

EXPERIMENT - 1

Aim : An Introduction about MATLAB.

Description : MATLAB stands for Matrix Laboratory. It is a high-performance language that is used for technical computing. It was developed by Cleve Moler of the company MathWorks.Inc in the year 1984.It is written in C, C++, Java. It allows matrix manipulations, plotting of functions, implementation of algorithms and creation of user interfaces.

Getting Started with MATLAB

It is both a programming language as well as a programming environment. It allows the computation of statements in the command window itself.

- **Command Window:** In this window one must type and immediately execute the statements, as it requires quick prototyping. These statements cannot be saved. Thus, this is can be used for small, easily executable programs.
- **Editor (Script):** In this window one can execute larger programs with multiple statements, and complex functions These can be saved and are done with the file extension ‘.m ‘
- **Workspace:** In this window the values of the variables that are created in the course of the program (in the editor) are displayed.
- **Command History window :** This window displays a log of statements that you ran in the current and previous MATLAB sessions. The Command History lists the time and date of each session in the short date format for your operating system, followed by the statements from that session.
- This window displays the exact location(path) of the program file being created.

Writing a MATLAB Program :-

1. **Using Command Window:** Only one statement can be typed and executed at a time. It executes the statement when the enter key is pressed. This is mostly used for simple calculations. **Note:** ans is a default variable created by MATLAB that stores the output of the given computation.
2. **Using Editor:** Multiple lines of code can be written here and only after pressing the run button (or F5) will the code be executed. It is always a good practice to write clc, clear and close all in the beginning of the program.**Note:** Statements ending with a semicolon will not be displayed in the command window, however, their values will be displayed in the workspace. Any statement followed by % in MATLAB is considered as a comment
3. **Vector Operations:** Operations such as addition, subtraction, multiplication and division can be done using a single command instead of multiple loops

Here's a breakdown of its key features :

1. Mathematics and Computation: MATLAB offers a rich set of mathematical functions for linear algebra, statistics, optimization, and differential equations. It allows users to perform complex calculations efficiently, making it a popular choice for engineers, scientists, and researchers.

2. Data Analysis and Visualization: With powerful plotting and visualization tools, MATLAB enables users to explore and analyze data in various formats, including arrays, tables, and images. It supports customizable plots, 2D and 3D visualizations, as well as interactive graphics for conveying insights from data.

3. Algorithm Development: MATLAB facilitates algorithm prototyping and development with its intuitive syntax and extensive library of pre-built functions. Users can implement and test algorithms quickly, making it ideal for signal processing, image processing, machine learning, and control system design.

4. Simulation and Modeling: MATLAB includes simulation capabilities for simulating dynamic systems, such as mechanical, electrical, and biochemical systems. It supports simulation of continuous-time and discrete-time models, enabling engineers to design and analyze complex systems before implementation.

5. Application Development: MATLAB can be used for developing standalone applications and graphical user interfaces (GUIs) for deploying computational tools to end-users. It provides tools for building interactive apps with buttons, sliders, and plots, allowing developers to create user-friendly interfaces for their MATLAB programs.

6. Parallel Computing and Deployment: MATLAB supports parallel computing to accelerate computationally intensive tasks by distributing computations across multiple processors or computing nodes. Additionally, MATLAB Compiler allows users to package MATLAB code into standalone executables or shared libraries for deployment to systems without MATLAB installed.

7. Education and Research: MATLAB is widely used in academia for teaching and research purposes across various disciplines, including engineering, physics, biology, and finance. Its easy-to-learn syntax and extensive documentation make it accessible to students and researchers alike.

Overall, MATLAB's versatility and broad range of functionalities make it a valuable tool for professionals and researchers in numerous fields, from engineering and science to finance and beyond. Its interactive nature, combined with its extensive library of functions and toolboxes, makes it a preferred choice for tackling diverse computational and data analysis challenges.

Using MATLAB offers several advantages across various fields and applications:

1. Easy to Learn and Use: MATLAB's syntax is designed to be intuitive and easy to learn, especially for those with a background in mathematics or engineering. Its interactive environment allows users to quickly test ideas and algorithms without needing to compile code.

2. Extensive Functionality: MATLAB provides a vast array of built-in functions and toolboxes for numerical computation, data analysis, signal processing, image processing, optimization, machine learning, and more. This extensive functionality reduces the need for writing code from scratch and accelerates development time.

3. Rich Visualization Tools: MATLAB offers powerful plotting and visualization capabilities, allowing users to create high-quality 2D and 3D plots, animations, and interactive graphics. Visualization is crucial for analyzing data and conveying insights effectively.

4. Interdisciplinary Applications: MATLAB is widely used across various disciplines, including engineering, science, finance, economics, and biology. Its versatility makes it a valuable tool for researchers, educators, and professionals working in diverse fields.

5. Simulation and Modeling: MATLAB supports simulation and modeling of dynamic systems, enabling engineers to design, analyze, and optimize complex systems before implementation. Its simulation capabilities are crucial for understanding system behavior and making informed decisions.

6. Algorithm Prototyping: MATLAB's interactive environment is well-suited for algorithm prototyping and development. Users can quickly implement and test algorithms, iterate on designs, and refine solutions before deploying them in production environments.

7. Community and Support: MATLAB has a large and active user community, which provides access to resources, forums, and online tutorials. Additionally, MathWorks offers comprehensive documentation, technical support, and training programs to help users get the most out of the software.

8. Integration with Other Tools: MATLAB can easily integrate with other programming languages, tools, and hardware platforms. It supports interoperability with C/C++, Python, Java, and other languages, as well as interfaces to hardware devices and external software packages.

9. Parallel Computing and Deployment: MATLAB supports parallel computing to leverage multi-core processors, clusters, and cloud computing resources for accelerating computationally intensive tasks. Additionally, MATLAB Compiler enables users to deploy MATLAB code as standalone applications or shared libraries for distribution to end-users.

10. Education and Research: MATLAB is widely used in academia for teaching and research purposes due to its accessibility, versatility, and broad range of functionalities. Its adoption in educational institutions prepares students for careers in industry and research.

EXPERIMENT - 2

Aim : Find addition, subtraction, multiplication and division of two matrices using MATLAB.

Description :-

Defining a Matrix:

In MATLAB, a matrix is a rectangular array of numbers, symbols, or expressions arranged in rows and columns. It's a fundamental data structure used for storing and manipulating numerical data.

Matlab

```
A = [1 2 3; 4 5 6; 7 8 9]; % Creates a 3x3 matrix
```

MATLAB provides a powerful and versatile environment for performing various mathematical operations. Among these, fundamental arithmetic operations like addition, subtraction, multiplication, and division form the building blocks for more complex calculations.

- **Addition (+):** This operator combines two numerical values or matrices of the same size, element-wise. For example, $2 + 3$ returns 5 , and $[1\ 2; 3\ 4] + [5\ 6; 7\ 8]$ results in $[6\ 8; 10\ 12]$.
- **Subtraction (-):** This operator finds the difference between two numbers or subtracts corresponding elements from matrices of the same dimensions. For example, $5 - 2$ returns 3 , and $[10\ 20; 30\ 40] - [5\ 10; 15\ 20]$ gives $[5\ 10; 15\ 20]$.
- **Multiplication (*):** Multiplication in MATLAB can have different meanings depending on the context.
 - For scalars (numbers), it performs standard multiplication. For example, $4 * 5$ returns 20 .
 - For matrices, element-wise multiplication is performed if the matrices have the same dimensions. For example, $[1\ 2; 3\ 4] * [5\ 6; 7\ 8]$ results in $[5\ 12; 21\ 32]$.
 - To perform matrix multiplication, use the $*$ operator after defining the matrices using square brackets $[]$. For example, multiplying a matrix **A** with dimensions (m x n) by a matrix **B** with dimensions (n x p) results in a new matrix **C** with dimensions (m x p): $C = A * B$.
- **Division (/):** Division in MATLAB can also have different interpretations.
 - For scalars, it performs standard division. For example, $10 / 2$ returns 5 .
 - For matrices, element-wise division is done if the matrices are of the same size. For example, $[6\ 12; 18\ 24] / [2\ 3; 3\ 4]$ results in $[3\ 4; 6\ 6]$.
 - For left division (\backslash) and right division ($/$), use these operators when working with matrices. Left division solves a system of linear equations ($Ax = b$) where **A** is a matrix, **x** is the unknown vector, and **b** is the right-hand side vector. Right division is the inverse of left division ($b = A \backslash x$).

Implementation :-

```
MatlabExp.m × +
/MATLAB Drive/MatlabExp.m

1  A = [1 2 3; 4 5 6];
2  B = [7 8 9; 10 11 12];
3  % Addition C = A + B;
4  C = A + B;
5  disp('Sum of A and B:')
6  disp(C)
7  % Subtraction D = A - B;
8  D = A - B;
9  disp('Difference of A and B:')
10 disp(D)
11 % Multiplication (assuming A and B have compatible dimensions for multiplication)
12 E = A .* B; % Matrix multiplication
13 disp('Product of A and B:')
14 disp(E)
15 % Element-wise division
16 F = A ./ B;
17 disp('Element-wise division of A and B:')
18 disp(F)
```

Command Window

>> MatlabExp

Sum of A and B:

8	10	12
14	16	18

Difference of A and B:

-6	-6	-6
-6	-6	-6

Product of A and B:

7	16	27
40	55	72

Element-wise division of A and B:

0.1429	0.2500	0.3333
0.4000	0.4545	0.5000

EXPERIMENT - 3

Aim : Verify whether the given matrix is singular or non singular and compute the inverse if applicable

Description :-

- **What is a Singular Matrix?**

A square matrix is said to be a **singular matrix** if its determinant is zero and it is not invertible. In a singular matrix, some rows and columns are linearly dependent. As the rows and columns of a singular matrix are linearly dependent, the rank of the matrix will be less than the order of the matrix.

For example a square matrix $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, the condition of it being a singular matrix is the determinant of this matrix A is a zero value. $|A| = |ad - bc| = 0$.

- **What Is a Non-Singular Matrix?**

A non-singular matrix is a square matrix whose determinant is not equal to zero. The non-singular matrix is an invertible matrix, and its inverse can be computed

as it has a determinant value. For example square matrix $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, the condition of it being a non singular matrix is the determinant of this matrix A is a non zero value. $|A| = |ad - bc| \neq 0$.

- **Inverse of Matrix**

Just like a number has its reciprocal, even a matrix has an inverse. If we consider a matrix A, we denote its inverse as A^{-1} . The inverse of a matrix is another matrix that, when multiplied by the given matrix, yields the multiplicative identity.

For a matrix A, its inverse is A^{-1} . And $A \cdot A^{-1} = I$, where I is denoted as the identity matrix. In order to find the inverse matrix, the square matrix must be non-singular and have a determinant value that is not zero. Let us consider a 2×2 square matrix A.

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

The determinant of matrix A is denoted as $ad - bc$, and the value of the determinant should not be zero in order for the inverse matrix of A to exist. A simple formula can be used to calculate the inverse of a 2×2

matrix. Furthermore, in order to obtain the inverse of a 3×3 matrix, we must first determine the determinant and adjoint of the matrix.

Implementation:

```
MatlabExp.m × +
/MATLAB Drive/MatlabExp.m

1 % Define the matrix
2 matrix = [5, 2; 7, 4];
3
4 % Compute the determinant
5 det_matrix = det(matrix);
6
7 if det_matrix ~= 0
8     % Matrix is non-singular, compute the inverse
9     inverse_matrix = inv(matrix);
10    disp('Matrix is non-singular. ');
11    disp('Inverse Matrix: ');
12    disp(inverse_matrix);
13 else
14    disp('Matrix is singular. ');
15 end

Command Window

>> MatlabExp
Matrix is non-singular.
Inverse Matrix:
    0.6667   -0.3333
   -1.1667    0.8333
```

```
MatlabExp.m × +
/MATLAB Drive/MatlabExp.m

1 % Define the matrix
2 matrix = [4, 4; 4, 4];
3
4 % Compute the determinant
5 det_matrix = det(matrix);
6
7 if det_matrix ~= 0
8     % Matrix is non-singular, compute the inverse
9     inverse_matrix = inv(matrix);
10    disp('Matrix is non-singular. ');
11    disp('Inverse Matrix: ');
12    disp(inverse_matrix);
13 else
14    disp('Matrix is singular. ');
15 end

Command Window

>> MatlabExp
Matrix is singular.
>>
```

Experiment-4

Aim: Sorting of 1-D and 2-D array and searching in an array/matrix. Also, list the set of numbers that obey a common condition in an array/matrix using find().

Description:

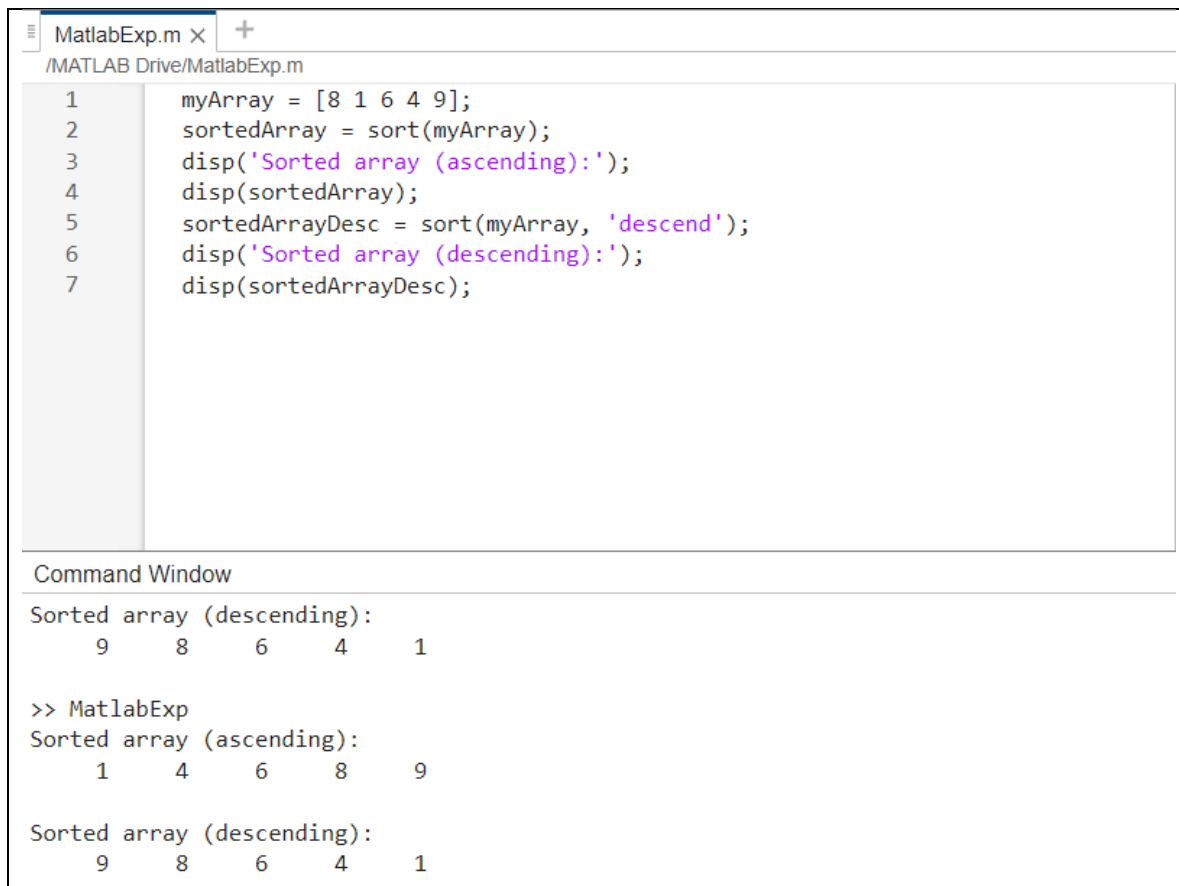
Sorting Arrays: sorting rearranges array elements in a specific order:

1. Ascending (default): Smaller values come first.
2. Descending: Larger values come first.

- **1-D Arrays:**

Sort(myArray) sorts myArray in ascending order. Sort(myArray, 'descend') sorts in descending order.

Example- (1-D array):



The screenshot shows the MATLAB environment with a script editor and a command window. The script editor displays the following code:

```
1 myArray = [8 1 6 4 9];
2 sortedArray = sort(myArray);
3 disp('Sorted array (ascending):');
4 disp(sortedArray);
5 sortedArrayDesc = sort(myArray, 'descend');
6 disp('Sorted array (descending):');
7 disp(sortedArrayDesc);
```

The command window shows the output of the script:

```
Sorted array (descending):
     9     8     6     4     1

>> MatlabExp
Sorted array (ascending):
     1     4     6     8     9

Sorted array (descending):
     9     8     6     4     1
```

- **2-D Arrays (Matrices):**

Sort(matrix) sorts columns (default) in ascending order.

Sort(matrix, 2) sorts rows in ascending order (specify dimension 2).

Add 'descend' for descending order (columns or rows).

Example-(2-D Array):

- **Sorting by columns:**


```
MatlabExp.m x +
/MATLAB Drive/MatlabExp.m
1 % Create a 2-D array
2 matrix = [
3 3 1 4;
4 7 2 5;
5 6 8 9;
6 ];
7 % Sort each column in ascending order
8 sortedMatrixCol = sort(matrix);
9 disp('Sorted matrix by columns (ascending):');
10 disp(sortedMatrixCol);
11 % Sort each column in descending order
12 sortedMatrixColDesc = sort(matrix, 'descend');
13 disp('Sorted matrix by columns (descending):');
14 disp(sortedMatrixColDesc);
```

Command Window

```
>> MatlabExp
Sorted matrix by columns (ascending):
     3     1     4
     6     2     5
     7     8     9

Sorted matrix by columns (descending):
     7     8     9
     6     2     5
     3     1     4
```

- **Sorting by Rows:**

```
MatlabExp.m x +
/MATLAB Drive/MatlabExp.m
1 % Sort each row in ascending order
2 sortedMatrixRow = sort(matrix, 2);
3 disp('Sorted matrix by rows (ascending):');
4 disp(sortedMatrixRow);
5 % Sort each row in descending order
6 sortedMatrixRowDesc = sort(matrix, 2, 'descend');
7 disp('Sorted matrix by rows (descending):');
8 disp(sortedMatrixRowDesc);
```

Command Window

```
>> MatlabExp
Sorted matrix by rows (ascending):
     1     3     4
     2     5     7
     6     8     9

Sorted matrix by rows (descending):
     4     3     1
     7     5     2
     9     8     6
```

- **Searching:** searching an array means finding elements that you're interested in.

There are two main ways to **search arrays** in MATLAB:

1. **Indexing:** Use for finding elements at specific locations based on row and column positions.

Implementation:

```
MatlabExp.m * X +
/MATLAB Drive/MatlabExp.m

2   index = find(myArray == 5);
3   disp('Index of 5:');
4   disp(index);
5   indices = find(matrix > 6);
6   disp('Indices of elements greater than 6:');
7   disp(indices); |

Command Window

>> MatlabExp
Sorted array (ascending):
     1     2     5     8     9

Sorted array (descending):
     9     8     5     2     1

>> MatlabExp
Index of 5:
     1
```

2. **find() function:** More powerful for finding elements based on conditions. It returns indices of elements that meet criteria using comparison operators or logical expressions.

Implementation:

```
MatlabExp.m X +
/MATLAB Drive/MatlabExp.m

1   evenElements = find(mod(myArray, 2) == 0);
2   disp('Indices of even elements:');
3   disp(evenElements);
4   conditions = matrix > 5 & matrix <= 8;
5   matchingElements = find(conditions);
6   disp('Indices of elements greater than 5 and less than or equal to 8:');
7   disp(matchingElements);

Command Window

>> MatlabExp
Indices of even elements:|
     2     3
```

Experiment-5

Aim: Solve simultaneous equations (maximum of three) using Cramer's rule. [Simultaneous equations may be obtained by applying KCL or KVL for a circuit and they can be solved for voltages or currents, respectively].

Description:

- Cramer's rule is a method for solving systems of linear equations that works by using determinants. It can be used to solve systems of equations with up to three variables.
- Cramer's rule is a method for solving systems of linear equations with the same number of equations and variables (up to three). Here's how it works:

Cramer's Rule for a 3x3 System

Let's consider a system of three equations with three unknowns:

$$ax + by + cz = d$$

$$ex + fy + gz = h$$

$$ix + jy + kz = t$$

We can express this system in matrix form as:

$$A * X = B$$

where:

- A is the coefficient matrix:
 $A = [[a, b, c], [e, f, g], [i, j, k]]$
- X is the unknowns vector:
 $X = [[x], [y], [z]]$
- B is the constants vector:
 $B = [[d], [h], [t]]$

Cramer's rule expresses each solution variable as the quotient of two determinants:

- The determinant of a modified coefficient matrix where the constant term's column is replaced by the constants vector for that variable.
- The determinant of the original coefficient matrix (A).

Implementation :

```
MatlabExp.m × +
/MATLAB Drive/MatlabExp.m

1  A = [2 1 3; 1 -1 -1; 1 2 1];
2  b = [3; 0; 0];
3  detA = det(A);
4
5  if detA == 0
6      disp('The system has no unique solution (singular).');
7      return;
8  end
9
10 x = det([b A(:,2) A(:,3)]) / detA;
11 y = det([A(:,1) b A(:,3)]) / detA;
12 z = det([A(:,1) A(:,2) b]) / detA;
13 disp(['x = ' num2str(x)]);
14 disp(['y = ' num2str(y)]);
15 disp(['z = ' num2str(z)]);

Command Window

>> MatlabExp
x = 0.33333
y = -0.66667
z = 1
>>
```

- ❖ **Kirchhoff's Current Law (KCL) and Kirchhoff's Voltage Law (KVL)** are fundamental principles for analyzing electrical circuits. They can be used to create a system of equations that describe the relationships between currents and voltages in a circuit.

KCL (Kirchhoff's Current Law)

- States that the algebraic sum of currents entering a junction in a circuit must equal the algebraic sum of currents leaving the junction.
- This law is based on the principle of conservation of charge – current cannot disappear.

KVL (Kirchhoff's Voltage Law)

- States that the algebraic sum of the voltages around a closed loop in a circuit must equal zero.
- This law is based on the principle of conservation of energy – in a DC circuit, energy cannot be created or destroyed.

Solving Simultaneous Equations in MATLAB

1. **Define Variables:** Assign variables to represent unknown currents or voltages in your circuit.
2. **Write KCL/KVL Equations:** Apply KCL or KVL to write equations that relate the variables you defined.
3. **Create a Coefficient Matrix and Solution Vector:** Arrange your equations in a matrix form, with coefficients of variables on one side and constant terms on the other side.
4. **Solve the System:** Use MATLAB's built-in functions like `\` or `linsolve` to solve the system of equations and find the values of your variables.

Here's an example showing how to solve a simple circuit using KCL and MATLAB:

Example Circuit:

Consider a circuit with a voltage source (V) connected to two resistors (R1 and R2) in parallel. We want to find the current through each resistor.

Steps:

1. **Define Variables:**
 - Let I be the current source current
 - Let I1 be the current through R1
 - Let I2 be the current through R2

2. Write KCL Equation:

- At the junction where the current source splits, $I = I_1 + I_2$

3. Solve in MATLAB:

MatlabExp.m * × +

/MATLAB Drive/MatlabExp.m

```
1 R1 = 100;
2 R2 = 200;
3 % Define Voltage Source (assuming a value here)
4 V = 12;
5 % Create a coefficient matrix (A) and solution vector (b)
6 A = [1 1; 1/R1 1/R2];
7 b = [V; 0]; % 0 here represents no current leaving the second node
8 % Solve the system using backslash operator
9 solution = A \ b;
10 % Extract currents from the solution vector
11 I1 = solution(1);
12 I2 = solution(2);
13 % Display results
14 fprintf('Current through R1: %.2f Amps\n', I1);
15 fprintf('Current through R2: %.2f Amps\n', I2);
16
```

Command Window

```
>> MatlabExp
Current through R1: -12.00 Amps
Current through R2: 24.00 Amps
>>
```

EXPERIMENT - 6

Aim : a) Show that $\log_{10}(A*B) = \log_{10}(A) + \log_{10}(B)$ and $\log_{10}(A/B) = \log_{10}(A) - \log_{10}(B)$ b) Plot the voltage across capacitor during charging $V_c = V_0[1 - e^{-(t/RC)}]$

Descriptions :- Plotting the voltage across a capacitor during charging is crucial in understanding its behavior in an RC circuit. The equation $V_c = V_0[1 - e^{-(t/RC)}]$ represents this voltage, where V_c is the voltage across the capacitor at time t , V_0 is the initial voltage, R is the resistance, C is the capacitance, and e is the base of the natural logarithm. As time progresses, the voltage across the capacitor approaches V_0 exponentially, with the time constant RC determining the rate of charging. Plotting V_c against time provides insights into the charging process, illustrating how the capacitor's voltage evolves over time until it reaches its steady-state value of V_0 .

Implementation (a):-

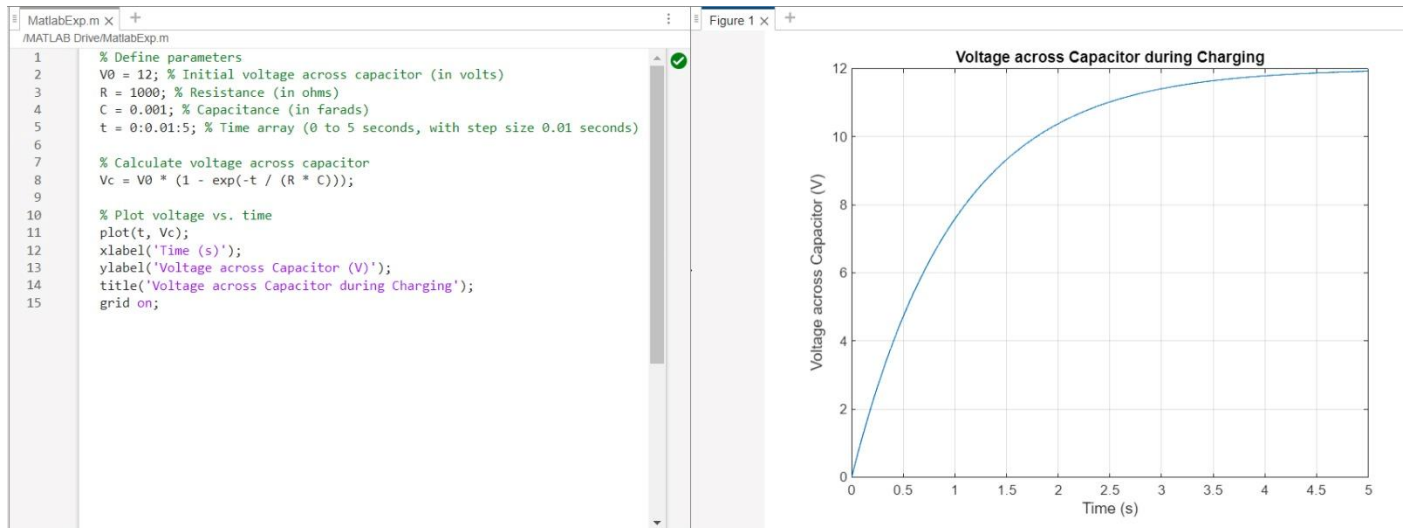
```
MatlabExp.m × +
/MATLAB Drive/MatlabExp.m

1      % Define values for A and B
2      A = 50; % Example value
3      B = 25; % Example value
4      % Calculate log10(A*B)
5      result1 = log10(A * B);
6      % Calculate log10(A) + log10(B)
7      result2 = log10(A) + log10(B);
8      % Compare the results
9      disp(['log10(A*B) = ', num2str(result1)]);
10     disp(['log10(A) + log10(B) = ', num2str(result2)]);
11     % Calculate log10(A/B)
12     result3 = log10(A / B);
13     % Calculate log10(A) - log10(B)
14     result4 = log10(A) - log10(B);
15     % Compare the results
16     disp(['log10(A/B) = ', num2str(result3)]);
17     disp(['log10(A) - log10(B) = ', num2str(result4)]);

Command Window

>> MatlabExp
log10(A*B) = 3.0969
log10(A) + log10(B) = 3.0969
log10(A/B) = 0.30103
log10(A) - log10(B) = 0.30103
>>
```

Implementation (b) :-



EXPERIMENT – 7

Aim : a) Plot a straight line for the given slope and intercept using different plot attributes. b) Differentiate and integrate $y=mx+c$, separately, and display the results on the same plot

a) Plot a straight line for the given slope and intercept using different plot attributes:

In this part, we generate a straight line based on the equation $y=mx+c$, where m is the slope and c is the intercept. We create an array of x values using `linspace()` function, then calculate the corresponding y values for the line using the equation $y=mx+c$.

We use the `plot()` function to plot the line with different plot attributes (colors and line styles) by specifying different format strings ('b', 'r--', 'g-.', 'k:'). These format strings control the color and style of the lines in the plot.

b) Differentiate and integrate $y=mx+c$, separately, and display the results on the same plot:

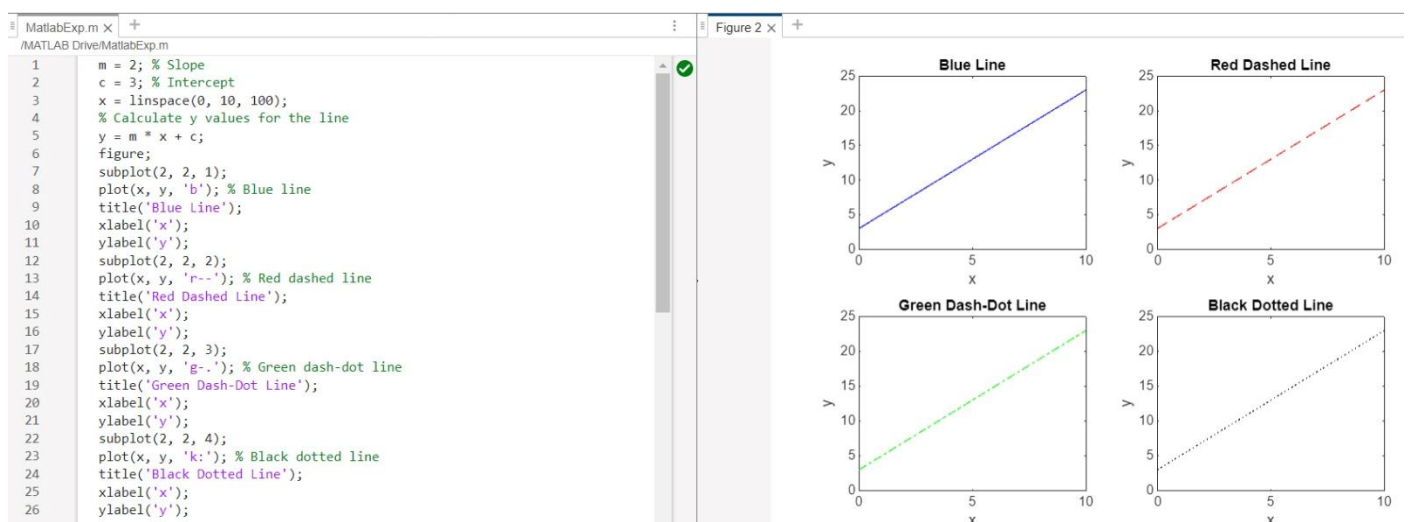
For this part, we first define the symbolic variable x using the `syms` function. Then, we define the function $y=mx+c$ symbolically using the symbolic variables.

We differentiate y with respect to x using the `diff()` function to find the derivative of the function. Similarly, we integrate y with respect to x using the `int()` function to find the integral of the function.

After differentiating and integrating the function symbolically, we convert the resulting symbolic expressions to MATLAB functions using the `matlabFunction()` function. This allows us to evaluate the derivative and integral at different x values.

We generate a range of x values using `linspace()` and then evaluate the original function, its derivative, and its integral at these x values. Finally, we plot all three functions on the same graph using the `plot()` function, and add labels, titles, legends, and grid lines to make the plot clear and informative.

Implementation (a):-

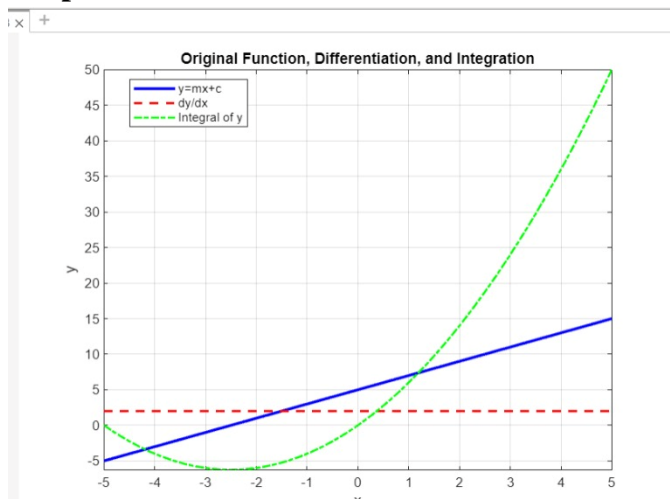


Implementation (b) :-

Code:

```
% Define slope and intercept
m = 2;
c = 5;
x = linspace(0, 10, 100);
% Calculate corresponding y values
y = m*x + c;
% Plotting straight line with different plot attributes
figure;
subplot(2, 2, 1);
plot(x, y, '-'); % Default line style
title('Default Line Style');
subplot(2, 2, 2);
plot(x, y, '--r'); % Dashed red line
title('Dashed Red Line');
subplot(2, 2, 3);
plot(x, y, 'g-'); % Dash-dot green line
title('Dash-dot Green Line');
subplot(2, 2, 4);
plot(x, y, 'm'); % Dotted magenta line
title('Dotted Magenta Line');
% Differentiate and integrate y=mx+c separately
syms x;
dy_dx = diff(m*x + c, x); % Differentiate
int_y = int(m*x + c, x); % Integrate
% Plotting differentiation and integration
figure;
fplot(m*x + c, 'b', 'LineWidth', 2); % Original function
hold on;
fplot(dy_dx, 'r--', 'LineWidth', 1.5); % Differentiated function
fplot(int_y, 'g-', 'LineWidth', 1.5); % Integrated function
legend({'y=mx+c', 'dy/dx', 'Integral of y'}, 'Location', 'best');
title('Original Function, Differentiation, and Integration');
xlabel('x');
ylabel('y');
grid on;
hold off;
```

Output B:-



EXPERIMENT – 8

Aim : Integrate and differentiate $\sin(x)$ and display the results on the same plot in different colors. Also display $\sin(x)$ on the same plot.

Description :- This MATLAB code integrates and differentiates the function $\sin(x)$ symbolically using the symbolic mathematics toolbox. It then calculates the corresponding values for each function over a range of x values. By plotting these functions on the same plot with different colors and line styles, the effects of differentiation and integration on the sine function are visually represented, aiding in understanding their relationships and behaviors.

1. $\sin(x)$:

- **Function:** Calculates the sine of an angle x in radians.
- **Input:** Numeric array or scalar representing angles in radians.
- **Output:** Numeric array or scalar with the corresponding sine values.
- **Example:** $y = \sin(\pi/2)$; (calculates \sin of $\pi/2$, which is 1).

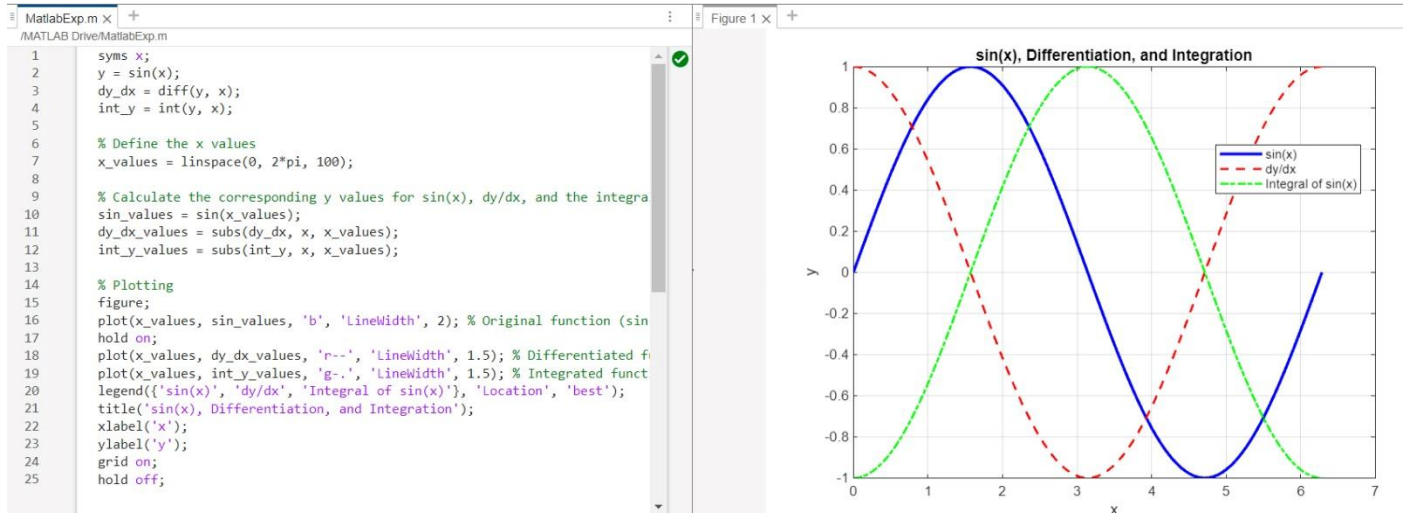
2. $\text{int}(\text{symbolic expression})$:

- **Function:** Performs symbolic integration of a mathematical expression.
- **Input:** Symbolic expression representing the function to be integrated.
- **Output:** Symbolic expression representing the indefinite integral of the input expression.

3. $\text{diff}(x)$:

- **Function:** Performs numerical differentiation of a numeric array or vector.
- **Input:** Numeric array or vector representing the data points.
- **Output:** Numeric array or vector with the approximate derivatives of the input data.
 - The length of the output is one less than the input length.
- **Example:** $\text{dy_dx} = \text{diff}(y_sin)$; (calculates the derivative of y_sin)

Implementation :-



Experiment 9

Aim: Compute mean, median, standard deviation and variance of a set of data using formulae and verify using built-in functions.

Mean:

In statistics, the mean is one of the measures of central tendency, apart from the mode and median. Mean is nothing but the average of the given set of values. It denotes the equal distribution of values for a given data set. Mean is the average of the given numbers and is calculated by dividing the sum of given numbers by the total number of numbers.

Mean = (Sum of all the observations/Total number of observations)

Example:

In a class there are 20 students and they have secured a percentage of 88, 82, 88, 85, 84, 80, 81, 82, 83, 85, 84, 74, 75, 76, 89, 90, 89, 80, 82, and 83.

Find the mean percentage obtained by the class.

Solution:

Mean = Total of percentage obtained by 20 students in class/Total number of students
= $[88 + 82 + 88 + 85 + 84 + 80 + 81 + 82 + 83 + 85 + 84 + 74 + 75 + 76 + 89 + 90 + 89 + 80 + 82 + 83]/20$
= $1660/20$
= 83

Hence, the mean percentage of each student in the class is 83%.

Median:

Median is defined as the middle value in a given set of numbers or data. The middle value of the given data is defined by a median.

Based on the definition, the formula to find the median of the dataset is given by:

If the given number of observations/data is odd, then the formula to calculate the median is:

Median = $\{(n+1)/2\}$ th term

If the given number of observations is even, then the formula to find the median is given by:

Median = $[(n/2)\text{th term} + \{(n/2)+1\}\text{th term}]/2$

Where,

“n” is the number of observations.

Example:

In a class there are 20 students and they have secured a percentage of 88, 82, 88, 85, 84, 80, 81, 82, 83, 85, 84, 74, 75, 76, 89, 90, 89, 80, 82, and 83.

Find the median.

Solution:

To find the median, you first need to arrange the percentages in ascending order:

74, 75, 76, 80, 80, 81, 82, 82, 82, 83, 83, 84, 84, 85, 85, 88, 88, 89, 89, 90

Since there are 20 students, an even number, the median will be the average of the 10th and 11th values in the sorted list:

Median = $(83 + 83) / 2 = 83$

So, the median percentage obtained by the class is 83.

Variance:

Variance is the measure of how notably a collection of data is spread out. If all the data values are identical, then it indicates the variance is zero. All non-zero variances are considered to be positive. A little variance represents that the data points are close to the mean, and to each other, whereas if the data points are highly spread out from the mean and from one another indicates the high variance.

Variance Formula:

The population variance formula is given by:

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^N (X_i - \mu)^2$$

The sample variance formula is given by:

$$s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$$

Standard Deviation:

Standard Deviation is a measure which shows how much variation (such as spread, dispersion, spread,) from the mean exists. The standard deviation indicates a “typical” deviation from the mean. It is a popular measure of variability because it returns to the original units of measure of the data set. Like the variance, if the data points are close to the mean, there is a small variation whereas the data points are highly spread out from the mean, then it has a high variance. Standard deviation calculates the extent to which the values differ from the average.

Standard Deviation Formula

The population standard deviation formula is given as:

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (X_i - \mu)^2}$$

Here,

σ = Population standard deviation

N = Number of observations in population

X_i = ith observation in the population

μ = Population mean

Similarly, the sample standard deviation formula is:

$$s = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2}$$

Here,

s = Sample standard deviation

n = Number of observations in sample

x_i = ith observation in the sample

\bar{x} = Sample mean

Example:

In a class there are 20 students and they have secured a percentage of 88, 82, 88, 85, 84, 80, 81, 82, 83, 85, 84, 74, 75, 76, 89, 90, 89, 80, 82, and 83.

Find the standard deviation and variance obtained by the class.

Solution:

To find the standard deviation, first, calculate the variance using the formula I provided earlier:

$$\text{Variance} = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$$

Then, take the square root of the variance to find the standard deviation.

Let's use the data provided:

$$\bar{x} = \frac{88+82+88+85+84+80+81+82+83+85+84+74+75+76+89+90+89+80+82+83}{20}$$
$$\bar{x} = \frac{1689}{20} = 84.45$$

Now, calculate the variance using the formula:

$$\text{Variance} = \frac{1}{20} \sum_{i=1}^{20} (x_i - 84.45)^2$$

$$\text{Variance} = \frac{1}{20} [(88 - 84.45)^2 + (82 - 84.45)^2 + \dots + (83 - 84.45)^2]$$

$$\text{Variance} = \frac{1}{20} [12.9025 + 6.0025 + \dots + 1.3225]$$

$$\text{Variance} = \frac{1}{20} \times 105.55$$

$$\text{Variance} = 5.2775$$

Now, to find the standard deviation, take the square root of the variance:

$$\text{Standard Deviation} = \sqrt{5.2775} \approx 2.30$$

So, the variance obtained by the class is approximately 5.28 and the standard deviation is approximately 2.30.

Inbuilt functions in MATLAB for finding:

1.Mean:

`M = mean(A)` returns the mean of the elements of A along the first array dimension whose size is greater than 1.

2.Median:

`M = median(A)` returns the median value of A.

3.Variance:

`V = var(A)` returns the variance of the elements of A along the first array dimension whose size is greater than 1. By default, the variance is normalized by N-1, where N is the number of observations.

4.Standard Deviation:

`S = std(A)` returns the standard deviation of the elements of A along the first array dimension whose size is greater than 1. By default, the standard deviation is normalized by N-1, where N is the number of observations.

Example of the same data set with all calculations with inbuilt functions in MATLAB:

Output:

u1.m × +

/MATLAB Drive/u1.m

```
1 % Given data
2 percentages = [88, 82, 88, 85, 84, 80, 81, 82, 83, 85, 84, 74, 75, 76, 89, 90, 89, 80, 82, 83];
3
4 % Mean
5 mean_percentage = mean(percentages);
6
7 % Median
8 median_percentage = median(percentages);
9
10 % Standard deviation
11 std_deviation = std(percentages);
12
13 % Variance
14 variance = var(percentages);
15
16 % Displaying results
17 fprintf('Mean: %.2f\n', mean_percentage);
18 fprintf('Median: %.2f\n', median_percentage);
19 fprintf('Standard Deviation: %.2f\n', std_deviation);
20 fprintf('Variance: %.2f\n', variance);
21
```

Command Window

New to MATLAB? See resources for [Getting Started](#).

>> u1
Mean: 83.00
Median: 83.00
Standard Deviation: 4.59
Variance: 21.05
>>

Experiment 10

Aim: Demonstrate (a) reading and display image, (b) converting color image to gray and black and-white and plotting their histograms, and (c) conversion of image file formats.

(a)Reading and displaying image:

Reading an image in MATLAB refers to the process of loading image data from a file into MATLAB's workspace. When you read an image, MATLAB creates a matrix representation of the image data, where each element of the matrix corresponds to a pixel in the image.

Syntax

```
A = imread(filename)

A = imread(filename,fmt)

A = imread(___,idx)

A = imread(___,Name,Value)

[A,map] = imread(___)

[A,map,transparency] = imread(___)
```

Description

A = imread(filename) reads the image from the file specified by filename, inferring the format of the file from its contents. If filename is a multi-image file, then imread reads the first image in the file.

A = imread(filename,fmt) additionally specifies the format of the file with the standard file extension indicated by fmt. If imread cannot find a file with the name specified by filename, it looks for a file named filename.fmt.

A = imread(___,idx) reads the specified image or images from a multi-image file. This syntax applies only to GIF, PGM, PBM, PPM, CUR, ICO, TIF, SVS, and HDF4 files. You must specify a filename input, and you can optionally specify fmt.

A = imread(___,Name,Value) specifies format-specific options using one or more name-value pair arguments, in addition to any of the input arguments in the previous syntaxes.

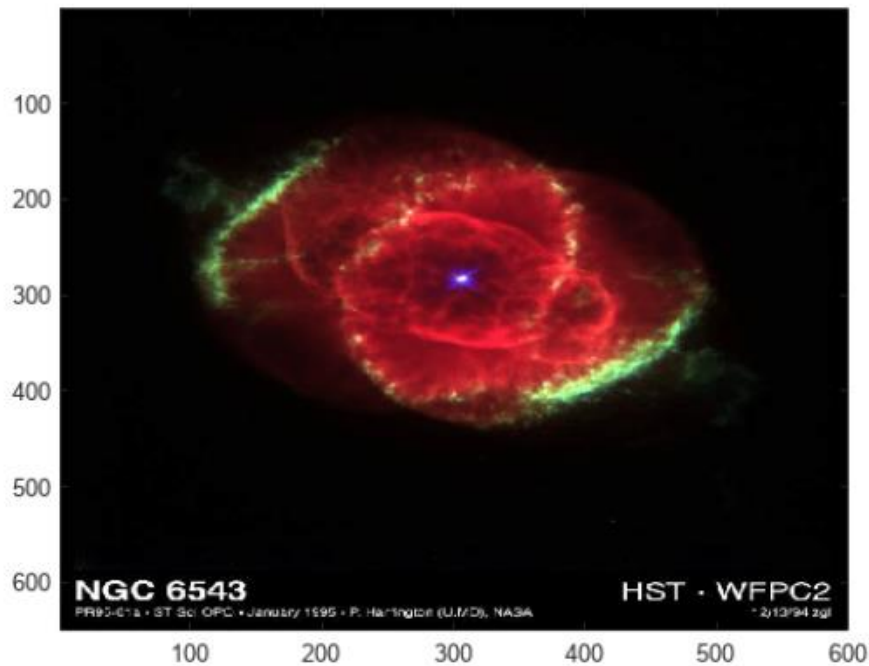
[A,map] = imread(___) reads the indexed image in filename into A and reads its associated colormap into map. Colormap values in the image file are automatically rescaled into the range [0,1].

[A,map,transparency] = imread(___) additionally returns the image transparency. This syntax applies only to PNG, CUR, and ICO files. For PNG files, transparency is the alpha channel, if one is present. For CUR and ICO files, it is the AND (opacity) mask.

Example:

```
A = imread('ngc6543a.jpg');
image(A)
```

Output:



Displaying an image in MATLAB refers to the process of visualizing the image matrix stored in MATLAB's workspace. Once an image is read and stored as a matrix, you can use the `imshow` function to display it as an image on your screen. To display image data, use the `imshow` function. The following example reads an image into the workspace and then displays the image in a figure window using the `imshow` function.

Example:

```
moon = imread("moon.tif");  
imshow(moon)
```

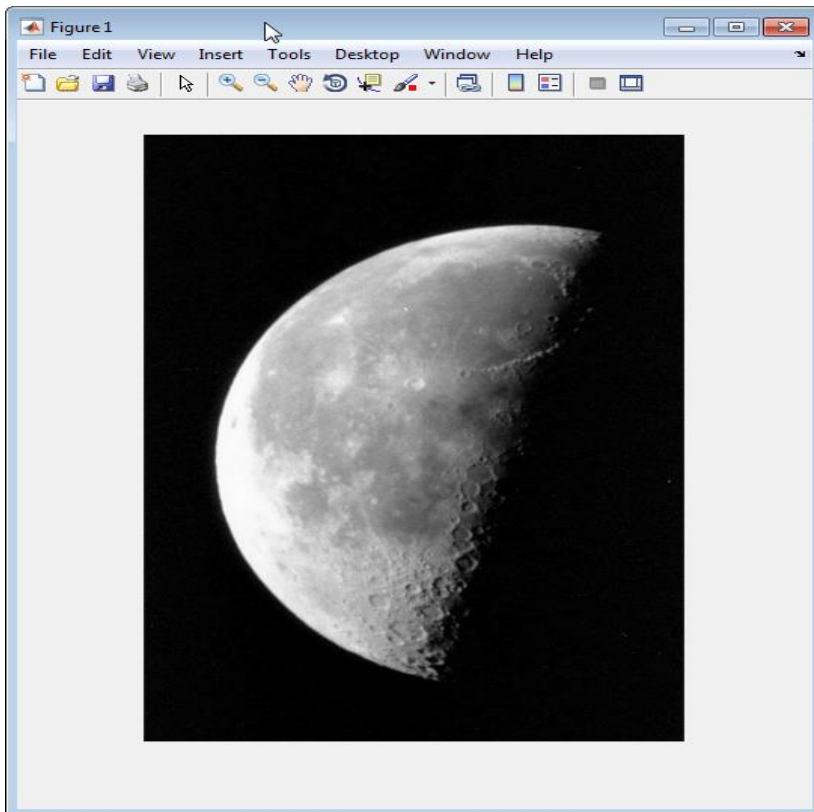
You can also pass `imshow` the name of a file containing an image.

```
imshow("moon.tif");
```

This syntax can be useful for scanning through images. Note, however, that when you use this syntax, `imread` does not store the image data in the workspace. If you want to bring the image into the workspace, you must use the `getimage` function, which retrieves the image data from the current image object. This example assigns the image data from `moon.tif` to the variable `moon`, if the figure window in which it is displayed is currently active.

```
moon = getimage;
```

Output:



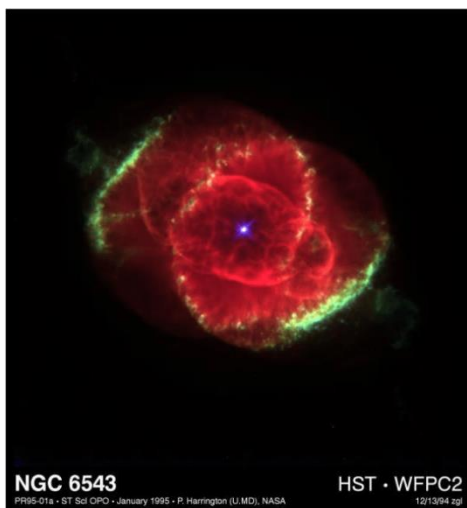
(b) converting color image to gray and black and-white and plotting their histograms

$I = \text{im2gray}(\text{RGB})$ converts the specified truecolor image RGB to a grayscale intensity image I. The `im2gray` function accepts grayscale images as inputs and returns them unmodified. The `im2gray` function converts RGB images to grayscale by eliminating the hue and saturation information while retaining the luminance.

Example:

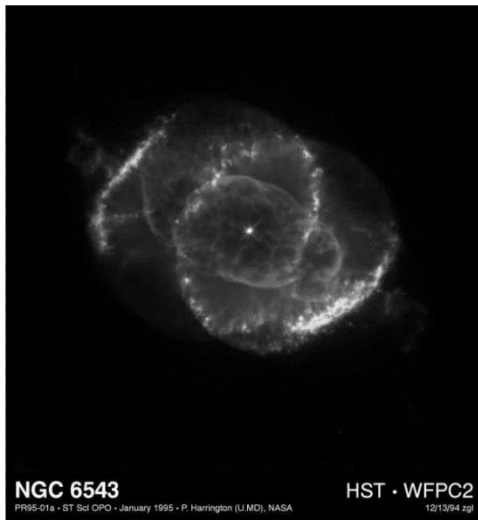
```
RGB = imread('example.tif');  
imshow(RGB)
```

Before:



```
I = im2gray(RGB);  
imshow(I)
```

After:



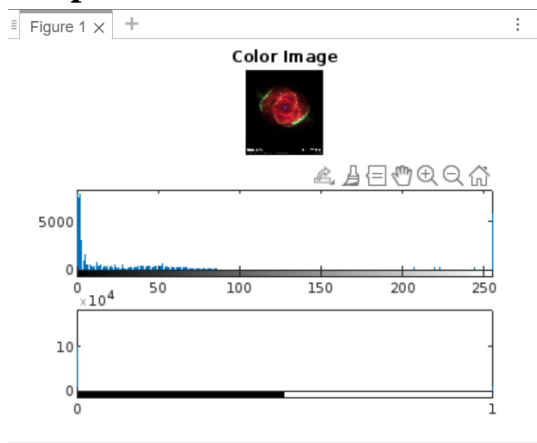
Plotting histogram:

Finally, we plot the original color image and its grayscale and black-and-white versions, along with their respective histograms using the subplot and imhist functions.

Code:

```
color_image = imread('/MATLAB Drive/Screenshot 2024-05-12 120853.png'); % Replace
'path_to_your_image.jpg' with the actual path to your image file
gray_image = rgb2gray(color_image);
threshold = graythresh(gray_image); % Compute a threshold automatically
bw_image = imbinarize(gray_image, threshold);
subplot(3, 1, 1);
imshow(color_image);
title('Color Image');
subplot(3, 1, 2);
imshow(gray_image);
title('Grayscale Image');
xlabel('Pixel Intensity');
ylabel('Frequency');
imhist(gray_image);
subplot(3, 1, 3);
imshow(bw_image);
title('Black-and-White (Binary) Image');
xlabel('Pixel Intensity');
ylabel('Frequency');
imhist(bw_image);
```

Output:



(c) conversion of image file formats

In MATLAB, you can use the `imwrite()` function to convert an image from one file format to another. Here's a simple example:

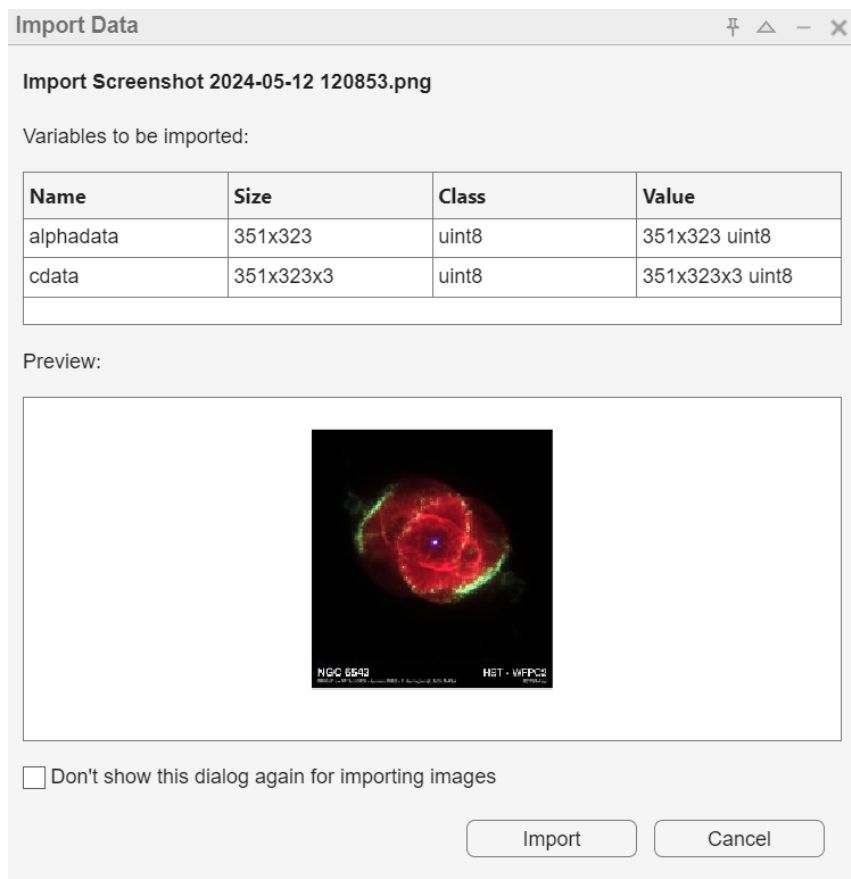
Code:

```
% Read the image
image_data = imread('/MATLAB Drive/Screenshot 2024-05-12 120853.png'); % Replace
'input_image.jpg' with the actual path to your input image file

% Convert and save the image in a different file format
imwrite(image_data, '/MATLAB Drive/Screenshot 2024-05-12 120853.jpg'); % Replace
'output_image.png' with the desired output file path and format
```

Output:

PNG:



JPG:

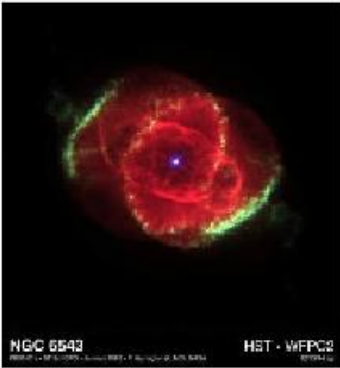
Import Data

Import Screenshot 2024-05-12 120853.jpg

Variables to be imported:

Name	Size	Class	Value
Screenshot2024_05_12120853	351x323x3	uint8	351x323x3 uint8

Preview:



☐ Don't show this dialog again for importing images

Import

Cancel