professor
Department of Aerospace Engineering



Lakireddy Bali Reddy College of Engineering (A)
Mylavaram, NTR (Dt.), Andhra Pradesh, India - 521 230

Affiliated To
Jawaharlal Nehru Technological University, Kakinada, Andhra
Pradesh, India

LBRCE VISION

To empower students to become technologically vibrant, innovative, and emotionally mature; and to train them to face the challenges of the quality-conscious globalized world economy.

LBRCE MISSION

1. To provide an environment most conducive to learning and to create a stimulating intellectual atmosphere on the campus.
2. To achieve Academic Excellence.
3. To ensure a holistic development of personality.
4. To spread education to rural areas.
5. To establish partnership between Institution & Industry.

LBRCE QUALITY POLICY

We at Lakireddy Bali Reddy College of Engineering are committed to impart quality Technical Education and Training, to prepare dynamic and socially responsible technocrats for effective nation-building by recruiting dedicated and competent faculty, providing state-of-the-art Infrastructure and equipment that income-passes the stated University Curriculum and beyond.

1. To achieve Academic Excellence of a high order.
2. To nurture nation-building skills with emphasis on developing a well built in personality.
3. To ensure that education spreads to rural areas.
4. To establish partnership between Institute and Industry.
5. To explore the Innate Caliber of the students by providing a unique learning experience.
6. To inculcate acquisition of superior intellectual and interpersonal skills.
7. To improve analytical thinking and problem-solving skills.
8. To understand teamwork dynamics and decision-making process.
9. To enable to cope with intense pressure over an extended period.
10. To improve the communication skills so that the students communicate cogently, rationally, and convincingly through acceptable social behaviour.
11. To generate competence to build, motivate and lead project teams effectively.
12. To make learning highly interactive and object-oriented by adopting innovative methods.

DEPARTMENT OF AEROSPACE ENGINEERING

VISION

Department of Aerospace Engineering aims to be indispensable source in Aerospace Engineering which has a zeal to provide the value driven platform for the students to acquire knowledge and empower themselves to shoulder higher responsibility in building a strong nation.

MISSION

The primary mission of the department is to promote engineering education and research. To strive consistently to provide quality education, keeping in pace with time and technology. Department passions to integrate the intellectual, spiritual, ethical, and social development of the students for shaping them into dynamic engineers.

QUALITY POLICY STATEMENT

Impart up-to-date knowledge to the students in Aerospace area to make them quality engineers. Make the students experience the applications on quality equipment and tools. Provide systems, resources, and training opportunities to achieve continuous improvement. Maintain global standards in education, training, and services.

23AES2- MATLAB APPLICATIONS IN ENGINEERING LAB

L	T	P	Cr.
0	1	2	2

Pre-requisites: Engineering Mechanics and Numerical Methods

Course Educational Objectives: This course is designed to use the basic in-built commands and to write the MATLAB code to solve ordinary differential equation, integration and make the user- friendly environment using graphical user interface.

Course Outcomes: At the end of the course, the student will be able to

CO1: Apply the basic MATLAB operations in basic engineering problems (Apply – L3)

CO2: Solve the system of linear algebraic equation using matrix operation. (Understand – L2)

CO3: Apply the graphical user interface to write the code as more user friendly (Apply – L3)

LIST OF EXPERIMENTS**Part – I: Introduction to MATLAB**

1. (a) Find the sum of first 100 natural numbers (b) Perform Basic matrix operations?
2. (a) Write a MATLAB program that adds all elements of an array named S with even indices.
(b) Perform double integration problems?
3. Find the roots of a linear equation by using Newton's and secant Method
4. (a) Introduction to basic plots 2D, 3D (b) MATLAB Graphical user interface addition and subtraction

Part – II: Application of MATLAB

5. Write a MATLAB Code to determine Lift Curve Slope from the given parameters?
6. Solving of ordinary differential equation using Runge-Kutta method a numerical approach
7. (a) Write a MATLAB Code to determine Compute the Laplace transform of $1/\sqrt{x}$ (b) Write a MATLAB Code to determine Eigenvalues and eigenvectors of a linear algebraic equations
8. (a) Solve System of Linear Equations (b) Solve system of nonlinear equations
9. Graphics kinematics of particle position, velocity, and acceleration
10. Develop the graphical user interface to identify the area moment of inertia of simple section trapezoidal and triangle.
11. Identification of shear force and bending moment diagram of cantilever beam with point load
12. MATLAB Graphical user interface: Design and develop a scientific calculator.

E.No	NAME OF THE EXPERIMENT	Pg No
	Introduction to MATLAB environment:	
Part – I: Introduction to MATLAB		
1	(a) Find the sum of first 100 natural numbers (b) Perform Basic matrix operations?	
2	(a) Write a MATLAB program that adds all elements of an array named S with even indices. (b) Perform double integration problems?	
3	Find the roots of a linear equation by using Newton's and secant Method	
4	(a) Introduction to basic plots 2D, 3D (b) MATLAB Graphical user interface addition and subtraction	
Part – II: Application of MATLAB		
5	Write a MATLAB Code to determine Lift Curve Slope from the given parameters?	
6	Solving of ordinary differential equation using Runge-Kutta method a numerical approach	
7	(a) Write a MATLAB Code to determine Compute the Laplace transform of $1/\sqrt{x}$ (b) Write a MATLAB Code to determine Eigenvalues and eigenvectors of a linear algebraic equation	
8	(a) Solve System of Linear Equations (b) Solve system of nonlinear equations	
9	Graphics kinematics of particle position, velocity, and acceleration	
10	Develop the graphical user interface to identify the area moment of inertia of simple section trapezoidal and triangle.	
11	Identification of shear force and bending moment diagram of cantilever beam with point load	
12	MATLAB Graphical user interface: Design and develop a scientific calculator.	

INDEX

Part – I: Introduction to MATLAB

1. (a) Find the sum of first 100 natural numbers (b) Perform Basic matrix operations?
2. (a) Write a MATLAB program that adds all elements of an array named S with even indices.
(b) Perform double integration problems?
3. Find the roots of a linear equation by using Newton's and secant Method
4. (a) Introduction to basic plots 2D, 3D (b) MATLAB Graphical user interface addition and subtraction

Part – II: Application of MATLAB

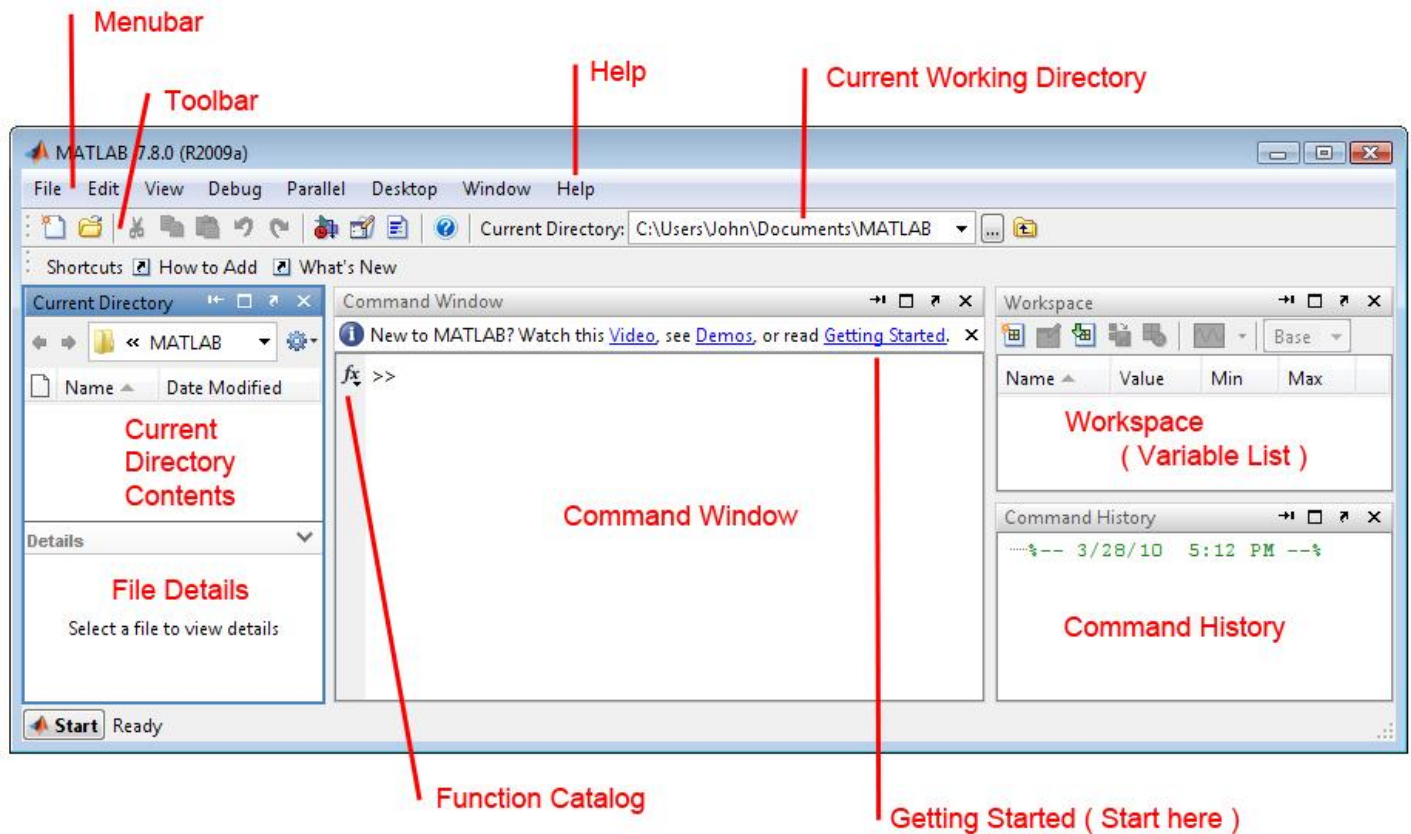
5. Write a MATLAB Code to determine Lift Curve Slope from the given parameters?
6. Solving of ordinary differential equation using Runge-Kutta method a numerical approach
7. (a) Write a MATLAB Code to determine Compute the Laplace transform of $1/\sqrt{x}$ (b) Write a MATLAB Code to determine Eigenvalues and eigenvectors of a linear algebraic equation
8. (a) Solve System of Linear Equations (b) Solve system of nonlinear equations
9. Graphics kinematics of particle position, velocity, and acceleration
10. Develop the graphical user interface to identify the area moment of inertia of simple section trapezoidal and triangle.
11. Identification of shear force and bending moment diagram of cantilever beam with point load
12. MATLAB Graphical user interface: Design and develop a scientific calculator.

Part – I

Introduction to MATLAB

INTRODUCTION TO MATLAB ENVIRONMENT:

The MATLAB Work Environment



Command Window

Command History

Workspace

Current Directory

Figure Window

Command Window: Whenever MATLAB is invoked, the main window called command window is activated. The command window displays the command prompt '>>>' and a cursor where commands are entered and are executed instantaneously.

Command History Window: Command history window consists of a list of all the commands that are entered at the command window. These commands remain in the list until they are deleted. Any command may be executed by selecting and double clicking it with the mouse.

Workspace: A workspace is a collection of all the variables that have been generated so far in the current MATLAB session and shows their data type and size. All the commands executed from Command Window and all the script files executed from the Command Window share common workspace, so they can share all the variables.

Current Directory: The Current Directory window contains all the files and folders present in the Current Directory. To run any file, it must either be in the Current Directory or on the search path.

Edit Window: An Edit Window is used to create a new program file, or to modify existing files. In this window, programs can be written, edited and saved. The programs written using the MATLAB editor are automatically assigned an extension (.m) by the editor and are known as M- files.

Figure Window: A Figure Window is a separate window with default white background and is used to display MATLAB graphics. The results of all the graphic commands executed are displayed in the figure window.

MATLAB allows two different types of arithmetic operations –

- Matrix arithmetic operations
- Array arithmetic operations

Matrix arithmetic operations are same as defined in linear algebra. Array operations are executed element by element, both on one dimensional and multi-dimensional array.

The matrix operators and arrays operators are differentiated by the period (.) symbol. However, as the addition and subtraction operation is same for matrices and arrays, the operator is same for both cases.

The following table gives brief description of the operators –

Operator	Description
+	Addition or unary plus. A+B adds the values stored in variables A and B. A and B must have the same size, unless one is a scalar. A scalar can be added to a matrix of any size.
-	Subtraction or unary minus. A-B subtracts the value of B from A. A and B must have the same size, unless one is a scalar. A scalar can be subtracted from a matrix of any size.
*	Matrix multiplication. $C = A*B$ is the linear algebraic product of the matrices A and B.
	$C(i, j) = \sum_{k=1}^n A(i, k)B(k, j)$ More precisely, For non-scalar A and B, the number of columns of A must be equal to the number of rows of B. A scalar can multiply a matrix of any size.
.*	Array multiplication. A.*B is the element-by-element product of the arrays A and B. A and B must have the same size, unless one
one	

/	of them is a scalar. Slash or matrix right division. B/A is roughly the same as $B \cdot \text{inv}(A)$. More precisely, $B/A = (A' \backslash B)'$.
./	Array right division. $A./B$ is the matrix with elements $A(i,j)/B(i,j)$. A and B must have the same size, unless one of them is a scalar.
\	Backslash or matrix left division. If A is a square matrix, $A \backslash B$ is roughly the same as $\text{inv}(A) \cdot B$, except it is computed in a different way. If A is an n-by-n matrix and B is a column vector with n components, or a matrix with several such columns, then $X = A \backslash B$ is the solution to the equation $AX = B$. A warning message is displayed if A is badly scaled or nearly singular.
.\	Array left division. $A \backslash B$ is the matrix with elements $B(i,j)/A(i,j)$. A and B must have the same size, unless one of them is a scalar.
^	Matrix power. X^p is X to the power p, if p is a scalar. If p is an integer, the power is computed by repeated squaring. If the integer is negative, X is inverted first. For other values of p, the calculation involves eigenvalues and eigenvectors, such that if $[V,D] = \text{eig}(X)$, then $X^p = V \cdot D.^p / V$.
.^	Array power. $A.^B$ is the matrix with elements $A(i,j)$ to the $B(i,j)$ power. A and B must have the same size, unless one of them is a scalar.
'	Matrix transpose. A' is the linear algebraic transpose of A. For complex matrices, this is the complex conjugate transpose.
.'	Array transpose. A' is the array transpose of A. For complex matrices, this does not involve conjugation.

Functions for Arithmetic Operations: -

Apart from the above-mentioned arithmetic operators, MATLAB provides the following commands/functions used for similar purpose – **Function**
uplus(a)

plus (a,b)

uminus(a)

minus(a, b)

Description

Unary plus; increments by the amount a

Plus; returns $a + b$

Unary minus; decrements by the amount a

Minus; returns $a - b$

times(a, b)	Array multiply; returns $a.*b$
mtimes(a, b)	Matrix multiplication; returns $a*b$
rdivide(a, b)	Right array division; returns $a ./ b$
ldivide(a, b)	Left array division; returns $a ./ b$
mrdivide(A, B)	Solve systems of linear equations $xA = B$ for x
mldivide(A, B)	Solve systems of linear equations $Ax = B$ for x
power(a, b)	Array power; returns $a.^b$
mpower(a, b)	Matrix power; returns a^b

PROGRAM:-

```

a = 10;
b = 20;
c = a + b
d = a - b
e = a * b
f = a / b
g = a \ b
x = 7;
y = 3;   z = x ^ y

```

MATLAB allows writing two kinds of program files –

The M Files

- **Scripts** – script files are program files with **.m extension**. In these files, you write series of commands, which you want to execute together. Scripts do not accept inputs and do not return any outputs. They operate on data in the workspace.
- **Functions** – functions files are also program files with **.m extension**. Functions can accept inputs and return outputs. Internal variables are local to the function.

You can use the MATLAB editor or any other text editor to create your **.m** files.

RESULTS: - All the operations have been performed using MATLAB and results have been seen on the command window.

PRECAUTIONS: -

- 1) Program must be written carefully to avoid errors.
- 2) Programs can never be saved as standard function name.
- 3) Functions in MATLAB are case sensitive so commands must be written in proper format.

1. (a) Find the sum of first 100 natural numbers

(b) Perform Basic matrix operations?

OBJECTIVES:

- (i) To define and use variables and functions in MATLAB.
- (ii) To define and use vectors, matrices, MATLAB.
- (iv) To create and use m-files.

SOFTWARE USED: MATLAB 2022 A

MATLAB (an abbreviation of "Matrix Laboratory") is a proprietary multi-paradigm programming language and numeric computing environment developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages. Although MATLAB is intended primarily for numeric computing, an optional toolbox uses the MuPAD symbolic engine allowing access to symbolic computing abilities. An additional package, Simulink, adds graphical multi-domain simulation and model-based design for dynamic and embedded systems. As of 2020, MATLAB has more than 4 million users worldwide. They come from various backgrounds of engineering, science, and economics. As of 2017, more than 5000 global colleges and universities use MATLAB to support instruction and research.

It allows matrix manipulations; plotting of functions and data; implementation of algorithms; creation of user interfaces; interfacing with programs written in other languages, including C, C++, Java, and FORTRAN; analyse data; develop algorithms; and create models and applications.

In MATLAB environment, every variable is an array or matrix. Variables can be defined in MATLAB in the following ways:

(a) Find the sum of first 100 natural numbers

To find the sum of the first 100 natural numbers in MATLAB, you can use the following code:

```
n = 100; % Number of terms
sum_natural_numbers = sum(1: n); % Sum of first n natural numbers
disp(sum_natural_numbers= );
```

Answer : sum_natural_numbers =
5050

(or)

Alternatively, since the sum of the first n natural numbers follows the formula:

$$S = \frac{n(n+1)}{2}$$

Code:

```
n = 100; % Number of terms
sum_natural_numbers = n * (n + 1) / 2; % Using the formula
disp(sum_natural_numbers);
```

(b) Perform Basic matrix operations

1. Matrix Addition:

```
A = [1, 2, 3; 4, 5, 6; 7, 8, 9]; % Define matrix A
B = [9, 8, 7; 6, 5, 4; 3, 2, 1]; % Define matrix B
C = A + B; % Matrix addition
disp('Matrix A + B:');
disp(C);
```

2. Matrix Subtraction:

```
C = A - B; % Matrix subtraction
disp('Matrix A - B:');
disp(C);
```

3. Matrix Multiplication:

```
D = A * B; % Matrix multiplication
disp('Matrix A * B:');
disp(D);
```

4. Element-wise Multiplication:

```
E = A .* B; % Element-wise multiplication
disp('Element-wise multiplication (A .* B):');
disp(E);
```

Matrix Transpose:

```
F = A.'; % Matrix transpose
disp('Transpose of Matrix A:');
disp(F);
```

6. Matrix Determinant:

```
det_A = det(A); % Determinant of matrix A
disp('Determinant of Matrix A:');
disp(det_A);
```

7. Matrix Inverse (if the matrix is invertible):

```

if det_A ~= 0
    inv_A = inv(A);           % Matrix inverse
    disp('Inverse of Matrix A:');
    disp(inv_A);
else
    disp('Matrix A is singular, so it does not have an inverse.');
```

```

end
```

8. Matrix Rank:

```

rank_A = rank(A);           % Rank of matrix A
disp('Rank of Matrix A:');
disp(rank_A);
```

9. Matrix Size:

```

size_A = size(A);           % Size of matrix A
disp('Size of Matrix A:');
disp(size_A);
```

10. Matrix Identity:

```

I = eye(3);                 % 3x3 identity matrix
disp('3x3 Identity Matrix:');
disp(I);
```

RELATED QUESTIONS: -

1. How to create matrix in MATLAB?
2. What is the function of MATLAB function clc?
3. What happens if we put a semicolon after a statement in a MATLAB program?
4. What are M –files?

2. (a) Write a MATLAB program that adds all elements of an array named S with even indices. (b) Perform double integration problems?

OBJECTIVE: -

- (a) MATLAB program that adds all elements of an array named S with even indices.
- (b) Perform double integration problems

SOFTWARE USED: - MATLAB 2022a or 2019a

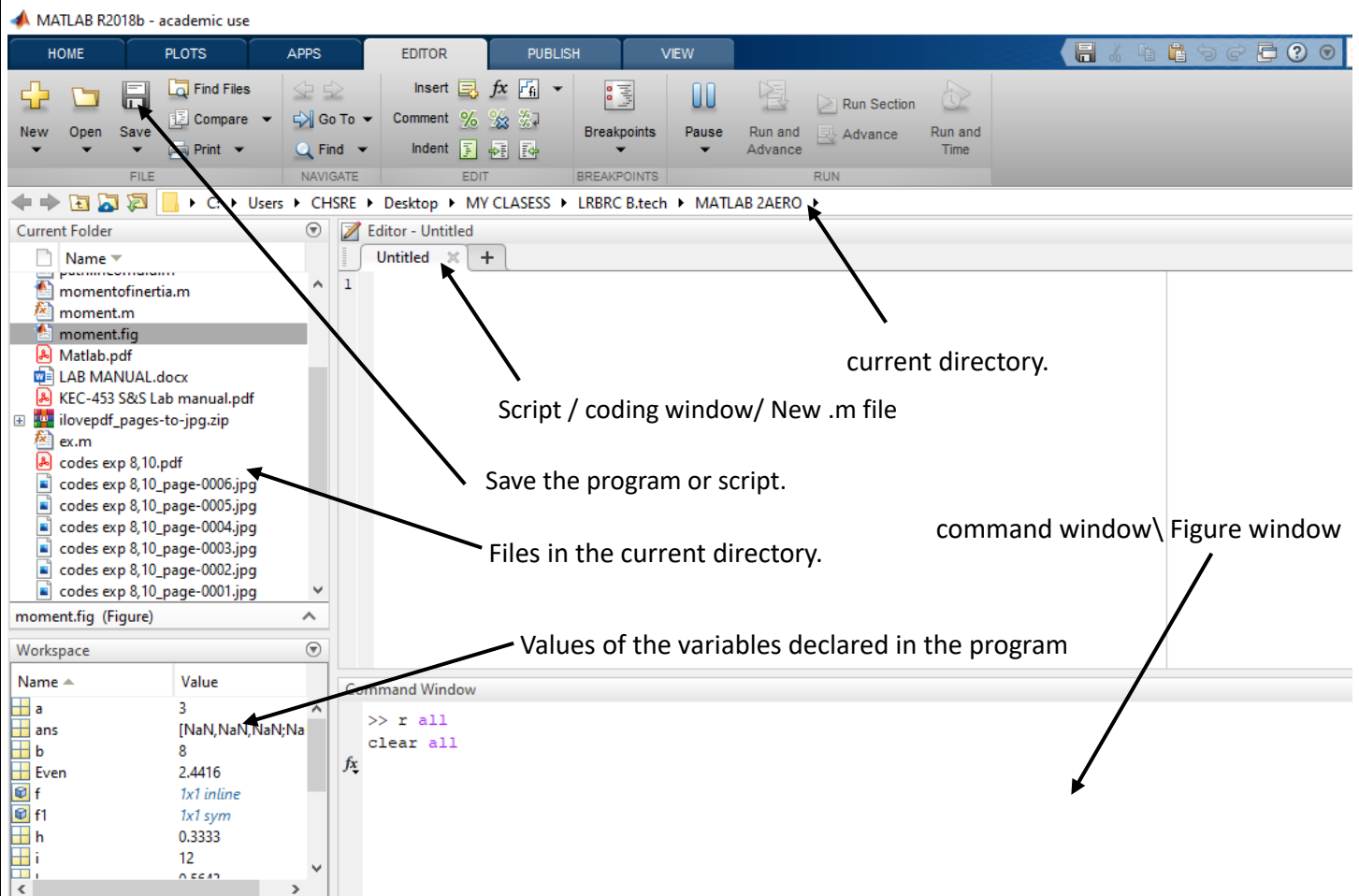


Fig1. Default programming windows in MATLAB

PROCEDURE: -

- Open MATLAB
- Open new M-file
- Type the program.
- Save in current directory.
- Compile and Run the program
- For the output see command window\ Figure window

2(a) Write a MATLAB program that adds all elements of an array named S with even indices.

Code:

```
% Define the array S
S = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]; % Example array
% Initialize the sum variable
sum_even_indices = 0;
% Loop through the array and sum the elements at even indices
for i = 1:2:length(S)
    sum_even_indices = sum_even_indices + S(i);
end
% Display the result
disp('Sum of elements with even indices:');
disp(sum_even_indices);
```

Explanation:

- The array S is defined as an example. You can change S to any array you want.
- The for loop starts at index 1 and increments by 2 to select only elements with even indices (1-based indexing in MATLAB).
- The sum of these elements is calculated and stored in “sum_even_indices”.
- Finally, the sum is displayed using disp.

Example output:

For the array $S = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$, the program will add the elements at indices 1, 3, 5, 7, and 9, which are $1 + 3 + 5 + 7 + 9 = 25$. The output will be 2

2b) Perform double integration problems.

Theory:

The double integral of a non-negative function $f(x, y)$ defined on a region in the plane tells us about the volume of the region under the graph. The double integral of a function of two variables, $f(x, y)$ over the region R can be expressed as follows :

$$\int \int_R f(x, y) dA = \int \int_R f(x, y) dx dy$$

Case 1: Double Integral of a Function

Let's solve the double integral of the function $f(x, y) = x^2 + y^2$ over the region where $0 \leq x \leq 2$

Code :

```
% Define the function to be integrated
```

```
f = @(x, y) x.^2 + y.^2;
```

```
% Perform the double integration
```

```
result = integral2(f, 0, 2, 0, 3);
```

```
% Display the result
```

```
disp('Result of the double integral:');
```

```
disp(result);
```

Explanation:

- f is a function handle for $f(x, y) = x^2 + y^2$
- `integral2(f, 0, 2, 0, 3)` performs the integration of $f(x, y)$ over the region $0 \leq x \leq 2$ and $0 \leq y \leq 3$
- The result will be displayed in the command window.

Case 2: Double Integral with Changing Limits

Code :

```
% Define the function to be integrated
```

```
f = @(x, y) x .* y;
```

```
% Perform the double integration with changing limits
```

```
result = integral2(f, 0, 2, @(x) x, 3);
```

```
% Display the result
```

```
disp('Result of the double integral with changing limits:');
```

```
disp(result);
```

Explanation:

- `f` is a function handle for $f(x, y) = x * y$.
- The `integral2` function uses `@(x) x` to define the lower limit for y as x , and the upper limit for y is 3.
- The result is displayed after the integration is complete.

Case 3: Symbolic Double Integration (Using Symbolic Toolbox)

If you want to use symbolic integration (i.e., finding the exact analytical result), you can use the `int` function from the Symbolic Math Toolbox.

Code :

```
% Define symbolic variables
syms x y;

% Define the function to be integrated
f = x^2 + y^2;

% Perform the double integration symbolically
integral_result = int(int(f, y, 0, 3), x, 0, 2);
% Display the result
disp('Result of the symbolic double integral:');
disp(integral_result);
```

Explanation:

- `syms x y` defines the symbolic variables for x and y .
- `f = x^2 + y^2` defines the function.
- `int(int(f, y, 0, 3), x, 0, 2)` performs the symbolic double integration first with respect to y (from 0 to 3), and then with respect to x (from 0 to 2).
- The result is displayed as an exact analytical answer.

RELATED QUESTIONS: -

1. How to call `diff()` in MATLAB?
2. How to call `syms()` in MATLAB?
3. What is the function of MATLAB function `clc`?
4. what is the application of `dsolve()` function in MATLAB?

3. Find the roots of a linear equation by using Newton's and secant Method

OBJECTIVE: -

i). Find the roots of a linear equation by using Newton's and secant Method

SOFTWARE USED: - MATLAB 2022a or 2019a

Theory:

To find the roots of a linear equation using **Newton's Method** and the **Secant Method** in MATLAB, we need to implement the respective algorithms. Let's assume the linear equation is of the form:

$$f(x) = ax + b = 0$$

The root of this equation is $x = -b/a$ but we will apply both methods for educational purposes and to demonstrate their implementation.

1. Newton's Method:

Newton's Method (also called the Newton–Raphson Method) is an iterative technique for finding roots of equations. It uses tangent lines to successively approximate the solution of $f(x) = 0$, often converging very quickly if the initial guess is close

linear equations, let's take a general form:

$$f(x) = ax + b = 0$$

The root is analytically trivial:

$$x = -\frac{b}{a}$$

But you want to see how **Newton's Method** and the **Secant Method** work when applied to this case. Let's do it step by step with MATLAB code.

Newton's Method

Newton's iteration formula is:

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

For a linear equation $f(x) = ax + b$:

- $f'(x) = a$ (constant).
- So the update becomes:

$$x_{n+1} = x_n - \frac{ax_n + b}{a} = -\frac{b}{a}$$

That means Newton's method finds the root in **one iteration** for a linear equation.

2. Secant Method:

The **Secant Method** is an iterative numerical technique for finding the root of a nonlinear equation. It uses two initial guesses and repeatedly draws a *secant line* (a line through two points on the curve) to approximate the root, avoiding the need for derivatives

What is the Secant Method?

- The **Secant Method** is a root-finding algorithm in numerical analysis.
- It is similar to Newton's Method but does **not require the derivative** of the function.
- Instead, it approximates the slope of the curve using a **secant line** through two points $(x_{n-1}, f(x_{n-1}))$ and $(x_n, f(x_n))$.
- The root is then estimated by finding where this secant line crosses the x-axis

Secant iteration formula is: $x_{n+1} = x_n - f(x_n) \cdot \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$ where

- $x_{n-1}, x_n \rightarrow$ two initial guesses.
- $f(x) \rightarrow$ the function whose root we want.
- Each iteration gives a better approximation of the root.

Finally, The Secant Method is a **quasi-Newton method** that uses secant lines instead of tangents to approximate roots. It's efficient when derivatives are hard to compute, but requires careful choice of starting points.

For a linear function, the secant method also converges **in one step**, because the secant line is the same as the function line.

PROCEDURE: -

- Open MATLAB
- Open new M-file
- Type the program.
- Save in current directory.
- Compile and Run the program
- For the output see command window\ Figure window

Code : Newton's Method:

% Newton's Method for linear equation $ax + b = 0$

```
a = 2;
b = -4; % Equation:  $2x - 4 = 0 \rightarrow \text{root} = 2$ 
f = @(x) a*x + b;
df = @(x) a;
x0 = 0; % initial guess
x1 = x0 - f(x0)/df(x0);
fprintf('Root using Newton method = %.4f\n', x1);
```

Code : Secant Method:*% Secant Method Iteration in MATLAB**% Define the function*f = @(x) x^3 - x - 2; % Example: root near $x \approx 1.618$ *% Initial guesses*

x0 = 1;

x1 = 2;

% Tolerance and maximum iterations

tol = 1e-6;

maxIter = 50;

% Iteration loop

for k = 1:maxIter

% Apply Secant formula

x2 = x1 - f(x1)*(x1 - x0)/(f(x1) - f(x0));

% Display iteration info

fprintf('Iteration %d: x = %.6f\n', k, x2);

% Check convergence

if abs(x2 - x1) < tol

fprintf('Root found at x = %.6f after %d iterations\n', x2, k);

break;

end

% Update values for next iteration

x0 = x1;

x1 = x2;

end

Explanation of the Code:

- **Newton's Method:**
 - The function $f(x)$ is $ax + b$, and the derivative $f'(x)$ is a .
 - The loop iterates to update the root estimate using the Newton's method formula, checking for convergence (if the difference between successive estimates is smaller than a given tolerance `tol`).
- **Secant Method:**
 - Two initial guesses, `x0` and `x1`, are used to start the iteration.
 - The loop iterates to update the root estimate using the secant method formula, checking for convergence similarly.

Both methods should converge quickly to the correct root of 222 because the equation is simple and linear. For more complex equations, the convergence and number of iterations may vary.

Try to solve the bellow example.

Example 1: Find the root for the linear equation $2x-4=0$

4. (a) Introduction to basic plots 2D, 3D

(b) MATLAB Graphical user interface addition and subtraction

OBJECTIVE: -

i) Plotting of simple 2D and 3D graph and MATLAB Graphical user interface addition and subtraction

SOFTWARE USED: - MATLAB 2022a or 2019a .

PROCEDURE: -

- Open MATLAB
- Open new M-file
- Type the program.
- Save in current directory.
- Compile and Run the program
- For the output see command window\ Figure window

2-D and 3-D Plots

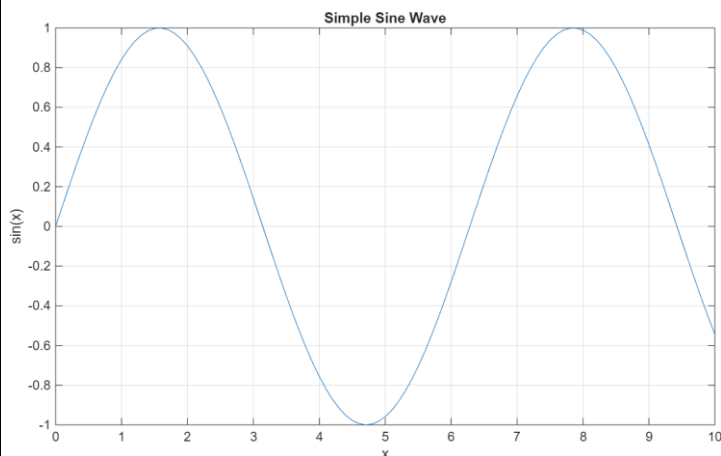
Line Plots

To create two-dimensional line plots, use the plot function. For example, plot the sine function over a linearly spaced vector of values from 0 to 2π :

Code :

Case-1: Simple Line Plot

```
x = 0:0.1:10;    % Define x range
y = sin(x);      % Compute y
plot(x, y);      % Plot
xlabel('x');      % Label x-axis
ylabel('sin(x)'); % Label y-axis
title('Simple Sine Wave');
```

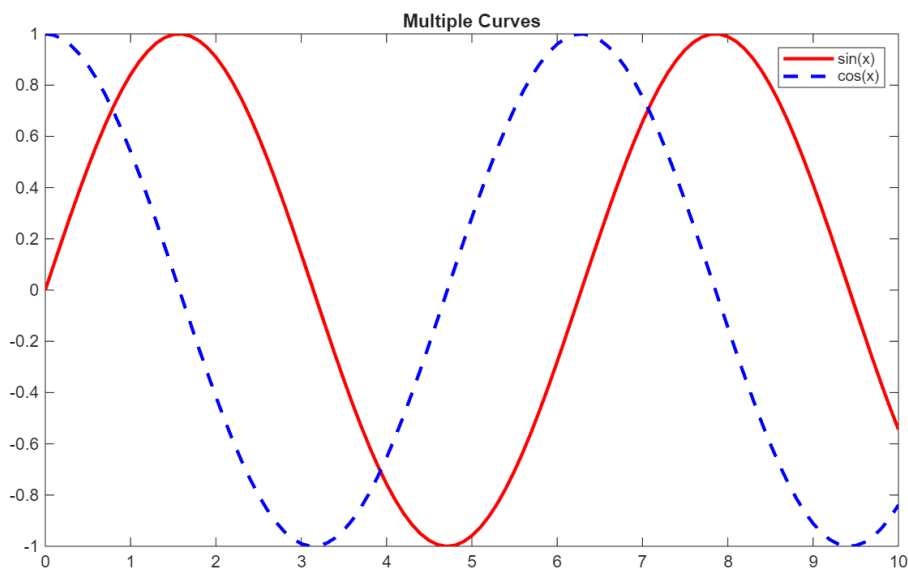


Explanation of code

- `plot()` draws 2D line graphs.
- `grid on` makes it easy to read values.
- Labels + title improve clarity.

Case-2: Multiple Curves on Same Plot

```
x = 0:0.1:10;
plot(x, sin(x), 'r', x, cos(x), 'b--', 'LineWidth', 2);
legend('sin(x)', 'cos(x)');
title('Multiple Curves');
```

**Explanation**

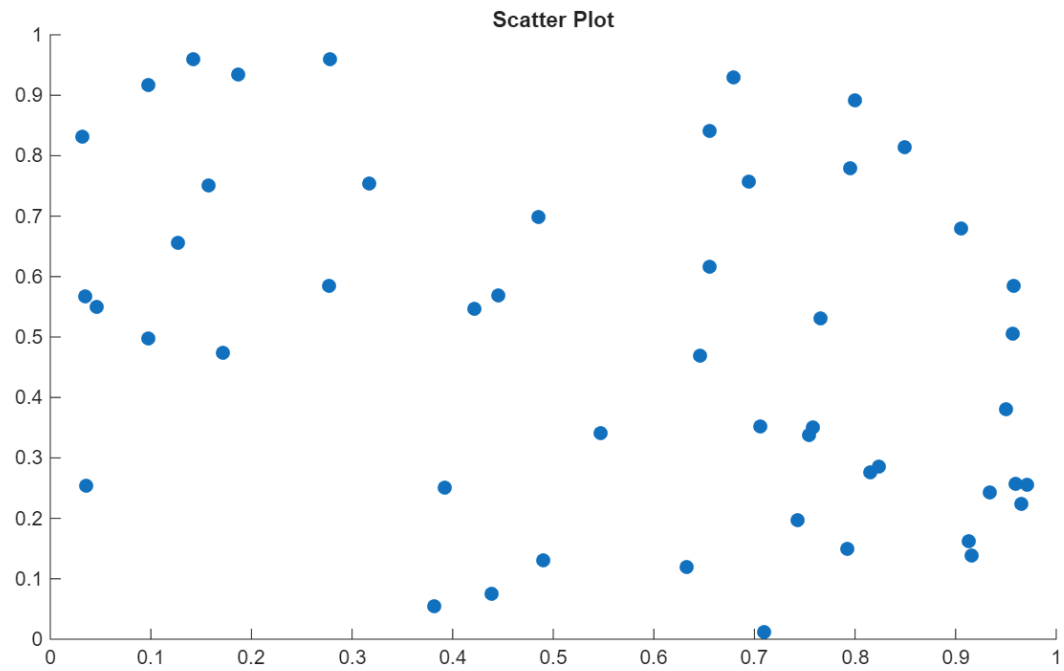
- `'r'` = red line
- `'b--'` = blue dashed
- `LineWidth` makes lines thicker
- `legend()` identifies curves

Case-3: Scatter Plot

```
x = rand(1,50);      % Random points
y = rand(1,50);
scatter(x, y, 50, 'filled');
title('Scatter Plot');
```

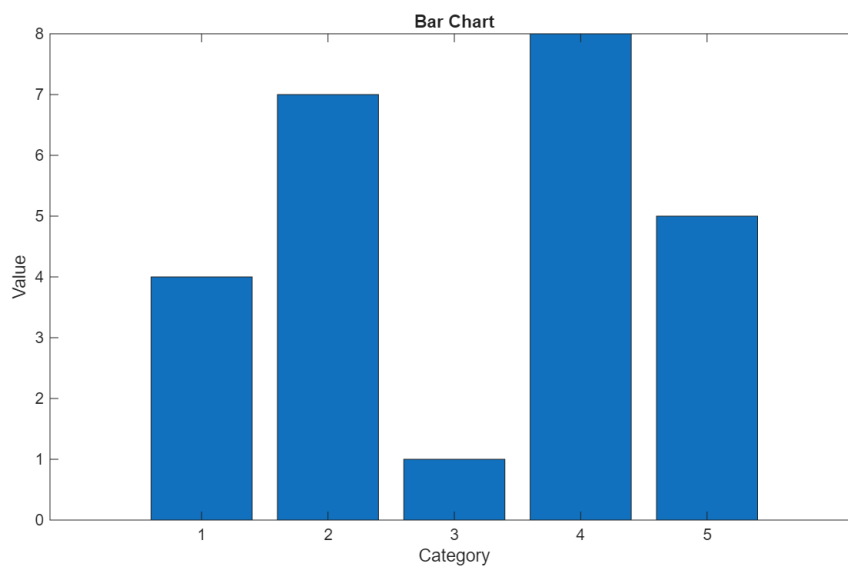
Explanation

- `scatter()` draws discrete points
- `50` = marker size



Case-4: Bar Chart

```
data = [4 7 1 8 5];  
bar(data);  
title('Bar Chart');  
xlabel('Category');  
ylabel('Value');
```



Case-5: Subplots (Multiple plots in one figure)

```
x = 0:0.1:10;
```

```
subplot(2,2,1);
```

```
plot(x, sin(x));
```

```
title('sin(x)');
```

```
subplot(2,2,2);
```

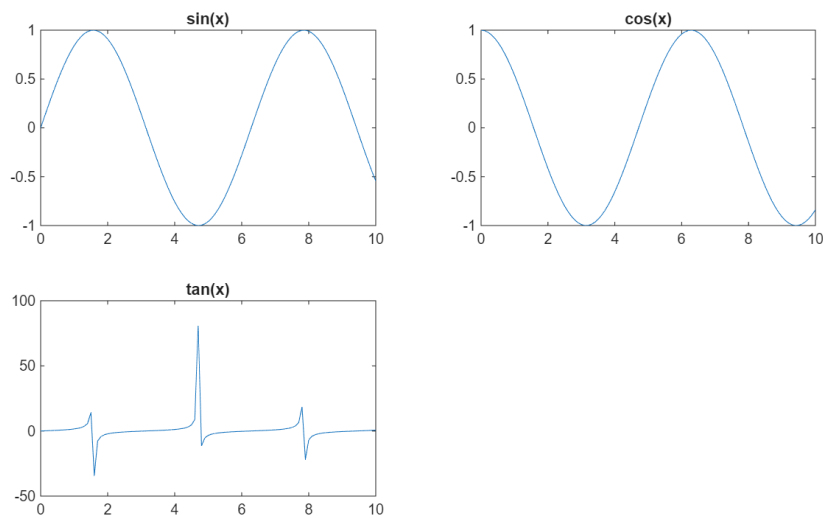
```
plot(x, cos(x));
```

```
title('cos(x)');
```

```
subplot(2,2,3);
```

```
plot(x, tan(x));
```

```
title('tan(x)');
```

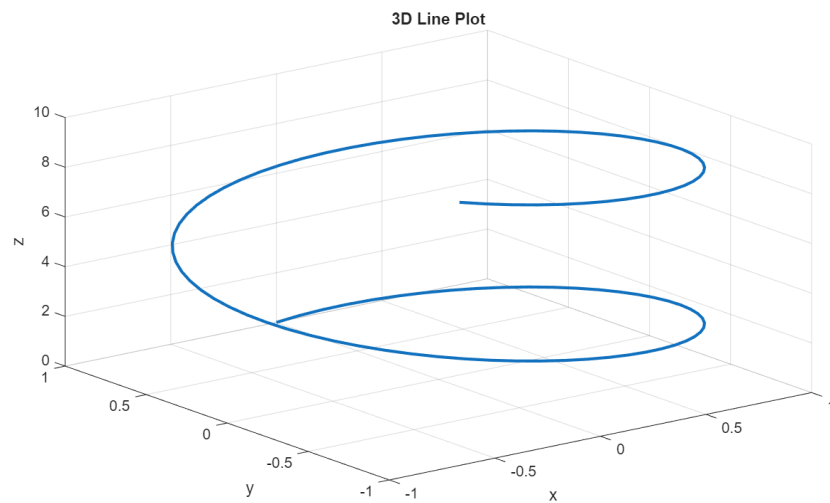
**Explanation**

- `subplot(m,n,p)` → grid of $m \times n$, place in p -th position

BASIC 3D PLOTS

Code for 3D Line Plot:

```
t = 0:0.1:10;  
x = sin(t);  
y = cos(t);  
z = t;  
  
plot3(x, y, z, 'LineWidth', 2);  
grid on;  
title('3D Line Plot');  
xlabel('x'); ylabel('y'); zlabel('z');
```

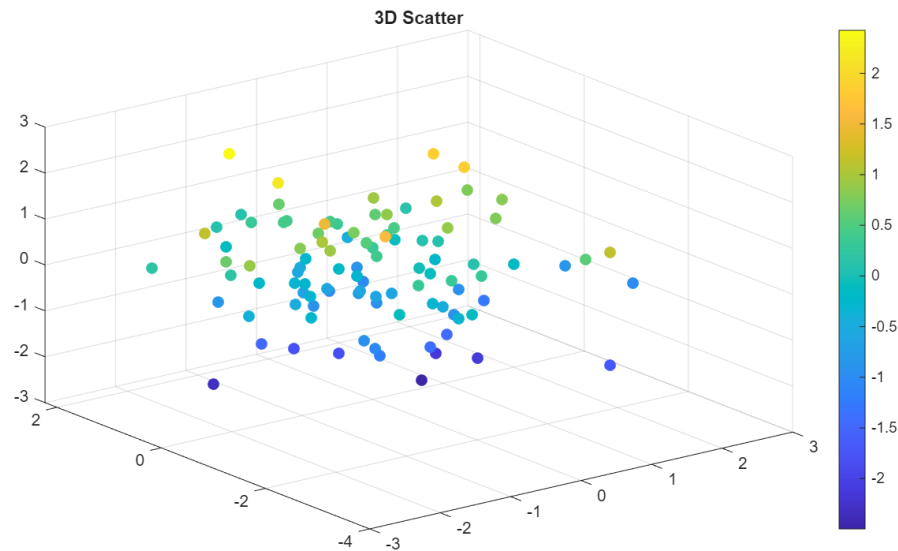


Explanation

- `plot3()` draws a 3D line
- Works like `plot` but with 3 axes

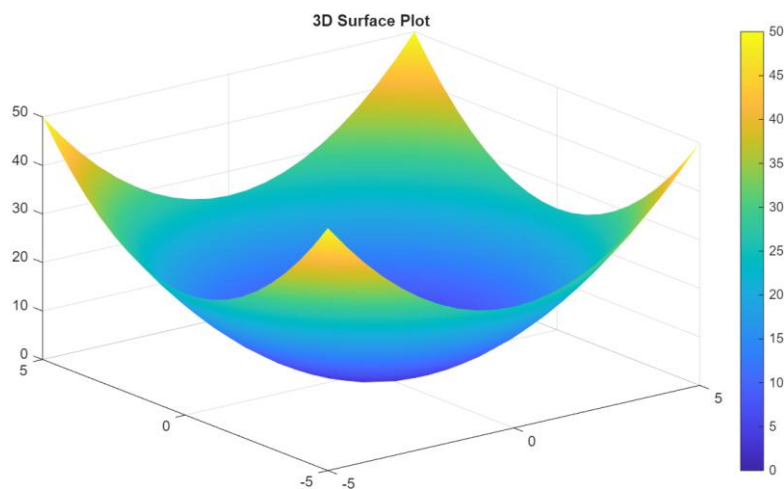
Code for 3D Scatter Plot:

```
x = randn(1,100);  
y = randn(1,100);  
z = randn(1,100);  
  
scatter3(x, y, z, 50, z, 'filled');  
colorbar;  
title('3D Scatter');
```



2.3 Surface Plot (Very Important)

```
[X,Y] = meshgrid(-5:0.5:5);
Z = X.^2 + Y.^2;      % Example surface
surf(X, Y, Z);
shading interp;
colorbar;
title('3D Surface Plot');
```

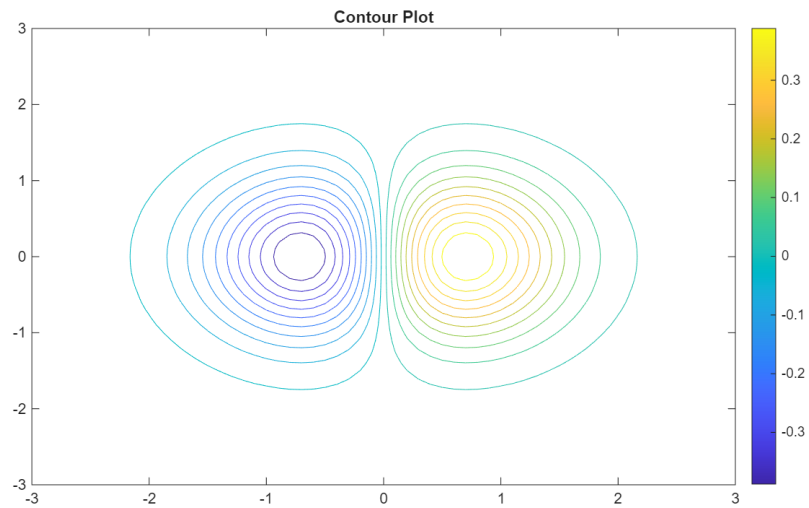
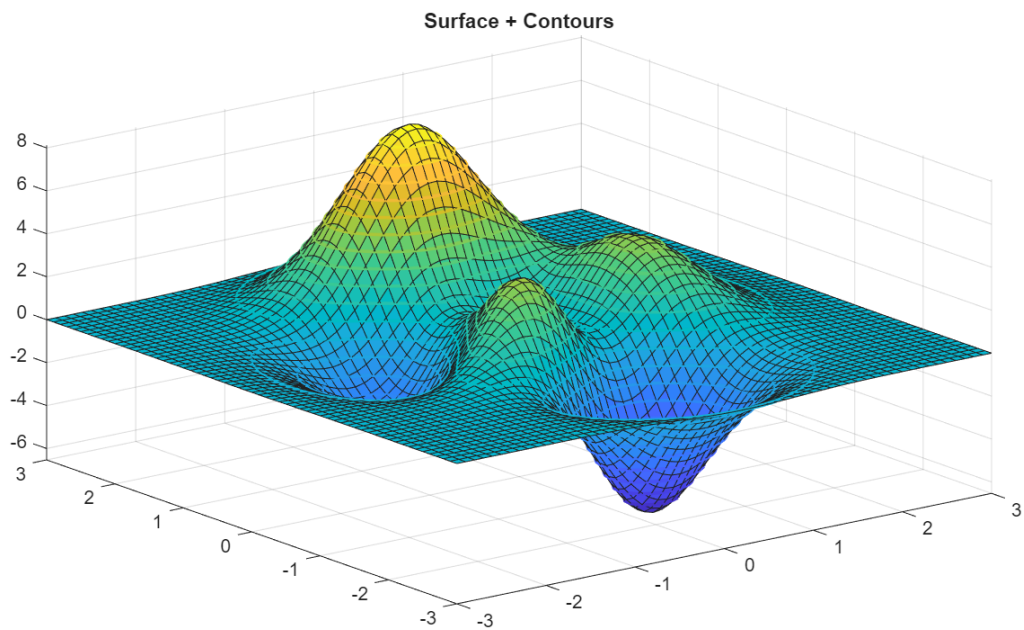


Explanation

- meshgrid produces a 2D grid
- Z defines height at each (X,Y)
- surf() draws a colored 3D surface
- shading interp smooths colors

Code for Contour Plot (2D view of 3D surface)

```
[X, Y] = meshgrid(-3:0.1:3);  
Z = X .* exp(-X.^2 - Y.^2);  
contour(X, Y, Z, 20);  
colorbar;  
title('Contour Plot');
```

**Code for COMBINING SURFACE + CONTOUR**

Finally Recall

Plot Type	Function
Line plot	plot()
Multiple lines	plot(x1,y1,x2,y2,...)
Scatter	scatter()
Bar	bar()
Subplots	subplot()
3D line	plot3()
3D scatter	scatter3()
Surface	surf()
Mesh	mesh()
Contour	contour(), contour3()

(b) Introduction to graphical user interface addition and subtraction**PROCEDURE: -**

- Open MATLAB
- Open new M-file
- Use the command `GUID` to open the GUI controller of the MATLAB.
- Save in current directory.
- Create the required text boxes, push buttons, static text boxes, radio buttons etc
- Save the program with a proper name.
- Code the push buttons with required coding
- Compile and run the program.
- For the output see command window\ Figure window\ gui window

Theory:

Graphical user interfaces (GUIs), also known as apps, provide point-and-click control of your software applications, eliminating the need for others to learn a language or type commands in order to run the application. You can share apps both for use within MATLAB and also as standalone desktop or web apps.

You can choose from the following three ways to create an app in MATLAB:

- Convert a script into a simple app: Choose this option when you want to share a script with students or colleagues and allow them to modify variables using interactive controls.
- Create an app interactively: Choose this option when you want to create a more sophisticated app using a drag-and-drop environment to build the user interface.
- Create an app programmatically: Choose this option when you want to create an app's user interface by writing the code yourself.

Convert a Script into a Simple App

Use the Live Editor to convert a script into a simple app that has interactive controls allowing others to experiment with variables in your code. Add sliders, dropdowns, edit fields, and buttons without writing any code. Specify what parts of the script will run when a value is changed. Hide the code to create simple apps and dashboards.

Create an App Interactively

App Designer is an interactive environment that integrates the two primary tasks of app building: laying out the visual components and programming the app's behaviour. It allows you to quickly move between visual design in the canvas and developing code in the MATLAB editor.

You can share your app with others to use in MATLAB on the desktop or in a web browser using MATLAB Online. App Designer apps can also be packaged for installation into the MATLAB Apps tab. To share with non-MATLAB users, you can compile apps into standalone desktop and web apps using MATLAB Compiler.

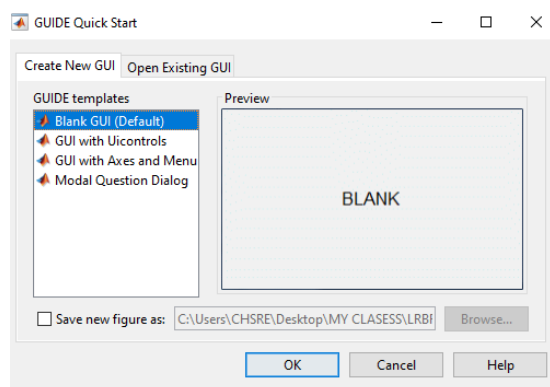
App Designer is good for interactively designing your layout and programming its behaviours in one environment. If you prefer, you can program the entire app yourself, including the user interface.

Create an App Programmatically

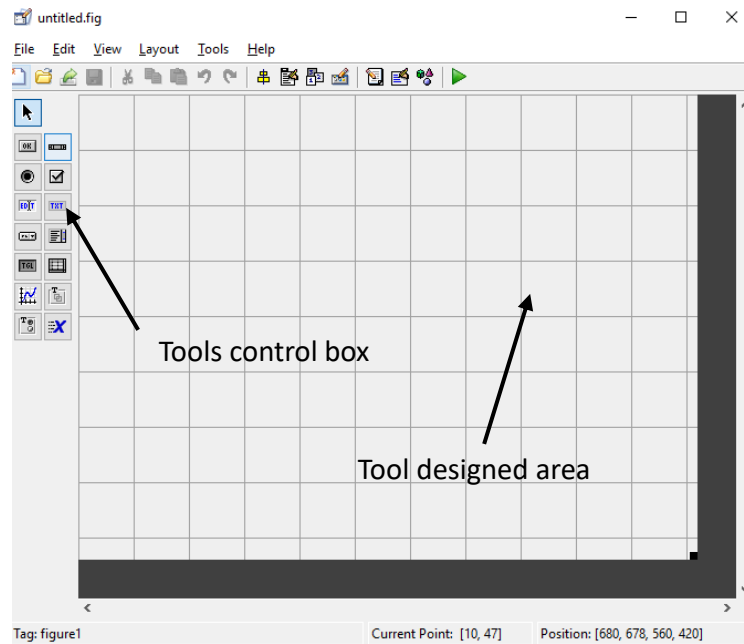
For added control over design and development, you can also use MATLAB functions to define the layout and behaviours of your app. In this approach, you create a figure to serve as the container for your user interface and add components to it programmatically.

Steps to be followed:

1. In the command window type "guide" then u will get this following window

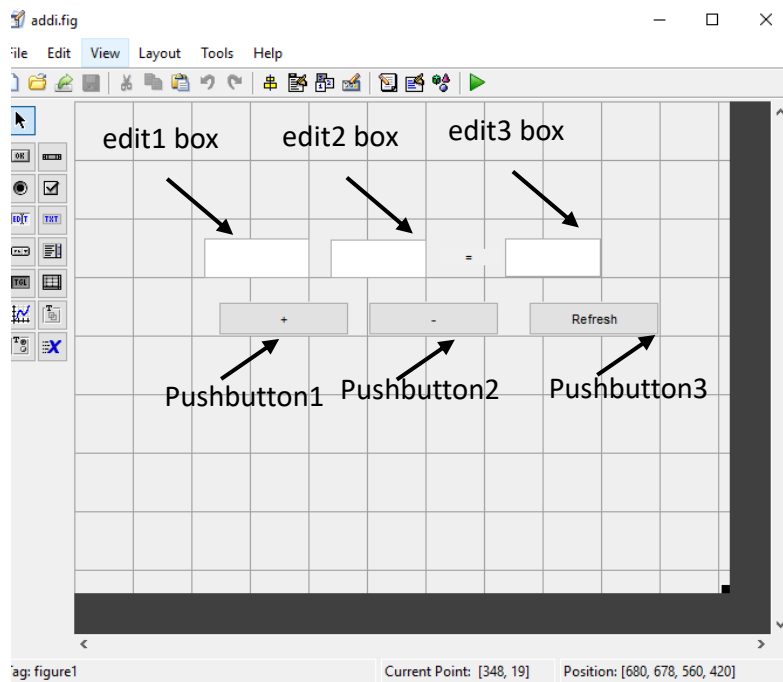


2. Select Blank GUI (default) by clicking ok button. Then bellow window will be appear.



The App Designer interface

3. Creat the App Designer interface like bellow for our program



3. Save the GUI interface with the proper name in this case I used “addi”
4. Once saving done automatic code will be created by the MATLAB coding area like bellow

```

function varargout = addi(varargin)
% ADDI MATLAB code for addi.fig
%   ADDI, by itself, creates a new ADDI or raises the existing
%   singleton*.
%
%   H = ADDI returns the handle to a new ADDI or the handle to
%   the existing singleton*.
%
%   ADDI('CALLBACK',hObject,eventData,handles,...) calls the local
%   function named CALLBACK in ADDI.M with the given input arguments.
%
%   ADDI('Property','Value',...) creates a new ADDI or raises the
%   existing singleton*. Starting from the left, property value pairs are
%   applied to the GUI before addi_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property application
%   stop. All inputs are passed to addi_OpeningFcn via varargin.
%

```

5. Now coding part

In the coding area find the followings and make the code as below

- For addition push button in the GUI interface

```

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
clear a
clear b
a=str2num(get(handles.edit1,'string'));
b=str2num(get(handles.edit2,'string'));
c=a+b;
set(handles.edit3,'string',num2str(c));

```

- For subtraction push button in the GUI interface

```

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton2 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
clear a
clear b
a=str2num(get(handles.edit1,'string'));
b=str2num(get(handles.edit2,'string'));
d=a-b;
set(handles.edit3,'string',num2str(d))

```

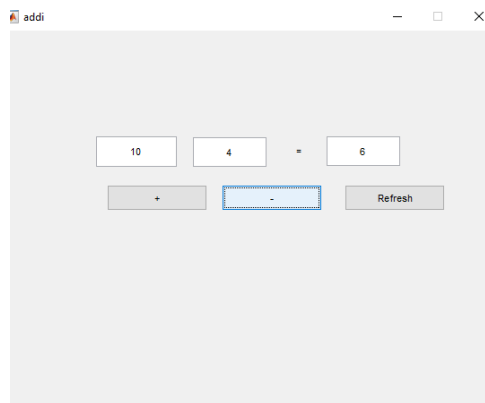
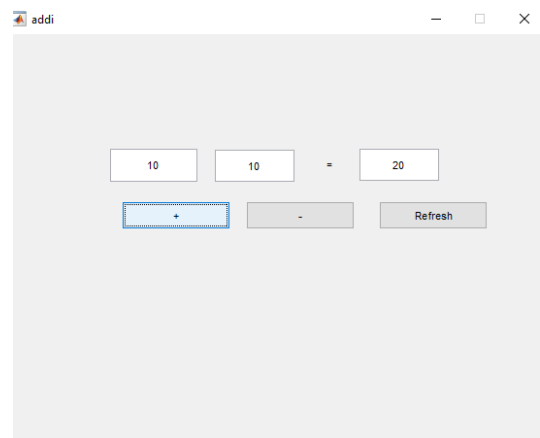
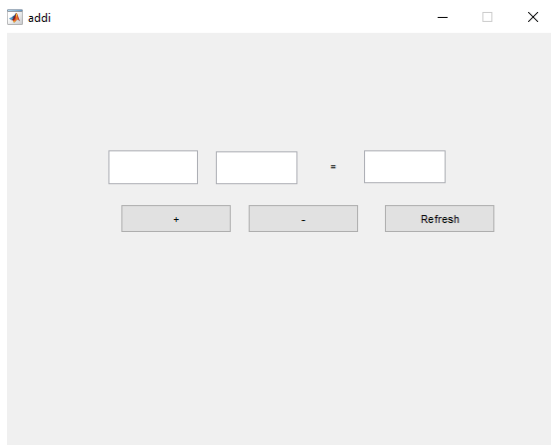
- For Refresh push button in the GUI interface

```

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton3 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
set(handles.edit1,'string','')
set(handles.edit2,'string','')
set(handles.edit3,'string','')

```

The output will be



RELATED QUESTIONS: -

1. Create axes and add a title for 3d plot?
2. Create legend for a 3d plot?
3. Create GUI for multiplication using MATLAB Gui interface!
4. Create GUI for Division using MATLAB Gui interface!

Part II:

Application of MATLAB

5. Write a MATLAB Code to determine Lift Curve Slope from the given parameters

OBJECTIVE: -

i) Write A Matlab Code To Determine Lift Curve Slope From The Given Parameters

SOFTWARE USED: - MATLAB 2022a or 2019a

PROCEDURE: -

- Open MATLAB
- Open new M-file
- Type the program.
- Save in current directory.
- Compile and run the program.
- For the output see command window\ Figure window

Theory:

The **lift curve slope** is a fundamental concept in **aerodynamics**, representing how the lift coefficient (C_L) of an Airfoil or wing changes with the angle of attack (α). It indicates the sensitivity of lift generation to changes in angle of attack.

Lift Curve Slope Definition

The **lift curve slope** is mathematically expressed as: $\frac{dC_L}{d\alpha}$

Where:

- dC_L = Lift coefficient
- $d\alpha$ = Angle of attack (in radians or degrees, depending on the context)

This slope describes how much the lift coefficient increases for each unit increase in the angle of attack.

Theoretical Lift Curve Slope (Thin Airfoil Theory)

For an ideal 2D thin airfoil in incompressible flow, the theoretical lift curve slope is given by:

$$\frac{dC_L}{d\alpha} = 2\pi$$

Assumptions:

1. Incompressible flow (low subsonic speeds, typically $M < 0.3$)
2. Thin airfoil approximation
3. Attached flow (no significant flow separation)

In real-world scenarios, the lift curve slope is slightly less than 2π due to viscosity, compressibility, and three-dimensional effects.

Factors Affecting the Lift Curve Slope

1. Aspect Ratio (AR):

- For finite wings, the lift curve slope is reduced due to wingtip vortices.
- The lift curve slope for a finite wing can be approximated by:

$$\frac{dC_L}{d\alpha} = \frac{2\pi AR}{2 + \sqrt{4 + AR^2(1 + \tan^2(\Lambda))}}$$

Where:

- AR : Aspect Ratio
- Λ : Sweep angle

2. Compressibility Effects:

- At higher Mach numbers ($M > 0.3$), compressibility reduces the lift curve slope.

3. Flow Separation:

- At high angles of attack, flow separation causes a deviation from the linear lift curve.

4. Airfoil Shape:

- Cambered airfoils have a different baseline lift curve slope compared to symmetric airfoils.

Code:

```
% Lift Curve Slope Calculation and Plotting for 2D and Finite Wings
% Author: [Dr. Sreenadh Chevula]
% Date: [07/01/2025]
clc;
clear;
close all;

%% Input Parameters
% Airfoil and Wing Properties
AR = 8; % Aspect Ratio (e.g., 8 for a typical wing)
sweep_angle_deg = 0; % Sweep angle in degrees (0 for unswept wings)
M = 0.2; % Mach number (below 0.3 for incompressible flow)
alpha_range = -5:1:15; % Angle of attack range (degrees)

%% Constants
gamma = 1.4; % Ratio of specific heats (air)
beta = sqrt(1 - M^2); % Prandtl-Glauert Compressibility Correction

%% Lift Curve Slope for Ideal 2D Thin Airfoil
CL_alpha_2D = 2 * pi; % Thin airfoil theory for incompressible flow

if M > 0.3
    % Apply compressibility correction
    CL_alpha_2D = CL_alpha_2D / beta;
end
```

```

%% Lift Curve Slope for Finite Wing (Using Prandtl's Lifting Line Theory)
sweep_angle_rad = deg2rad(sweep_angle_deg);

CL_alpha_finite = (2 * pi * AR) / ...
    (2 + sqrt(4 + AR^2 * (1 + tan(sweep_angle_rad)^2)));

if M > 0.3
    % Apply compressibility correction
    CL_alpha_finite = CL_alpha_finite / beta;
end

%% Calculate Lift Coefficient for Each Angle of Attack
alpha_rad = deg2rad(alpha_range); % Convert angles to radians

CL_2D = CL_alpha_2D * alpha_rad; % Lift coefficient for 2D airfoil
CL_finite = CL_alpha_finite * alpha_rad; % Lift coefficient for finite wing

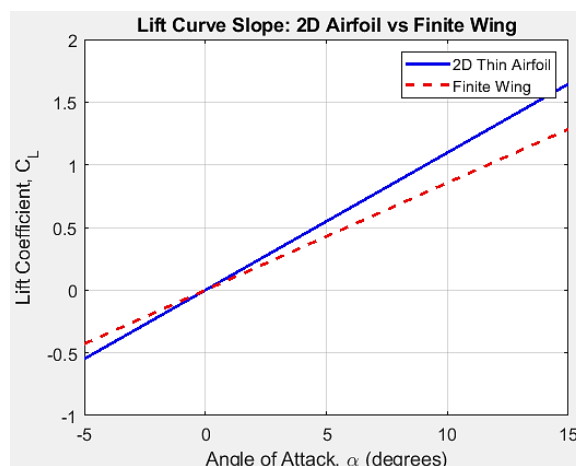
%% Plot Lift Curve
figure;
plot(alpha_range, CL_2D, 'b-', 'LineWidth', 2); hold on;
plot(alpha_range, CL_finite, 'r--', 'LineWidth', 2);
grid on;

xlabel('Angle of Attack, \alpha (degrees)');
ylabel('Lift Coefficient, C_L');
title('Lift Curve Slope: 2D Airfoil vs Finite Wing');
legend('2D Thin Airfoil', 'Finite Wing');
set(gca, 'FontSize', 12);

%% Display Results in Command Window
fprintf('---- Lift Curve Slope Calculation ----\n');
fprintf('Aspect Ratio (AR): %.2f\n', AR);
fprintf('Sweep Angle (degrees): %.2f\n', sweep_angle_deg);
fprintf('Mach Number (M): %.2f\n', M);
fprintf('-----\n');
fprintf('Ideal 2D Lift Curve Slope (dCL/d?): %.4f per radian\n', CL_alpha_2D);
fprintf('Finite Wing Lift Curve Slope (dCL/d?): %.4f per radian\n', CL_alpha_finite);
fprintf('-----\n');

```

Output:



---- Lift Curve Slope Calculation ----

Aspect Ratio (AR): 8.00

Sweep Angle (degrees): 0.00

Mach Number (M): 0.20

Ideal 2D Lift Curve Slope ($dCL/d\alpha$): 6.2832 per radian

Finite Wing Lift Curve Slope ($dCL/d\alpha$): 4.9058 per radian

6. Solving of ordinary differential equation using Runge-Kutta method numerical approach

OBJECTIVE: -

i) Solving of ordinary differential equation using Runge-Kutta method numerical approach

SOFTWARE USED: - MATLAB 2022a or 2019a

PROCEDURE: -

- Open MATLAB
- Open new M-file
- Type the program.
- Save in current directory.
- Compile and run the program.
- For the output see command window\ Figure window

Theory:

Runge-Kutta method is a popular iteration method of approximating solution of ordinary differential equations. Developed around 1900 by German mathematicians C. Runge and M. W. Kutta, this method is applicable to both families of explicit and implicit functions.

Also known as RK method, the Runge-Kutta method is based on solution procedure of initial value problem in which the initial conditions are known.

Based on the order of differential equation, there are different Runge-Kutta methods which are commonly referred to as: RK2, RK3, and RK4 methods.

Let's consider an initial value problem given as: $y' = f(t, y)$, $y(t_0) = y_0$

where,

- y is an unknown function of time t , which may be scalar or vector quantity
- y' is the rate of change of y with respect to t
- t_0 is the initial value of t
- y_0 is the value of y at time t_0
- Let 'h' be the step size such that $h > 0$. Now, the generation term of the series can be defined as:
- $y_{n+1} = y_n + \frac{h}{6} * (k_1 + 2k_2 + 2k_3 + k_4)$ $t_{n+1} = t_n + h$

for $n = 1, 2, 3, 4, \dots$ such that:

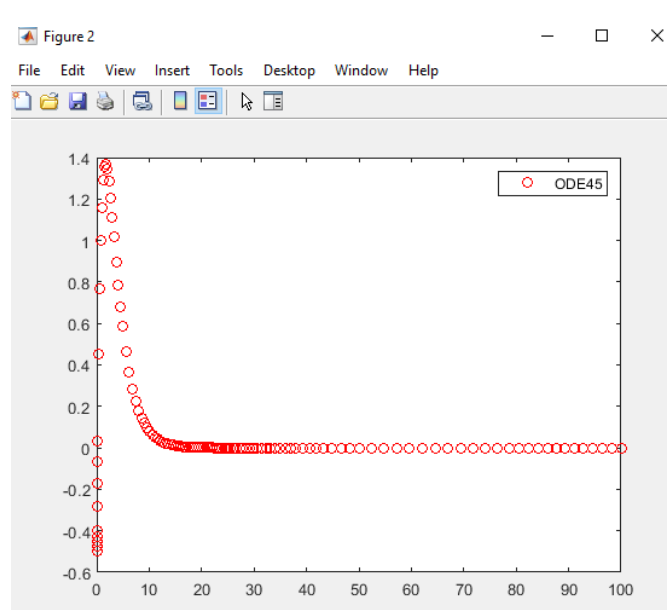
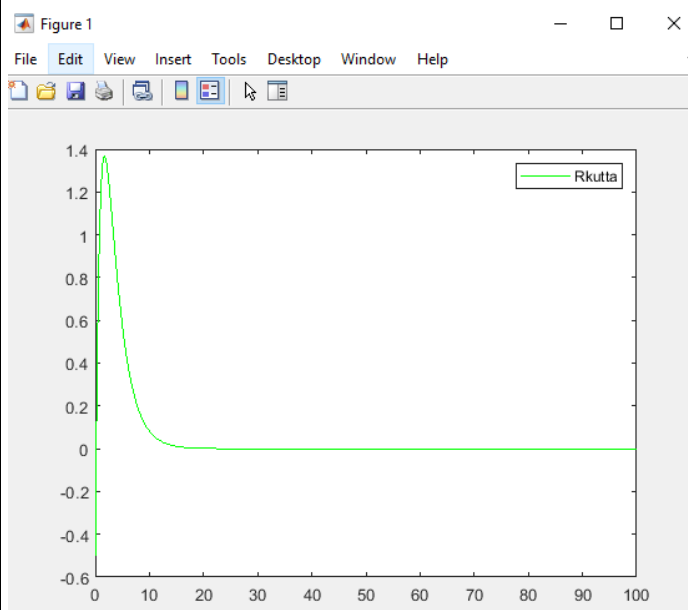
- $k_1 = f(t_n, y_n)$
- $k_2 = f(t_n + h/2, y_n + h/2 k_1)$
- $k_3 = f(t_n + h/2, y_n + h/2 k_2)$
- $k_4 = f(t_n + h, y_n + h k_3)$
- y_{n+1} is the Runge-Kutta method approximation of $y(t_{n+1})$
- k_1 is the increment which depends on the gradient of starting interval as in Euler's method
- k_2 is the increment which relies on the slope at the midpoint of the interval, $k_2 = y + h/2 * k_1$
- k_3 is also an increment based on the gradient at midpoint, $k_3 = y + h/2 * k_2$
- k_4 is again an increment whose value depends on end of the end of interval $k_4 = y + h k$

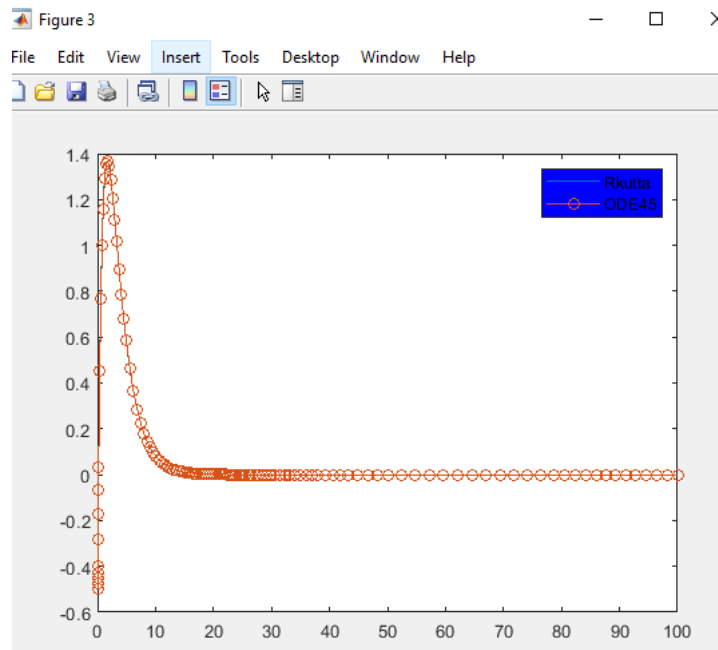
Code

```

h=0.05; % step size
x = 0:h:100; % Calculates up to y(3)
y = zeros(1,length(x));
y(1) = -0.5; % redo with other choices here.% initial condition
F_xy = @(t,r) 3.*exp(-t)-0.4*r; % change the function as you desire
for i=1:(length(x)-1) % calculation loop
    k_1 = F_xy(x(i),y(i));
    k_2 = F_xy(x(i)+0.5*h,y(i)+0.5*h*k_1);
    k_3 = F_xy((x(i)+0.5*h),(y(i)+0.5*h*k_2));
    k_4 = F_xy((x(i)+h),(y(i)+k_3*h));
    y(i+1) = y(i) + (1/6)*(k_1+2*k_2+2*k_3+k_4)*h; % main equation
end
% validate using a decent ODE integrator
tspan = [0,100]; y0 = y(1);
[tx, yx] = ode45(F_xy, tspan, y0);
%plot(x,y,'o', tx, yx, '--')
figure()
plot(x,y,'-', 'color', 'g')
legend('Rkutta');
figure()
plot(tx,yx,'o','color', 'r')
legend('ODE45');
figure()
plot(x,y,'-', tx, yx, '-o')
legend('Rkutta','ODE45','color', 'g','color', 'b');

```

Out put

**RELATED QUESTIONS: -**

1. Explain about Runge-Kutta method numerical approach?
2. Explain about the functions of `legend()`, `figure()`, `plot()` ?
3. Explain about the functions of `ode45()` ?

7. (a) Write a MATLAB Code to determine and Compute the Laplace transform of $1/\sqrt{x}$
- (b) Write a MATLAB Code to determine Eigenvalues and eigenvectors of a linear algebraic equations

OBJECTIVE: -

- i) 7. (a) Write a MATLAB Code to determine Compute the Laplace transform of $1/\sqrt{x}$
- (b) Write a MATLAB Code to determine Eigenvalues and eigenvectors of a linear algebraic equations

SOFTWARE USED: - MATLAB 2022a or 2019a

PROCEDURE: -

- Open MATLAB
- Open new M-file
- Type the program.
- Save in current directory.
- Compile and run the program.
- For the output see command window\ Figure window

Theory:

The Laplace Transform of $f(x) = \frac{1}{\sqrt{x}}$ is given by the integral:

$$\mathcal{L} \left\{ \frac{1}{\sqrt{x}} \right\} = \int_0^{\infty} \frac{1}{\sqrt{x}} e^{-sx} dx$$

Solution using Gamma Function

Using the **Gamma Function** identity for the Laplace Transform:

$$\int_0^{\infty} x^{p-1} e^{-sx} dx = \frac{\Gamma(p)}{s^p}, \quad \text{Re}(p) > 0, s > 0$$

1. Rewrite $\frac{1}{\sqrt{x}}$ as a power of x :

$$\frac{1}{\sqrt{x}} = x^{-\frac{1}{2}}$$

2. In this case:

$$p - 1 = -\frac{1}{2} \Rightarrow p = \frac{1}{2}$$

3. Substitute into the Gamma formula:

$$\int_0^{\infty} x^{-\frac{1}{2}} e^{-sx} dx = \frac{\Gamma\left(\frac{1}{2}\right)}{s^{1/2}}$$

4. From the Gamma Function:

$$\Gamma\left(\frac{1}{2}\right) = \sqrt{\pi}$$

5. Substitute into the equation:

$$\mathcal{L}\left\{\frac{1}{\sqrt{x}}\right\} = \sqrt{\frac{\pi}{s}}$$

Code:

```
% Laplace Transform of 1/sqrt(x)
% Author: [Dr. Sreenadh Chevula]
% Date: [07/1/2025]

clc;
clear;
close all;

%% Define the symbolic variable and function
syms x s

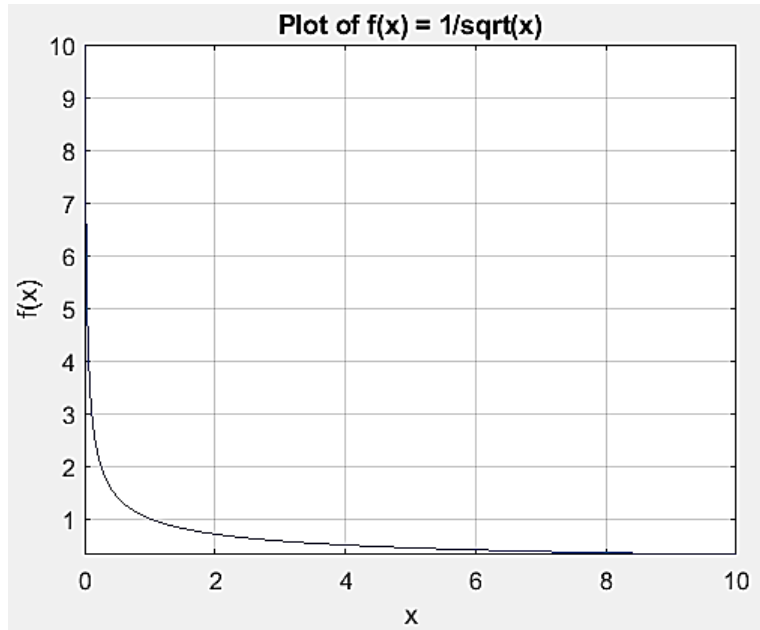
% Define the function
f = 1/sqrt(x);

%% Compute the Laplace Transform
F = laplace(f, x, s);

%% Display the Result
fprintf('The Laplace Transform of 1/sqrt(x) is:\n');
disp(F);

%% Plot the Original Function
fplot(f, [0.01, 10]);
grid on;
xlabel('x');
ylabel('f(x)');
title('Plot of f(x) = 1/sqrt(x)');
set(gca, 'FontSize', 12);
```

Output:



The Laplace Transform of $1/\sqrt{x}$ is:
 $\pi^{1/2}/s^{1/2}$

(b) Write a MATLAB Code to determine Eigenvalues and eigenvectors of a linear algebraic equations

Theory:

Eigenvalues and Eigenvectors of a System of Linear Algebraic Equations

Eigenvalues and eigenvectors are associated with square matrices derived from a system of linear algebraic equations. They represent fundamental properties of linear transformations.

1. Key Concepts

- **Eigenvalue (λ):** A scalar value such that:

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

- **Eigenvector ($\mathbf{v}$):** A nonzero vector that satisfies the above equation.
- **Matrix Form of Linear Equations:**

A system of linear algebraic equations can be written as:

$$A\mathbf{x} = \mathbf{b}$$

Where:

- A : Coefficient matrix
- $\mathbf{x}$: Unknown vector
- $\mathbf{b}$: Right-hand side vector

Eigenvalues and eigenvectors are typically computed from the coefficient matrix A .

2. Eigenvalues and Eigenvectors Calculation

For a matrix A , eigenvalues are found by solving:

$$\det(A - \lambda I) = 0$$

Where:

- I : Identity matrix of the same size as A
- $\det$: Determinant

Eigenvectors are then found by solving:

$$(A - \lambda I)\mathbf{v} = 0$$

Code:

```
% Eigenvalues and Eigenvectors Calculation
% Author: [Dr. Sreenadh Chevula]
% Date: [07/1/2025]
```

```

clc;
clear;
close all;

%% Define the Coefficient Matrix (A)
% Example matrix (change this as needed)
A = [4, 2, 1; 0, 3, -1; 0, 0, 2];
% Display the matrix
disp('The input matrix A is:');
disp(A);

%% Compute Eigenvalues and Eigenvectors
[eigenVectors, eigenValues] = eig(A);

% Extract eigenvalues from diagonal matrix
eigenValues_diag = diag(eigenValues);

%% Display Results
fprintf('Eigenvalues of the matrix are:\n');
disp(eigenValues_diag);

fprintf('Eigenvectors of the matrix are:\n');
disp(eigenVectors);

```

Output:

The input matrix A is:

```

4   2   1
0   3  -1
0   0   2

```

Eigenvalues of the matrix are:

```

4
3
2

```

Eigenvectors of the matrix are:

```

1.0000 -0.8944 -0.7276
0   0.4472  0.4851
0     0   0.4851

```

Verification for Eigenvalue $\lambda = 4.0000$:

$A * v$:

4
0
0

$\lambda * v$:

4
0
0

Difference (should be close to zero):

0
0
0

Verification for Eigenvalue $\lambda = 3.0000$:

$A * v$:

-2.6833
1.3416
0

$\lambda * v$:

-2.6833
1.3416
0

Difference (should be close to zero):

0
0
0

Verification for Eigenvalue $\lambda = 2.0000$:

$A * v$:

-1.4552

0.9701

0.9701

$\lambda * v$:

-1.4552

0.9701

0.9701

Difference (should be close to zero):

0

0

0

8. (a) Solve System of Linear Equations (b) Solve system of nonlinear equations

OBJECTIVE: -

i) (a) Solve System of Linear Equations (b) Solve system of nonlinear equations

SOFTWARE USED: - MATLAB 2024a, 2022a or 2019a

PROCEDURE: -

- Open MATLAB
- Open new M-file
- Type the program.
- Save in current directory.
- Compile and run the program.
- For the output see command window\ Figure window

Theory:

A system of linear equations is a collection of two or more linear equations involving the same set of variables. These systems can be solved to find values of the variables that satisfy all the equations simultaneously.

1. General Form

A system of linear equations can be written as:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m \end{aligned}$$

Where:

- a_{ij} : Coefficients of the variables
- $x_1, x_2, \dots, x_n$: Unknown variables
- $b_1, b_2, \dots, b_m$: Constants

In matrix form, this can be written as:

$$Ax = b$$

Where:

- A : Coefficient matrix
- x : Column vector of unknowns
- b : Column vector of constants

2. Types of Solutions

A system of linear equations can have:

1. **Unique Solution:** When the determinant of A is nonzero ($\det(A) \neq 0$).
2. **No Solution:** Inconsistent system (e.g., parallel lines in 2D).
3. **Infinite Solutions:** When equations are dependent (e.g., overlapping lines in 2D).

3. Methods to Solve Systems of Linear Equations

1. Gaussian Elimination
2. Gauss-Jordan Elimination
3. Matrix Inversion Method
4. LU Decomposition
5. Cramer's Rule (for square systems)
6. Iterative Methods (e.g., Jacobi, Gauss-Seidel)

6. Example Input and Output

For the system:

$$\begin{aligned} 2x - y + z &= 3 \\ x + 3y + 2z &= 12 \\ x - y + 2z &= 5 \end{aligned}$$

Solution:

$$x = 1, y = 2, z = 3$$

Code

```

% Solving System of Linear Equations
% Author: [Your Name]
% Date: [Date]

clc;
clear;
close all;

%% Define the Coefficient Matrix (A) and Constant Matrix (b)
A = [2, -1, 1;
     1, 3, 2;
     1, -1, 2]; % Coefficients of the variables

b = [3; 12; 5]; % Constants on the RHS

disp('Coefficient Matrix (A):');
disp(A);

disp('Constant Vector (b):');
disp(b);

%% Solve the System using Matrix Division
x = A\b; % Solves Ax = b

disp('Solution Vector (x):');
disp(x);

%% Verification
disp('Verification (A * x):');
disp(A * x);
disp('RHS Vector (b):');
disp(b);

if isequal(round(A * x, 5), round(b, 5))
    disp('The solution satisfies the system of equations.');
```

output:

Coefficient Matrix (A):

```

2  -1  1
1   3  2
1  -1  2
```

Constant Vector (b):

```

3
12
5
```

Solution Vector (x):

0.9167
1.7500
2.9167

Verification (A * x):

3
12
5

RHS Vector (b):

3
12
5

The solution satisfies the system of equations.

(b) Solve system of nonlinear equations

1. General Form

A system of nonlinear equations can be written as:

$$\begin{aligned} f_1(x_1, x_2, \dots, x_n) &= 0 \\ f_2(x_1, x_2, \dots, x_n) &= 0 \\ &\vdots \\ f_m(x_1, x_2, \dots, x_n) &= 0 \end{aligned}$$

Where:

- $f_1, f_2, \dots, f_m$: Nonlinear functions.
- $x_1, x_2, \dots, x_n$: Variables to be determined.

2. Example of a Nonlinear System

$$x^2 + y^2 = 1 \quad (\text{a circle})$$

$$e^x + y = 2 \quad (\text{an exponential function})$$

3. Methods to Solve Nonlinear Systems

1. Graphical Method (For 2D systems)
2. Newton-Raphson Method (Iterative numerical method)
3. Fixed-Point Iteration
4. fsolve (Numerical Solver in MATLAB)

4. Example Output

For the system:

$$x^2 + y^2 = 1$$

$$e^x + y = 2$$

- **Solution (Approximate):**

$$x = 0.6191, \quad y = 1.4596$$

- **Graph:**

The plot shows the intersection of the circle and the exponential curve, with the intersection point marked.

Code:

```
% Solving a System of Nonlinear Equations
% Author: [Dr.Sreenadh Chevula]
% Date: [07/01/2025]

clc;
clear;
close all;

%% Define the System of Nonlinear Equations
% The system is defined in a function handle
% Example System:
% f1 = x^2 + y^2 - 1;
% f2 = exp(x) + y - 2;

fun = @(vars) [vars(1)^2 + vars(2)^2 - 1;
               exp(vars(1)) + vars(2) - 2];

%% Initial Guess
x0 = [0.5; 0.5]; % Initial guess for [x; y]
```

```

%% Solve the System Using fsolve
options = optimoptions('fsolve', 'Display', 'iter');
[solution, fval, exitflag] = fsolve(fun, x0, options);

%% Display Results
if exitflag > 0
    fprintf('Solution found:\n');
    fprintf('x = %.4f\n', solution(1));
    fprintf('y = %.4f\n', solution(2));
else
    disp('fsolve did not converge to a solution.');
```

end

```

%% Verify the Solution
disp('Function values at the solution (should be close to zero):');
disp(fval);

%% Plot the Equations
fimplicit(@(x, y) x.^2 + y.^2 - 1, [-2, 2, -2, 2], 'r', 'LineWidth', 1.5);
hold on;
fimplicit(@(x, y) exp(x) + y - 2, [-2, 2, -2, 2], 'b', 'LineWidth', 1.5);
plot(solution(1), solution(2), 'ko', 'MarkerSize', 10, 'MarkerFaceColor', 'k');
xlabel('x');
ylabel('y');
title('Intersection of Nonlinear Equations');
legend('x^2 + y^2 - 1', 'exp(x) + y - 2', 'Solution Point');
grid on;
hold off;
```

OUT PUT:

Iteration	Func-count	f(x)	Norm of step	First-order optimality	Trust-region radius
0	3	0.272118		0.351	1
1	4	0.272118	1	0.351	1
2	7	0.147026	0.25	0.178	0.25
3	10	0.136603	0.625	0.932	0.625
4	13	0.000521958	0.151144	0.0465	0.625
5	16	6.3995e-08	0.0156765	0.000552	0.625
6	19	3.66851e-16	0.000138007	4.01e-08	0.625

Equation solved.

fsolve completed because the vector of function values is near zero

as measured by the value of the function tolerance, and

the problem appears regular as measured by the gradient.

<stopping criteria details>

Solution found:

x = -0.0000

y = 1.0000

Function values at the solution (should be close to zero): 1.0e-07 * 0.1905 0.0198

9. Graphics kinematics of particle position, velocity, and acceleration

OBJECTIVE: -

i) Graphics kinematics of particle position, velocity and acceleration

SOFTWARE USED: - MATLAB 2022a or 2019a

Theory:

The basic concepts of kinematics involve understanding the relationship between **position**, **velocity**, and **acceleration** of a particle moving along a path. These quantities can be defined as:

- **Position (x(t)):** The location of the particle at a specific time t.
- **Velocity (v(t)):** The rate of change of position with respect to time, i.e., the derivative of position.
- **Acceleration (a(t)):** The rate of change of velocity with respect to time, i.e., the derivative of velocity.

In mathematical terms:

$$\text{Velocity } v(t) = \frac{dx(t)}{dt}$$

$$\text{Acceleration } a(t) = \frac{dv(t)}{dt} = \frac{d^2x(t)}{dt^2}$$

an example **MATLAB code** to compute and plot the position, velocity, and acceleration of a particle. We'll assume the particle's position follows a function of time like $x(t)=t^3-5t^2+2t+1$

PROCEDURE: -

- Open MATLAB
- Open new M-file
- Type the program.
- Save in current directory.
- Compile and run the program.
- For the output see command window\ Figure window

Code :

```

% Kinematics of a Particle: Position, Velocity, and Acceleration
% Author: [Dr. Sreenadh Chevula]
% Date: [07/01/2025]

clc;
clear;
close all;

%% Define the time vector (from t = 0 to t = 10 seconds)
t = linspace(0, 10, 1000); % Time range

%% Define the position function x(t) = t^3 - 5t^2 + 2t + 1
x = @(t) t.^3 - 5*t.^2 + 2*t + 1;

%% Compute the velocity and acceleration by differentiating position function
v = @(t) 3*t.^2 - 10*t + 2; % First derivative of x(t)
a = @(t) 6*t - 10;          % First derivative of v(t), or second derivative of x(t)

%% Calculate position, velocity, and acceleration over time
position = x(t);
velocity = v(t);
acceleration = a(t);

%% Plot Position, Velocity, and Acceleration
figure;
subplot(3, 1, 1);
plot(t, position, 'LineWidth', 2, 'Color', 'b');
title('Position vs Time');
xlabel('Time (s)');
ylabel('Position (m)');
grid on;

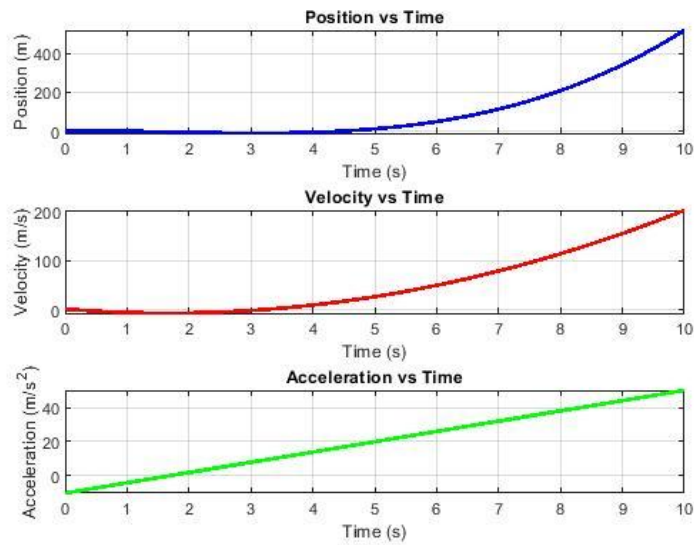
subplot(3, 1, 2);
plot(t, velocity, 'LineWidth', 2, 'Color', 'r');
title('Velocity vs Time');
xlabel('Time (s)');
ylabel('Velocity (m/s)');
grid on;

subplot(3, 1, 3);
plot(t, acceleration, 'LineWidth', 2, 'Color', 'g');
title('Acceleration vs Time');
xlabel('Time (s)');
ylabel('Acceleration (m/s^2)');
grid on;

%% Display values at a specific time (for example t = 5 seconds)
t_val = 5;
disp(['At t = ', num2str(t_val), ' seconds:']);
disp(['Position: ', num2str(x(t_val)), ' meters']);
disp(['Velocity: ', num2str(v(t_val)), ' m/s']);
disp(['Acceleration: ', num2str(a(t_val)), ' m/s^2']);

```

output:



At $t = 5$ seconds:

Position: 11 meters

Velocity: 27 m/s

Acceleration: 20 m/s²

RELATED QUESTIONS: -

1. Explain about particle position velocity, and acceleration?
2. Explain about the functions of subs()?
3. Explain about the functions of legend () ?

10. Develop the graphical user interface to identify the area moment of inertia of simple section trapezoidal and triangle

OBJECTIVE: -

i) Develop the graphical user interface to identify the area moment of inertia of simple section trapezoidal and triangle

SOFTWARE USED: - MATLAB 2022a or 2019a

- Open MATLAB
- Open new M-file
- Use the command GUID to open the GUI controller of the MATLAB.
- Save in current directory.
- Create the required text boxes, push buttons, static text boxes, radio buttons etc
- Save the program with a proper name.
- Code the push buttons with required coding
- Compile and run the program.
- For the output see command window\ Figure window\ gui window

Theory :

Graphical user interfaces (GUIs), also known as apps, provide point-and-click control of your software applications, eliminating the need for others to learn a language or type commands in order to run the application. You can share apps both for use within MATLAB and also as standalone desktop or web apps.

You can choose from the following three ways to create an app in MATLAB:

- Convert a script into a simple app: Choose this option when you want to share a script with students or colleagues and allow them to modify variables using interactive controls.
- Create an app interactively: Choose this option when you want to create a more sophisticated app using a drag-and-drop environment to build the user interface.
- Create an app programmatically: Choose this option when you want to create an app's user interface by writing the code yourself.

Convert a Script into a Simple App

Use the Live Editor to convert a script into a simple app that has interactive controls allowing others to experiment with variables in your code. Add sliders, dropdowns, edit fields, and buttons without writing any code. Specify what parts of the script will run when a value is changed. Hide the code to create simple apps and dashboards.

Create an App Interactively

App Designer is an interactive environment that integrates the two primary tasks of app building: laying out the visual components and programming the app's behaviour. It allows you to quickly move between visual design in the canvas and developing code in the MATLAB editor.

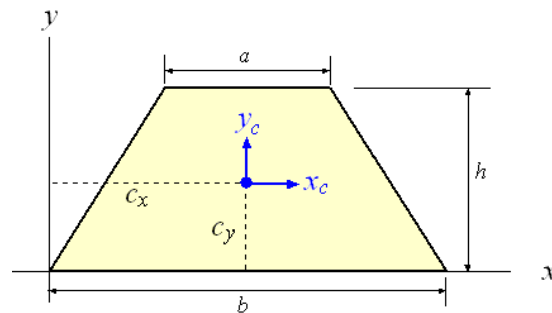
You can share your app with others to use in MATLAB on the desktop or in a web browser using MATLAB Online. App Designer apps can also be packaged for installation into the MATLAB Apps tab. To share with non-MATLAB users, you can compile apps into standalone desktop and web apps using MATLAB Compiler.

App Designer is good for interactively designing your layout and programming its behaviours in one environment. If you prefer, you can program the entire app yourself, including the user interface.

Create an App Programmatically

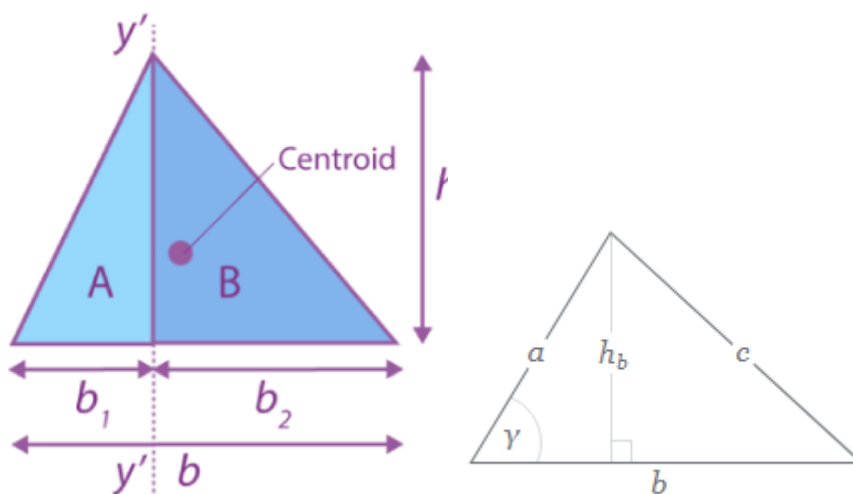
For added control over design and development, you can also use MATLAB functions to define the layout and behaviours of your app. In this approach, you create a figure to serve as the container for your user interface and add components to it programmatically.

The **area and moment of inertia** of simple section trapezoidal



C_x	C_y	Area
$\frac{b}{2}$	$\frac{h}{3} \left(\frac{2a+b}{a+b} \right)$	$\frac{h}{2} (a+b)$

Moment of Inertia about the x axis	Moment of Inertia about the y axis	Polar Moment of Inertia about the z axis
I_x	I_y	J_z
$\frac{h^3(3a+b)}{12}$	$\frac{h(a+b)(a^2+7b^2)}{48}$	$\frac{h(4h^2b+12h^2a+7b^3+7ab^2+ba^2+a^3)}{48}$



The moment of inertia and area of a triangle

Case-1 *axis passing through the centroid and parallel to its base :*

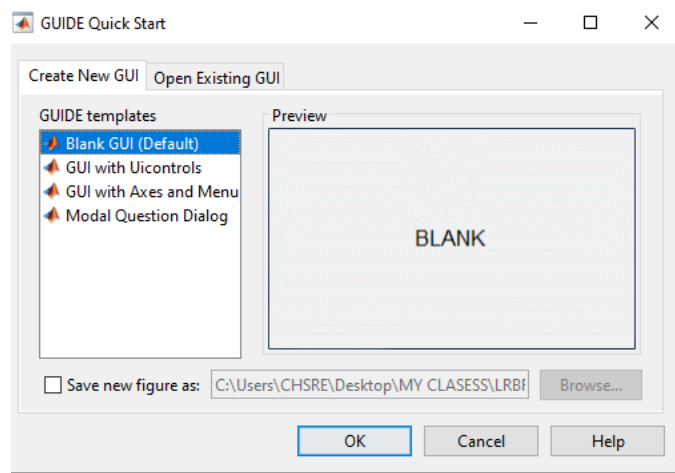
$$I = \frac{bh^3}{36} \text{ Where, } b = \text{base width } h = \text{height}$$

Case- 2. *Axis passing through the base* $I = \frac{bh^3}{12}$

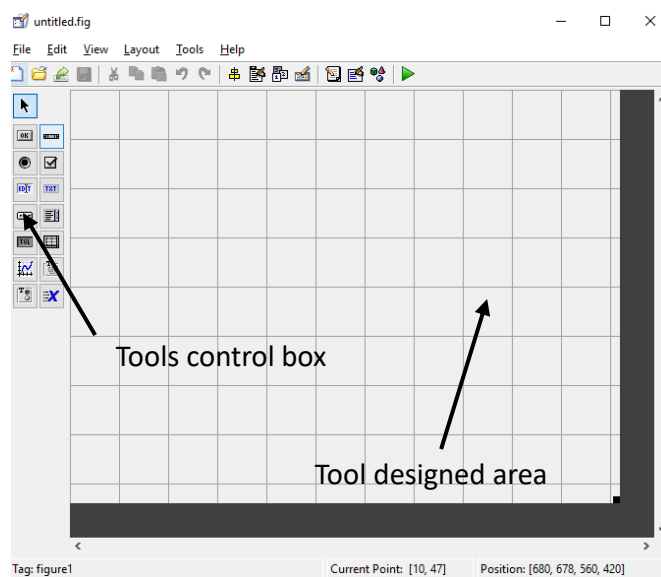
$$\text{Area } A = \frac{h_b b}{2}$$

Steps to be followed:

1. In the command window type “guide” then u will get this following window

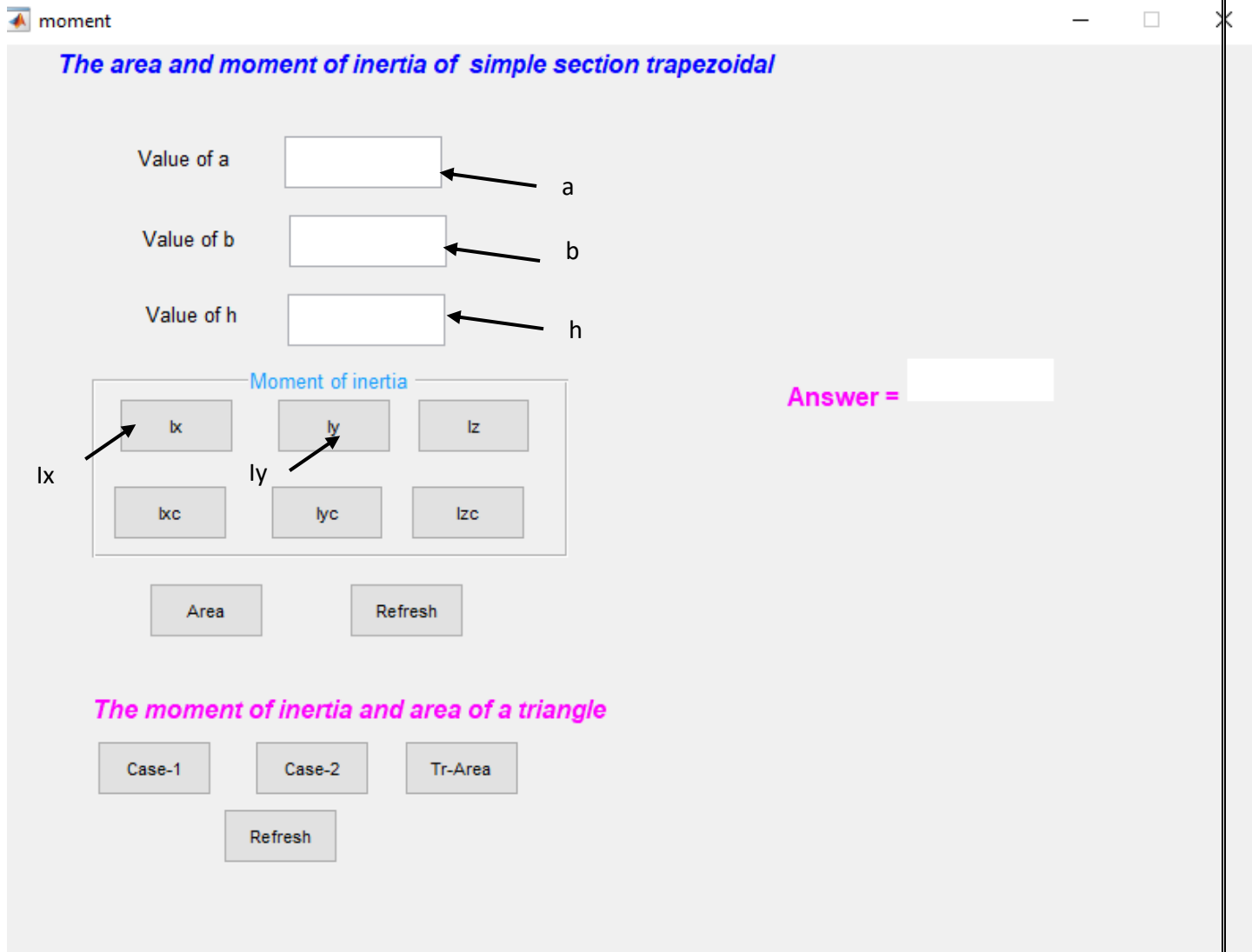


2. Select Blank GUI (default) by clicking ok button. Then bellow window will be appear.

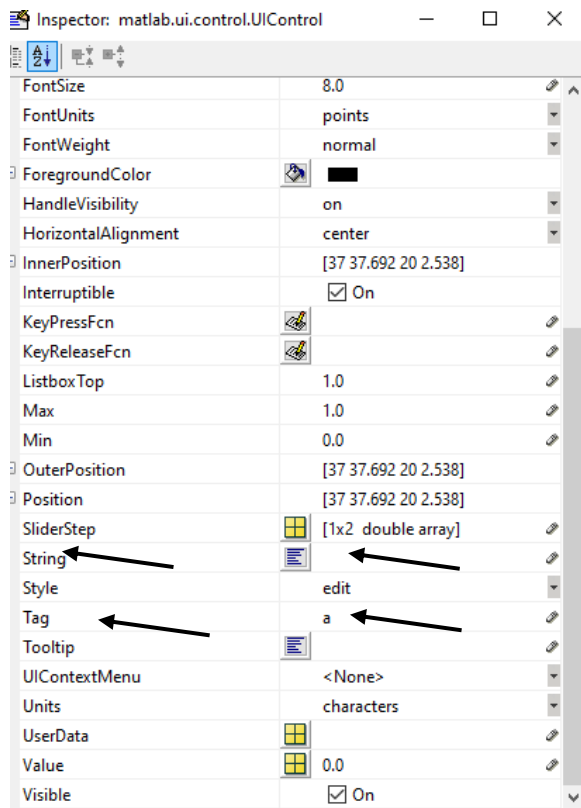


The App Designer interface

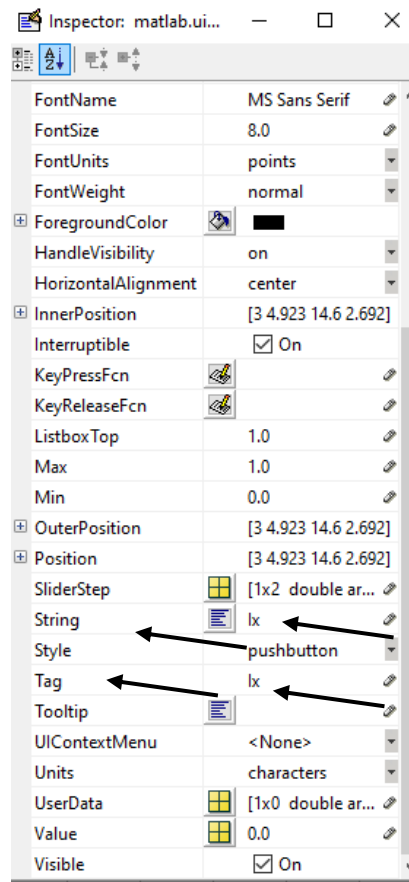
3. Create the App Designer interface like bellow for our program



3. Change the each edit, push button name and tags as they required from the properties window just like shown bellow (example shown for first edit button)



For edit box



For pushbutton

4. Save the GUI interface with the proper name in this case I used “moment”
5. Once saving done automatic code will be created by the MATLAB coding area like bellow.

```
function varargout = addi(varargin)
% ADDI MATLAB code for addi.fig
%
%   ADDI, by itself, creates a new ADDI or raises the existing
%   singleton*.
%
%   H = ADDI returns the handle to a new ADDI or the handle to
%   the existing singleton*.
%
%   ADDI('CALLBACK',hObject,eventData,handles,...) calls the local
%   function named CALLBACK in ADDI.M with the given input arguments.
%
%   ADDI('Property','Value',...) creates a new ADDI or raises the
%   existing singleton*. Starting from the left, property value pairs are
%   applied to the GUI before addi_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property application
%   stop. All inputs are passed to addi_OpeningFcn via varargin.
%
%* See also: GUIDE, GUIDATA, GUIHANDLES
```

6. Now coding part

In the coding area find the followings and make the code as below

```
% --- Executes on button press in Ix.
function Ix_Callback(hObject, eventdata, handles)
% hObject    handle to Ix (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
a=str2num(get(handles.a,'string'));
b=str2num(get(handles.b,'string'));
h=str2num(get(handles.h,'string'));
c=(h^3*(3*a+b))/12;
set(handles.answer,'string',num2str(c));

% --- Executes on button press in Iy.
function Iy_Callback(hObject, eventdata, handles)
% hObject    handle to Iy (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
a=str2num(get(handles.a,'string'));
b=str2num(get(handles.b,'string'));
h=str2num(get(handles.h,'string'));
c=(h*(a+b)*(a^2+7*b^2))/48;
set(handles.answer,'string',num2str(c));

% --- Executes on button press in Iz.
function Iz_Callback(hObject, eventdata, handles)
% hObject    handle to Iz (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
a=str2num(get(handles.a,'string'));
b=str2num(get(handles.b,'string'));
h=str2num(get(handles.h,'string'));
c=(h*((4*h^2*b)+(12*h^2*a)+(7*b^3)+(7*a*b^2)+(b*a^2)+(a^3)))/48;
set(handles.answer,'string',num2str(c));

% --- Executes on button press in Area.
function Area_Callback(hObject, eventdata, handles)
% hObject    handle to Area (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
a=str2num(get(handles.a,'string'));
b=str2num(get(handles.b,'string'));
h=str2num(get(handles.h,'string'));
c=(h/2)*(a+b);
set(handles.answer,'string',num2str(c));

% --- Executes on button press in Ixc.
function Ixc_Callback(hObject, eventdata, handles)
% hObject    handle to Ixc (see GCBO)
```

```
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
a=str2num(get(handles.a,'string'));
b=str2num(get(handles.b,'string'));
h=str2num(get(handles.h,'string'));
c=(h^3*(a^2+(4*a*b)+b^2))/(36*(a+b));
set(handles.answer,'string',num2str(c));
```

```
% --- Executes on button press in Refresh.
```

```
function Refresh_Callback(hObject, eventdata, handles)
% hObject handle to Refresh (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
set(handles.answer,'string','0');
set(handles.a,'string','0');
set(handles.b,'string','0');
set(handles.h,'string','0');
```

```
% --- Executes on button press in Cs1.
```

```
function Cs1_Callback(hObject, eventdata, handles)
% hObject handle to Cs1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
b=str2num(get(handles.b,'string'));
h=str2num(get(handles.h,'string'));
c=(h^3*b)/36
set(handles.answer,'string',num2str(c));
```

```
% --- Executes on button press in Cs2.
```

```
function Cs2_Callback(hObject, eventdata, handles)
% hObject handle to Cs2 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
b=str2num(get(handles.b,'string'));
h=str2num(get(handles.h,'string'));
c=(h^3*b)/12
set(handles.answer,'string',num2str(c));
```

```
% --- Executes on button press in TrAerea.
```

```
function TrAerea_Callback(hObject, eventdata, handles)
% hObject handle to TrAerea (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
b=str2num(get(handles.b,'string'));
h=str2num(get(handles.h,'string'));
c=(h*b)/2
set(handles.answer,'string',num2str(c));
```

```
% --- Executes on button press in Refresh1.
```

```
function Refresh1_Callback(hObject, eventdata, handles)
```

```

% hObject   handle to Refresh1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
set(handles.answer,'string','0');
set(handles.a,'string','0');
set(handles.b,'string','0');
set(handles.h,'string','0');

% --- Executes on button press in Izc.
function Izc_Callback(hObject, eventdata, handles)
% hObject   handle to Izc (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
a=str2num(get(handles.a,'string'));
b=str2num(get(handles.b,'string'));
h=str2num(get(handles.h,'string'));
c=(h*((16*h^2*a*b)+(4*h^2*a^2)+(3*a^4)+(6*a^2*b^2)+(6*a^3*b)+(6*a*b^3)+(3*b^4)))/(144*(a+b));
set(handles.answer,'string',num2str(c));

% --- Executes on button press in Iyc.
function Iyc_Callback(hObject, eventdata, handles)
% hObject   handle to Iyc (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles   structure with handles and user data (see GUIDATA)
a=str2num(get(handles.a,'string'));
b=str2num(get(handles.b,'string'));
h=str2num(get(handles.h,'string'));
c=(h*(a+b)*(a^2+b^2))/48;
set(handles.answer,'string',num2str(c));
Out put:

```

moment

The area and moment of inertia of simple section trapezoidal

Value of a

Value of b

Value of h

Moment of inertia

Answer =

The moment of inertia and area of a triangle

Moment of inertia for a trapezoidal with respect to x axis is I_x

moment

The area and moment of inertia of simple section trapezoidal

Value of a

Value of b

Value of h

Moment of inertia

Answer =

The moment of inertia and area of a triangle

Area for a trapezoidal

RELATED QUESTIONS: -

1. Explain about Moment of inertia for a trapezoidal with various axes of calculations?
2. Explain about Moment of inertia for a triangle with various cases of calculations?
3. Explain about the **handles** in the GUI code?
4. What is the push button and edit button in the Gui interface?
5. How we change the properties of push button, edit button in all other controls in the Gui interface?

11. Identification of shear force and bending moment diagram of cantilever beam with point load

OBJECTIVE: -

i) Identification of shear force and bending moment diagram of cantilever beam with point load

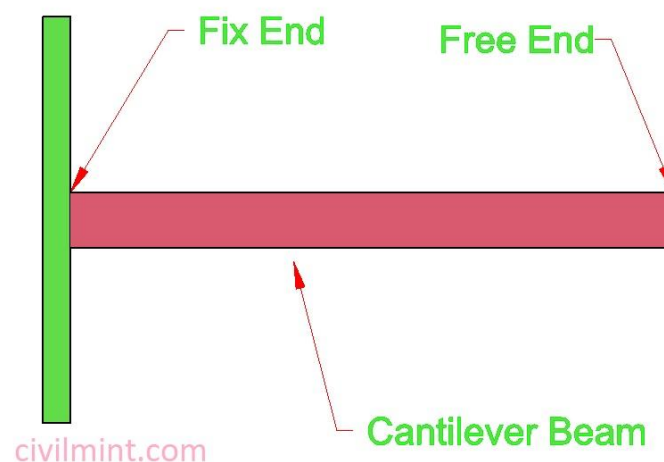
SOFTWARE USED: - MATLAB 2022a or 2019a

PROCEDURE: -

- Open MATLAB
- Open new M-file
- Type the program.
- Save in current directory.
- Compile and run the program.
- For the output see command window\ Figure window

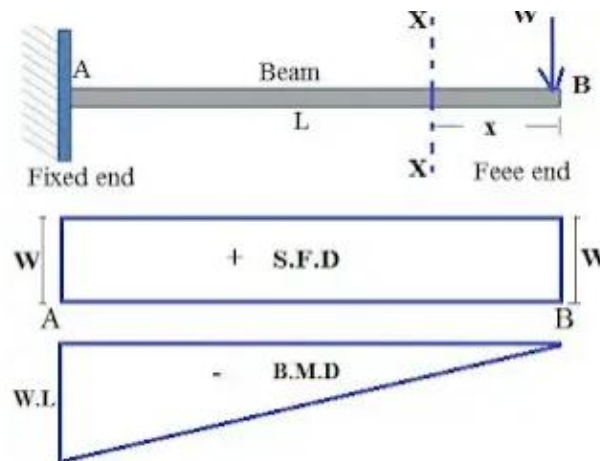
Theory:

A cantilever beam is a structure that is supported on one end and free on the other end. It is subjected to different types of loads and moments that affect its shear force and bending moment.



Shear force and bending moment diagram for cantilever beam with point load

- Let us consider one beam AB of length L as displayed in following figure.
- As we can see here that one end of the beam AB is fixed at one end i.e. at A and other end is free i.e. at end B.
- Therefore, we can say that we have one cantilever beam here and we will have to find the method to draw shear force and bending moment diagram.
- Let us consider that a point load W is acting at free end of the cantilever beam as displayed in following figure.
- Let us consider one section XX at a distance x from free end i.e. from end B.
- Now we will have two portion of the beam AB i.e. left portion and right portion.



- Let us deal with the right portion here; you can also go with left portion of the beam in order to draw shear force and bending moment diagram.

Shear force diagram

- Let us assume that F_X is shear force and M_X is the bending moment at section XX. Shear force at section XX will be equal to the resultant force acting to the right portion of the section.
- We will recall here the sign conventions for shear force and bending moment and we can conclude here that resultant force acting to the right portion of the section will be W and it will be positive.
- Therefore, shear force at section XX will be as mentioned here
 - $F_X = W$
- As we have seen in our post “Basics of shear force and bending moment diagrams” that if there will not be any load between two points then shear force will be constant. The figure shows here shear force diagram.

Bending moment diagram

- Bending moment at section XX will be written as
 - $M_X = -W \cdot x$
- Above equation indicates that bending moment will follow here the linear equation.
- Again, if we will recall sign conventions for shear force and bending moment, we will conclude here that bending moment at section XX will be negative.
- Bending moment will be directionally proportional with the distance x and we can secure here the value of bending moment at point A and at point B as mentioned here.
- Bending moment at free end i.e. at B, value of distance $x = 0$**
 - $M_B = 0$
- Bending moment at fixed end i.e. at A, value of distance $x = L$**
 - $M_A = -W \cdot L$

Code

```
% Simple algorithm to calculate the deflection, bending moment, shear force in a
% cantilever beam.
% load = loading distribution
% Example,
% 1.Uniform loading of 10 N/m: load=repmat(10,1,1001), l=20, dl=0.02

% 2.Point loading with P=400 N, load=[repmat(0,1,999) 10000 10000], l=20, dl=0.02

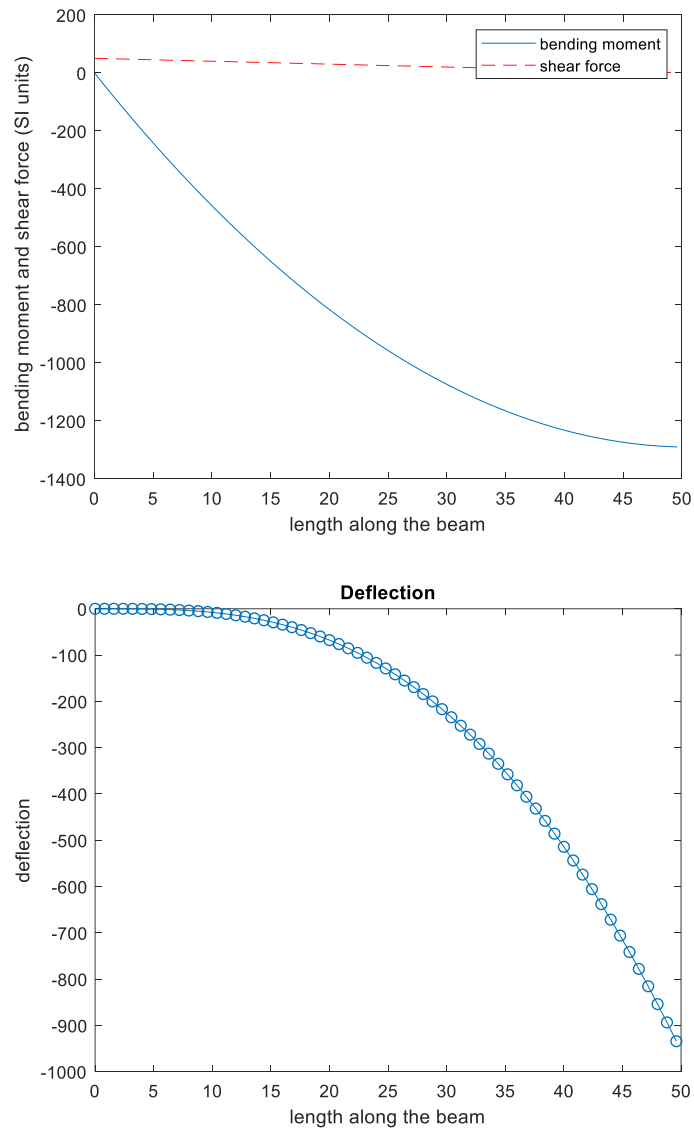
clc
clear all
close all
l=input('Enter length of beam in m:'); % length of beam in m
dl=input('Enter value of X:'); % length of X in m
ei=input('Enter initial value of point load P in N:'); %Point loading

y=0:dl:l;
load=repmat(1,length(y)); %loading distribution
m=sum((y.*load))*dl;
v=sum(load)*dl;
u_4=load/ei;
u_3=v/ei;

for i=2:length(load)
    u_3(i)=u_3(i-1)-u_4(i-1)*dl;
end
u_2=m/ei;
for i=2:length(load)
    u_2(i)=u_2(i-1)-u_3(i-1)*dl;
end
u_1=0;
for i=2:length(load)
    u_1(i)=u_1(i-1)+u_2(i-1)*dl;
end
u=0;
for i=2:length(load)
    u(i)=u(i-1)+u_1(i-1)*dl;
end
deflection=u;
plot(y,u_2*ei)
hold on
plot(y,u_3*ei,'r--')
legend('bending moment','shear force')
xlabel('length along the beam')
ylabel(' bending moment and shear force (SI units)')
hold off
figure,plot(y,u,'o-')
xlabel('length along the beam')
ylabel('deflection')
title('Deflection')
```

out put

```
Enter length of beam in m:50
Enter value of X:0.8
Enter initial value of point load P in N:800
```

**RELATED QUESTIONS: -**

1. What is Shear force diagram?
2. What is Bending moment diagram?
3. Explain about `repmat()` function in MATLAB?
4. How we convert the physical object of the cantilever beam properties in to numerical form to code in MATLAB?

12. MATLAB Graphical user interface: Design and develop a scientific calculator

OBJECTIVE: -

i) MATLAB Graphical user interface: Design and develop a scientific calculator

SOFTWARE USED: - MATLAB 2022a or 2019a

PROCEDURE: -

- Open MATLAB
- Open new M-file
- Type the program.
- Save in current directory.
- Compile and run the program.
- For the output see command window\ Figure window

Theory:

Design and Develop a Scientific Calculator Using MATLAB GUI

MATLAB provides the ability to create graphical user interfaces (GUIs) for custom applications using the **App Designer** or by manually creating a GUI with `uifigure`, `uicontrol`, and callback functions.

Below, I'll guide you through the process of designing a **scientific calculator** with essential functionalities like addition, subtraction, multiplication, division, trigonometric functions, square roots, and exponentiation. We'll use **MATLAB's App Designer** to make the process easy, but I'll also provide a solution using `uifigure` and `uicontrol` for a manual approach.

Steps for Designing a Scientific Calculator in MATLAB GUI

1. Create the GUI Layout

- Define the input field where the user will enter the expression.
- Create buttons for digits, operators, and scientific functions like sine, cosine, log, etc.
- Add a button for the "Calculate" functionality.

2. Define Callback Functions

- Implement callback functions that handle the button presses, including handling input and performing calculations.

3. Display Results

- Show the result in the input field.

MATLAB GUI Code for a Scientific Calculator

This code demonstrates how to manually create a simple scientific calculator using `uifigure` and `uicontrol`. The calculator will include basic operations and scientific functions like `sin`, `cos`, `tan`, etc.

Code:

```
function guiScientificCalculator()
clc
clear all
close all
% Create figure window
fig = figure('Name', 'Scientific Calculator', 'NumberTitle', 'off', ...
'Position', [200, 200, 400, 500], ...
'MenuBar', 'none', 'ToolBar', 'none');
% Create display edit field
displayField = uicontrol('Style', 'edit', 'FontSize', 16, ...
'HorizontalAlignment', 'right', ...
'Position', [10, 430, 380, 50]);
% Create buttons
buttons = {'7', '8', '9', '/', 'sin';
'4', '5', '6', '*', 'cos';
'1', '2', '3', '-', 'tan';
'0', '.', '=', '+', 'sqrt';
'(', ')', 'AC', '^', 'log';
'pi', 'e', 'x^2', '1/x', 'exp'};
buttonWidth = 75;
buttonHeight = 50;
startX = 10;
startY = 360;
for row = 1:size(buttons, 1)
for col = 1:size(buttons, 2)
btn = uicontrol('Style', 'pushbutton', 'String', buttons{row, col}, ...
'FontSize', 12, 'Position', [startX + (col-1)*buttonWidth, ...
startY - (row-1)*buttonHeight, buttonWidth, buttonHeight]);
set(btn, 'Callback', {@buttonPressed, buttons{row, col}});
end
end
% Set up initial state
currentState = '';
set(displayField, 'String', currentState);
% Button press callback function
function buttonPressed(~, ~, key)
switch key
case 'pi'
currentState = [currentState 'pi'];
case 'e'
currentState = [currentState 'exp(1)'];
case 'sin'
currentState = [currentState 'sin('];
case 'cos'
currentState = [currentState 'cos('];
case 'tan'
currentState = [currentState 'tan('];
case 'sqrt'
currentState = [currentState 'sqrt('];
case 'log'
currentState = [currentState 'log('];
case '^'

```

```

currentState = [currentState '^'];
case 'x^2'
currentState = [currentState '^2'];
case '1/x'
currentState = [currentState '1/'];
case 'AC'
currentState = '';
case '='
try
result = eval(currentState);
currentState = num2str(result);
catch
currentState = 'Error';
end
otherwise
currentState = [currentState key];
end
set(displayField, 'String', currentState);
end
end

```

output:

