SOMAIYA VIDYAVIHAR UNIVERSITY

K. J. Somaiya College of Engineering, Mumbai-77

Batch: A-4	**Roll No:-** 16010122151
Experiment No:- 1	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

Title: Introduction to Matlab

Objective: To familiarize the beginner to MATLAB by introducing the basic features and commands of the program.

Expected Outcome of Experiment:

CO	Outcome
CO1	Identify various discrete time signals and systems and perform signal manipulation

Books/ Journals/ Websites referred:

1. http://www.mathworks.com/support/
2. www.math.mtu.edu/~msgocken/intro/intro.html
3. www.mccormick.northwestern.edu/docs/efirst/matlab.pdf

Pre Lab/ Prior Concepts:

INTRODUCTION TO MATLAB

MATLAB (MATrix LABoratory) is an interactive software system for numerical computations and graphics. As the name suggests, MATLAB is especially designed for

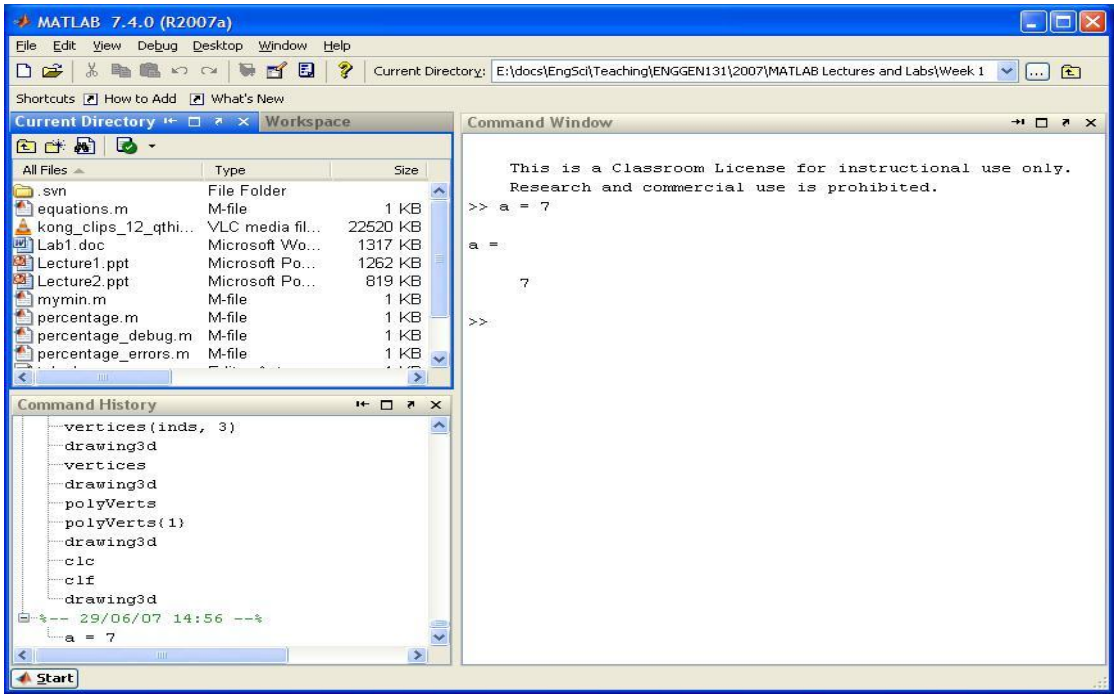
matrix computations: solving systems of linear equations, performing matrix transformations, factoring matrices, and so forth. In addition, it has a variety of graphical capabilities, and can be extended through programs written in its own programming language.

KEY FEATURES

- High-level language for numerical computation, visualization, and application development.
- Interactive environment for iterative exploration, design, and problem solving.
- Mathematical functions for linear algebra, statistics, Fourier analysis, filtering, optimization, numerical integration, and solving ordinary differential equations.
- Built-in graphics for visualizing data and tools for creating custom plots.
- Development tools for improving code quality and maintainability and maximizing performance.
- Tools for building applications with custom graphical interfaces.
- Functions for integrating MATLAB based algorithms with external applications and languages such as C, Java, .NET.

GETTING STARTED

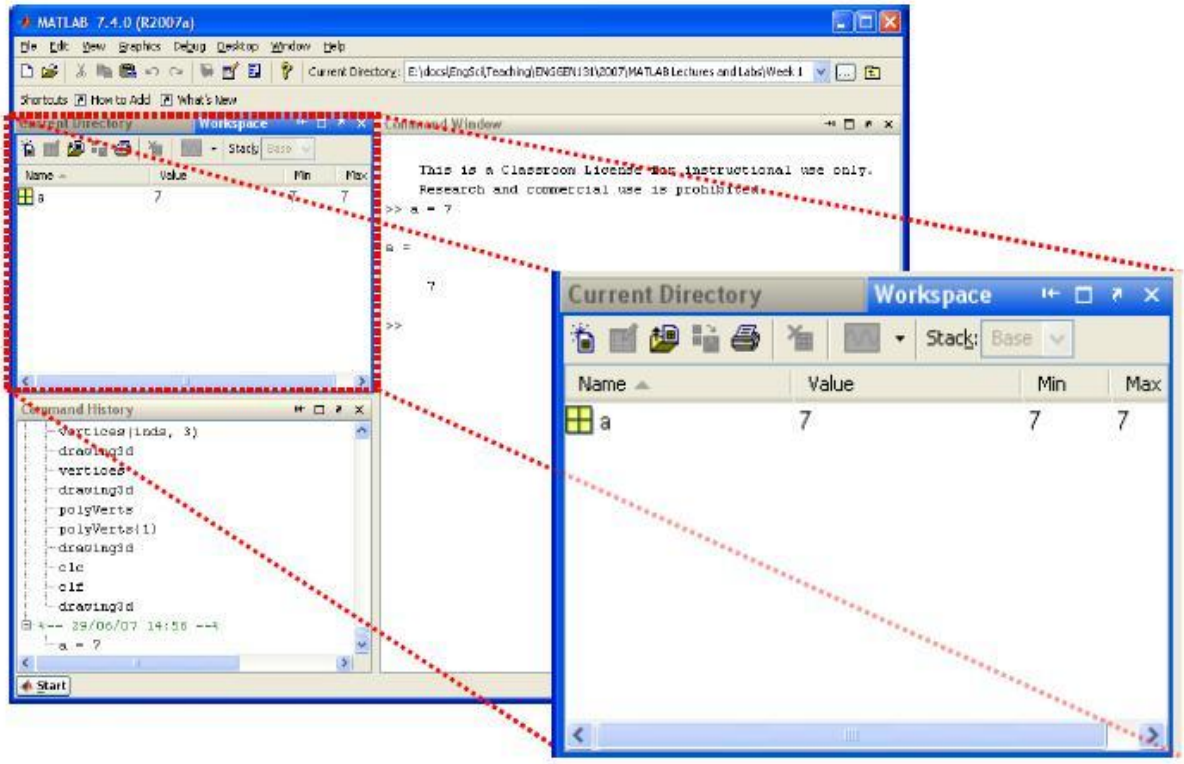
Double click on the MATLAB icon. The MATLAB window should come up on your screen. It looks like this:



The main window on the right is called the *Command* Window. This is the window in which you interact with MATLAB. Once the MATLAB prompt >> is displayed, all MATLAB commands are executed from this window. In the figure, you can see that we execute the command:

>> a = 7

In the top left corner you can view the *Workspace* window and the *Current Directory* window. Swap from one to the other by clicking on the appropriate tab. Note that when you first start up MATLAB, the workspace window is empty. However, as you execute commands in the *Command* window, the *Workspace* window will show all variables that you create in your current MATLAB session. In this example, the workspace contains the variable **a**.



In the bottom left corner you can see the *Command History* window, which simply gives a chronological list of all MATLAB commands that you used.

During the MATLAB sessions you will create files to store programs or workspaces. Therefore create an appropriate folder to store the lab files.

THE MATLAB HELP SYSTEM

MATLAB's help system provides information to assist you while using MATLAB. Select *Help for MATLAB Help* to open the help system.

We can then browse the commands via the *Contents* window, look through the *Index* of commands or *Search* for keywords or phrases within the documentation. This is the most comprehensive documentation for MATLAB and is the best place to find out what you need. You can also start the MATLAB *Help* using the **helpwin** command.

A faster way to find out information about commands is via the **help** command. If you just type **help** and hit return, you will be presented with a menu of help options. If you type **help <command>**, MATLAB will provide help for that particular command. For example, **help hist** will call up information on the **hist** command, which produces histograms. Note that the **help** command only displays a short summary of how to use the command you are interested in. For more detailed help documentation, use the **doc** command. For example, **doc hist** will display the full documentation of the **hist** command, including examples of how to use it and sample output.

Matlab Commands used for Signal and Image processing:

1. Imread
2. Imwrite
3. Imshow
4. Subplot
5. Title
6. Input
7. For
8. Nested for
9. If else
10. While
11. Continue
12. Break
13. Switch

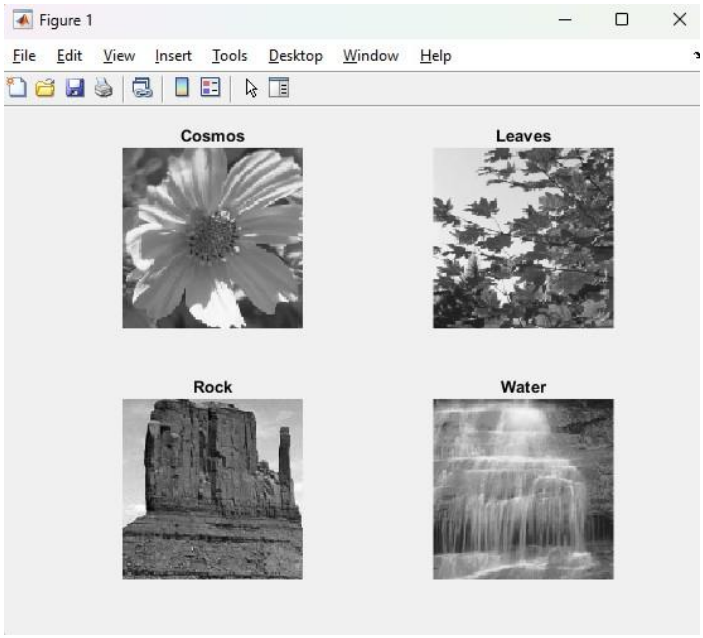
1) for , end imwrite, imshow, imread

```
A= imread("cosmos.bmp")
for i=1:128
   for j=1:128
       b(i,j)=255-A(i,j);
   end
end
imwrite(b,'imneg1.bmp');
imshow('imneg1.bmp')
```

Reading, Displaying, and Saving Images Using imread, imshow, imwrite, subplot, and title

```
% Code 1: Reading, Displaying, and Saving Images
% Read the images using imread
cosmos = imread('cosmos.bmp');
leaves = imread('leaves.bmp');
rock = imread('rock.bmp');
water = imread('water.bmp');
% Display images in a 2x2 grid using subplot
figure;
subplot(2,2,1);  % 1st image
imshow(cosmos);
title('Cosmos');
subplot(2,2,2);  % 2nd image
imshow(leaves);
title('Leaves');
subplot(2,2,3);  % 3rd image
imshow(rock);
title('Rock');
subplot(2,2,4);  % 4th image
imshow(water);
title('Water');
% Ask the user which image to save
disp('Choose an image to save:');
disp('1. Cosmos');
disp('2. Leaves');
disp('3. Rock');
disp('4. Water');
choice = input('Enter the number (1-4): ');
switch choice
   case 1
       imwrite(cosmos, 'cosmos_saved.bmp');
       disp('Cosmos image saved as cosmos_saved.bmp');
   case 2
       imwrite(leaves, 'leaves_saved.bmp');
       disp('Leaves image saved as leaves_saved.bmp');
   case 3
       imwrite(rock, 'rock_saved.bmp');
       disp('Rock image saved as rock_saved.bmp');
   case 4
       imwrite(water, 'water_saved.bmp');
       disp('Water image saved as water_saved.bmp');
   otherwise
       disp('Invalid choice.');
end
```

```
Choose an image to save:
1. Cosmos
2. Leaves
3. Rock
4. Water
Enter the number (1-4): 2
Leaves image saved as leaves_saved.bmp
```

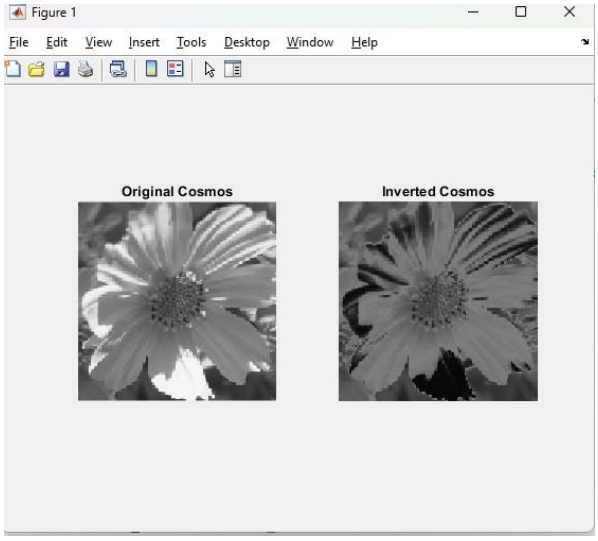


Looping Through Images with `for`, `nested for`, and `if-else` Statements

```
% Code 2: Looping Through Images with for and if-else
% Read the image using imread
cosmos = imread('cosmos.bmp');
[rows, cols, numChannels] = size(cosmos);  % Get the dimensions of the image
% Create a new image for the result
if numChannels == 1
   inverted_cosmos = zeros(rows, cols, 'uint8');  % Grayscale image
else
   inverted_cosmos = zeros(rows, cols, numChannels, 'uint8');  % Color image
end
% Loop through each pixel (nested for loops)
for i = 1:rows
   for j = 1:cols
       % Invert the color of the pixel for color images
```

```
        if numChannels == 3
            for k = 1:numChannels  % Loop through the RGB channels
                inverted_cosmos(i, j, k) = 255 - cosmos(i, j, k);
            end
        else
            % For grayscale images, just invert the pixel intensity
            inverted_cosmos(i, j) = 255 - cosmos(i, j);
        end

        % Use an if-else to check pixel intensity (for color and grayscale
images)
        if numChannels == 3
            if sum(cosmos(i, j, :)) < 255 * 3  % If the pixel is dark
                % Leave the pixel as is (no change)
                inverted_cosmos(i, j, :) = cosmos(i, j, :);
            else
                % If the pixel is bright, keep it inverted
                inverted_cosmos(i, j, :) = 255 - cosmos(i, j, :);
            end
        else
            % For grayscale images, check if the pixel intensity is dark
            if cosmos(i, j) < 128  % If the pixel is dark
                % Leave the pixel as is (no change)
                inverted_cosmos(i, j) = cosmos(i, j);
            else
                % If the pixel is bright, keep it inverted
                inverted_cosmos(i, j) = 255 - cosmos(i, j);
            end
        end
    end
end
% Show the original and inverted image
subplot(1,2,1);
imshow(cosmos);
title('Original Cosmos');
subplot(1,2,2);
imshow(inverted_cosmos);
title('Inverted Cosmos');
% Save the inverted image
imwrite(inverted_cosmos, 'cosmos_inverted.bmp');
disp('Inverted image saved as cosmos_inverted.bmp');
```

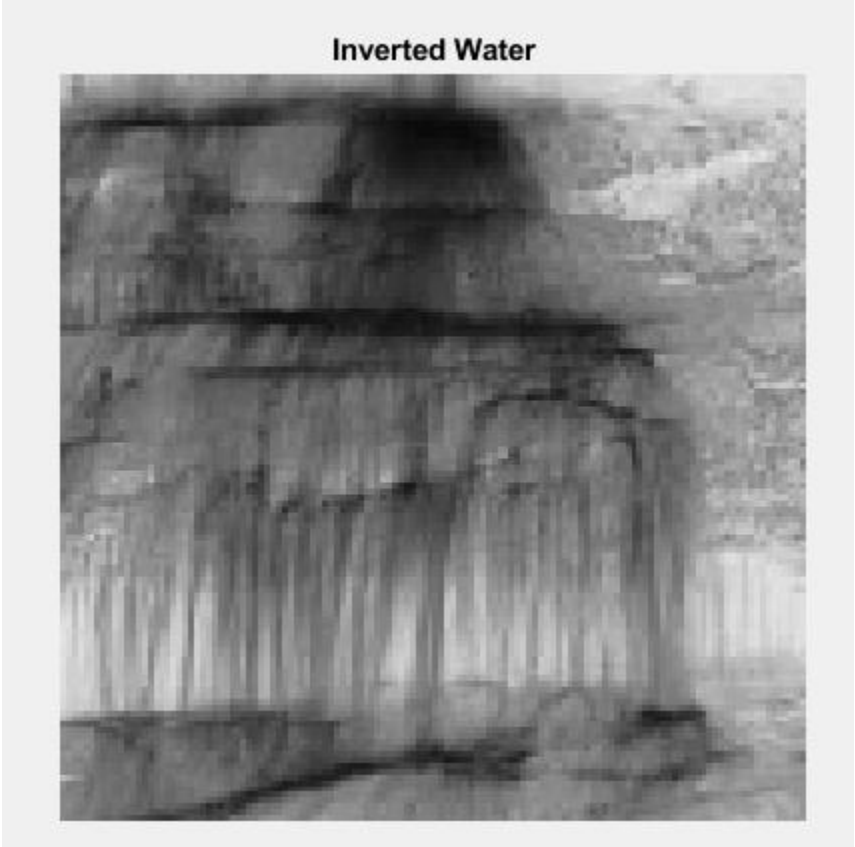


Using `while`, `continue`, `break`, and `input` for Interactive Image Processing

```
% Code 3: Interactive Image Processing with while, continue, break
% Read the images
cosmos = imread('cosmos.bmp');
leaves = imread('leaves.bmp');
rock = imread('rock.bmp');
water = imread('water.bmp');
% Define a cell array of images
images = {cosmos, leaves, rock, water};
image_names = {'Cosmos', 'Leaves', 'Rock', 'Water'};
% Start an interactive loop
while true
    disp('Choose an image to process:');
    disp('1. Cosmos');
    disp('2. Leaves');
    disp('3. Rock');
    disp('4. Water');
    disp('5. Exit');
    choice = input('Enter the number (1-5): ');
    if choice == 5
        disp('Exiting...');
        break;  % Exit the loop if the user chooses to quit
    elseif choice < 1 || choice > 4
        disp('Invalid choice. Try again.');
```

```
        continue;  % Skip the rest of the loop and prompt again
    end
    % Get the selected image
    selected_image = images{choice};
    selected_name = image_names{choice};
    % Ask the user to choose an operation
    disp('Choose an operation:');
    disp('1. Invert Colors');
    disp('2. Leave as is');
    operation = input('Enter the number (1 or 2): ');
    switch operation
        case 1
            % Invert colors of the image
            inverted_image = 255 - selected_image;
            imshow(inverted_image);
            title(['Inverted ', selected_name]);
            imwrite(inverted_image, [selected_name, '_inverted.bmp']);
            disp([selected_name, ' image inverted and saved.']);
        case 2
            % Just show the original image
            imshow(selected_image);
            title(['Original ', selected_name]);
        otherwise
            disp('Invalid operation. Try again.');
            continue;  % Skip the rest of the loop and prompt again
    end
end
```

```
>> Example1
Choose an image to process:
1. Cosmos
2. Leaves
3. Rock
4. Water
5. Exit
Enter the number (1-5): 4
Choose an operation:
1. Invert Colors
2. Leave as is
Enter the number (1 or 2): 1
Water image inverted and saved.
Choose an image to process:
1. Cosmos
2. Leaves
3. Rock
4. Water
5. Exit
Enter the number (1-5):
```



K. J. Somaiya College of Engineering, Mumbai-77
(Autonomous College Affiliated to University of Mumbai)

Conclusion:- Learned basics of Matlab

Date: 22-01-2025 **Signature of faculty in-charge**

❖ Post-Lab Descriptive Questions :

1. Create a vector of the even whole numbers between 31 and 75.

Ans:

```
even_numbers = 32:2:74;
disp(even_numbers);
```

```
    32    34    36    38    40    42    44    46    48    50    52    54    56    58    60    62
```

2. Let x = [2 5 1 6].

a) Add 16 to each element
b) Add 3 to just the odd-index elements
c) Compute the square root of each element
d) Compute the square of each element

Ans:

```
x = [2 5 1 6];


% a) Add 16 to each element
result_a = x + 16;


% b) Add 3 to just the odd-index elements
result_b = x;
result_b(1:2:end) = result_b(1:2:end) + 3;


% c) Compute the square root of each element
result_c = sqrt(x);


% d) Compute the square of each element
result_d = x.^2;


% Displaying the results
disp('a) Add 16 to each element:');
disp(result_a);
```

```
disp('b) Add 3 to just the odd-index elements:');
disp(result_b);


disp('c) Compute the square root of each element:');
disp(result_c);


disp('d) Compute the square of each element:');
disp(result_d);
```

Output:

```
a) Add 16 to each element:
    18    21    17    22

b) Add 3 to just the odd-index elements:
     5     5     4     6

c) Compute the square root of each element:
    1.4142    2.2361    1.0000    2.4495

d) Compute the square of each element:
     4    25     1    36
```

3. Let x = [3 2 6 8]' and y = [4 1 3 5]' (NB. x and y should be column vectors).

a. Add the sum of the elements in x to y
b. Raise each element of x to the power specified by the corresponding element in y.
c. Divide each element of y by the corresponding element in x
d. Multiply each element in x by the corresponding element in y, calling the result "z".
e. Add up the elements in z and assign the result to a variable called "w".
f. Compute x' *y - w and interpret the result

Ans:

```
% Given column vectors
x = [3; 2; 6; 8];
y = [4; 1; 3; 5];


% a. Add the sum of the elements in x to y
result_a = y + sum(x);
```

```
% b. Raise each element of x to the power specified by the corresponding element in
y.
result_b = x.^y;


% c. Divide each element of y by the corresponding element in x
result_c = y ./ x;


% d. Multiply each element in x by the corresponding element in y, calling the
result "z".
z = x .* y;


% e. Add up the elements in z and assign the result to a variable called "w".
w = sum(z);


% f. Compute x' *y - w and interpret the result
result_f = x' * y - w;


% Displaying the results
disp('a. Add the sum of the elements in x to y:');
disp(result_a);


disp('b. Raise each element of x to the power specified by the corresponding element
in y:');
disp(result_b);


disp('c. Divide each element of y by the corresponding element in x:');
disp(result_c);


disp('d. Multiply each element in x by the corresponding element in y, calling the
result "z":');
disp(z);


disp('e. Add up the elements in z and assign the result to a variable called "w":');
disp(w);


disp('f. Compute x'' * y - w and interpret the result:');
disp(result_f);
```

Output:

```
a. Add the sum of the elements in x to y:
    23
    20
    22
    24
b. Raise each element of x to the power specified by the corresponding element in y:
          81
           2
         216
       32768
c. Divide each element of y by the corresponding element in x:
    1.3333
    0.5000
    0.5000
    0.6250
d. Multiply each element in x by the corresponding element in y, calling the result "z":
    12
     2
    18
    40
e. Add up the elements in z and assign the result to a variable called "w":
    72
f. Compute x' * y - w and interpret the result:
     0
```

4. Create a vector x with the elements,

$x_n = (-1)^{n+1}/(2n-1)$

Add up the elements of the version of this vector that has 100 elements.

Ans:

```
% Initialize the vector x with 100 elements
n = 1:100;
x = (-1).^(n+1) ./ (2*n - 1);


% Add up the elements
result = sum(x);


% Display the result
```

```
disp('Sum of the elements of the vector with 100 elements:');
disp(result);
```

Output:

```
Sum of the elements of the vector with 100 elements:
    0.7829
```

5. **Given the array A = [2 4 1; 6 7 2; 3 5 9], provide the commands needed to**

a. Assign the first row of A to a vector called x1

b. Assign the last 2 rows of A to an array called y

c. Compute the sum over the columns of A

d. Compute the sum over the rows of A

e. Compute the standard error of the mean of each column of A (NB. the standard error of the mean is defined as the standard deviation divided by the square root of the number of elements used to compute the mean.

Ans:

```
% Given array A
A = [2 4 1; 6 7 2; 3 5 9];


% a. Assign the first row of A to a vector called x1
x1 = A(1, :);


% b. Assign the last 2 rows of A to an array called y
y = A(2:end, :);


% c. Compute the sum over the columns of A
column_sum = sum(A);


% d. Compute the sum over the rows of A
row_sum = sum(A, 2);
```

```
% e. Compute the standard error of the mean of each column of A
num_elements = size(A, 1); % Number of elements used to compute the mean
mean_A = mean(A);
std_error_mean = std(A) / sqrt(num_elements);


% Displaying the results
disp('a. Vector x1:');
disp(x1);


disp('b. Array y:');
disp(y);


disp('c. Column sum of A:');
disp(column_sum);


disp('d. Row sum of A:');
disp(row_sum);


disp('e. Standard error of the mean of each column of A:');
disp(std_error_mean);
```

Output:

```
a. Vector x1:
     2     4     1

b. Array y:
     6     7     2
     3     5     9
c. Column sum of A:
    11    16    12

d. Row sum of A:
     7
    15
    17
e. Standard error of the mean of each column of A:
    1.2019    0.8819    2.5166
```