**TITLE OF LAB: (INTRODUCTION TO MATRICES)**

**LAB # 02**



**Spring 2022**

**CSE301L Signals & Systems Lab**

Submitted by: **Safi Ullah Khan**

Registration No.: **20PWCSE1943**

Class Section: **B**

“On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work.”

Student Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Submitted to:

**Engr. Durr-e-Nayab**

Day, Date (e.g Friday, April 22th, 2022)

**Department of Computer Systems Engineering**

**University of Engineering and Technology, Peshawar**

**GETTING STARTED WITH MATLAB**

**Lab Objective(s):**

* To be able to define matrices in MATLAB
* To get hands on experience of using different type of matrices.
* To know about the built-in matrices that are provided by the MATLAB (eye, zeros etc.).
* To know about the built-in commands used on matrices.
* To know how to slice a matrix.
* To know how to perform arithmetic on matrices.

# -------------------------TASK 01--------------------------

# Write a program to generate a new matrix B from the matrix A given below such that each column in the new matrix except the first one is the result of subtraction of that column from the previous one i.e., 2nd new column is the result of subtraction of 2nd column and 1st column and so on. Copy the first column as it is in the new matrix.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 13 | 6 | 9 |
| 𝐴 = | 1 | 8 | 4 |
|  | 2 | 8 | 17 |

**Problem Analysis:**

This problem requires a little algorithmic solution. Firstly, we have a 3x3 matrix that is initialized to above values. Then we make another 3x3 matrix and we copy the first column as it is. This problem focuses around indices of matrices. Even if we do it using loops, it would still cover the entire indices related concepts. I am doing it using for loop since using loop makes the code more compact, readable, fast and, above all, generalized. Since it’s a 2-dimensional array, it would require 2 level nested loop. The enclosing loop runs till the number of rows and the nested loop runs till the number of columns but starts from 2 since the first column is copied as it is.

**Algorithm:**

Step 1) Initialized a matrix *mat* to the given values

Step 2) Find the size of matrix using size function, the returned value is a two-element matrix where first element is number of rows while second is number of columns.

Step 3) Declare another matrix mat1 which has same dimensions as mat. This mat1 will store the resultant matrix.

Step 4) Initialize *i* to 1.

Step 5) If i is greater than the number of rows then jump to Step 14.

Step 6) Set the value at row number i and column number 1 of mat1 to its corresponding element in mat.

Step 7) Initialize j to 2.

Step 8) If j is greater than the number of columns, than jump to Step 12.

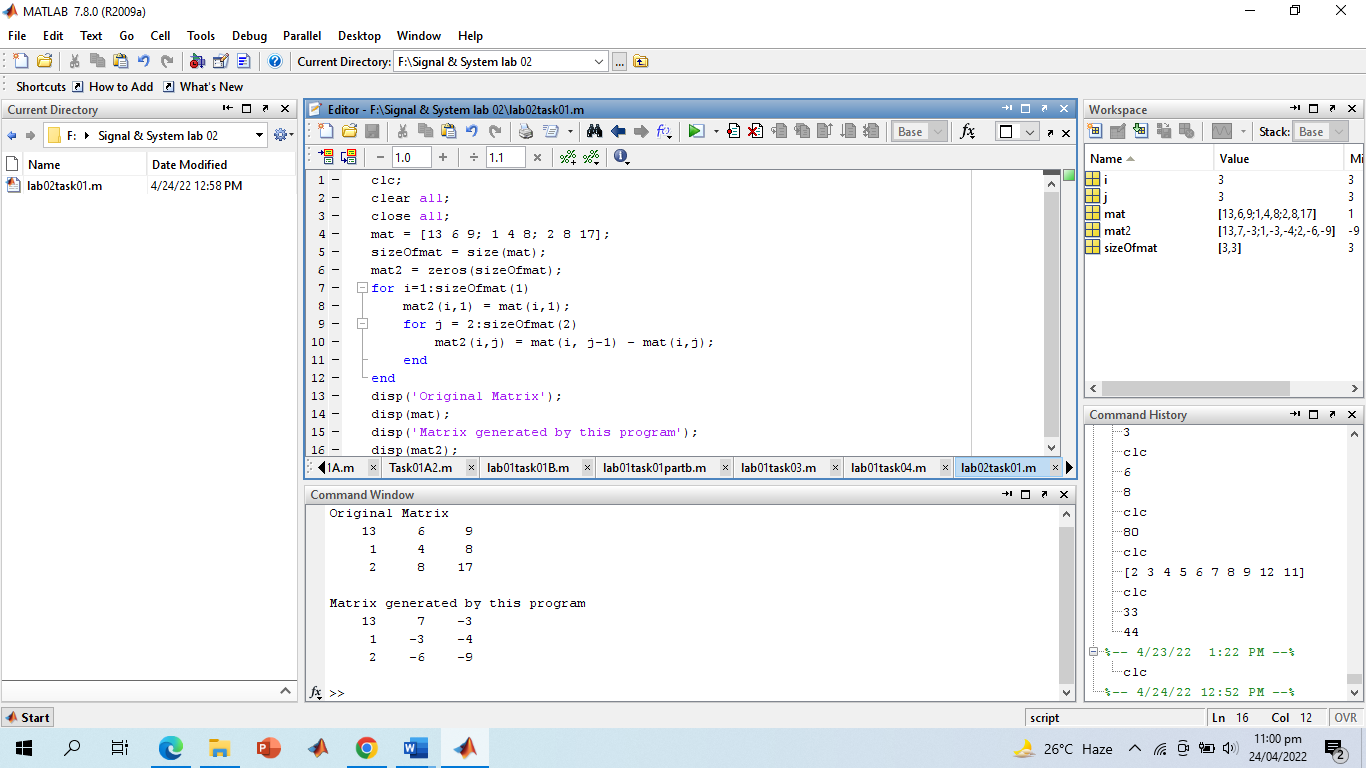
Step 9) Set the element value at row i and column j of mat1 equal to the difference of values of elements at row i and column j -1 and row i and column j. (mat1(i, j) = mat (i, j) – mat (i, j-1).

Step 10) Increment value of j by 1. Step 11) Jump to Step 8.

Step 12) Increment the value of i by 1. Step 13) Jump to Step 5.

Step 14) Display the newly created mat1.

**Screenshot of Source Code and Output:**



**Discussion and Conclusion:**

We see that all the concepts about indices are very well covered even this loop approach. I declared mat2 using the built-in zeros functions. This function takes in dimension and returns a matrix of that dimension where each element is 0. This way, all elements of mat1 are initially set to 0.

Here I used size function instead hard-coding values in matrix declaration and loop. This is because I put maximum emphasis on generalization of code. Now we are no longer limited to a 3x3 matrix. We can just mat to any dimension without changing anything in the rest of the code and the code would still work all fine.

**-------------------------TASK 02--------------------------**

Write a Generate two 2500 sampled random discrete time signals (1 dimensional) using rand () function i.e., rand (1, 2500). Write a program to add the two such random signals together using simple vector addition.

**Problem Analysis:**

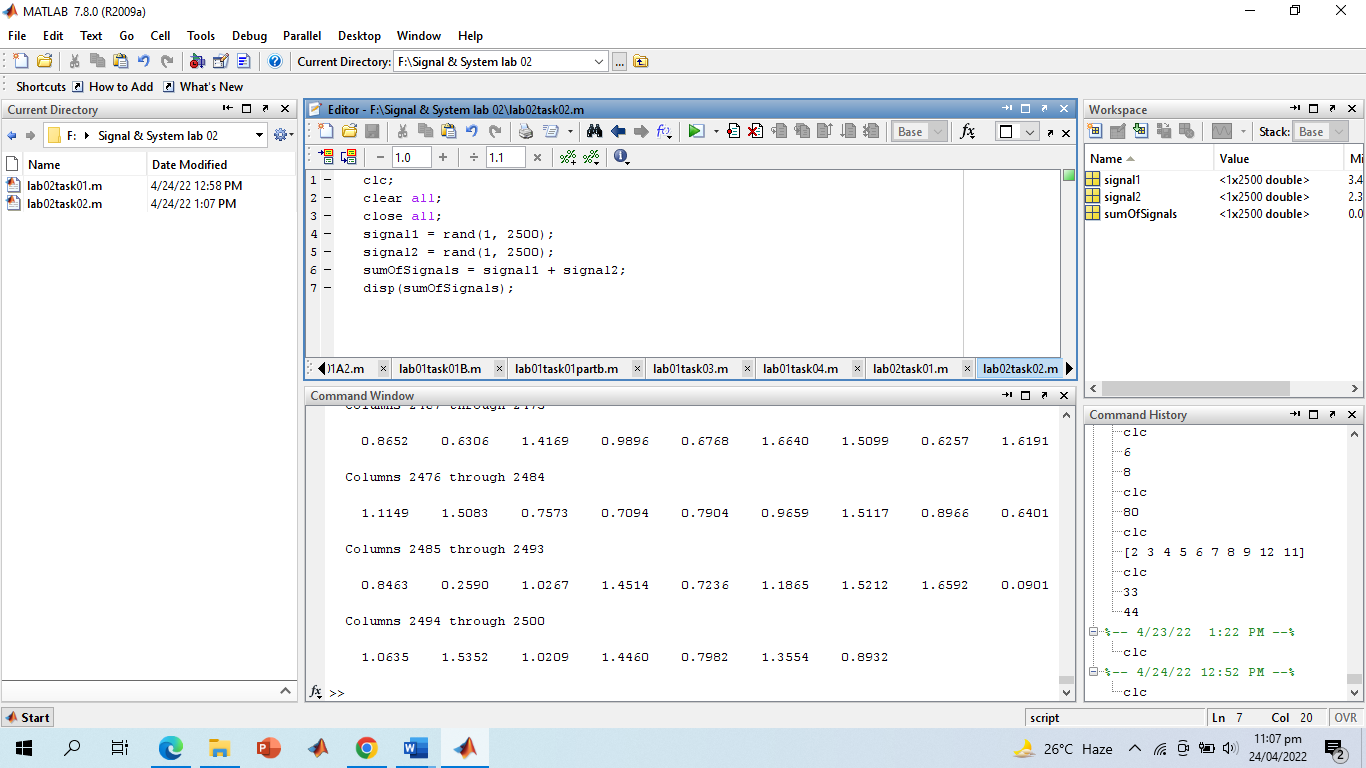
We need to create two 1 dimensional signals. For this I will use rand(m, n) functions. This functions basically returns an mxn matrix whose elements are uniformly distributed in [0,1] interval. Since the signal is 1 dimension, m will be 1 and since we need the signal sampled at 2500, so n will be 2500. I will create two such functions and then simply add them.

**Algorithm:**

Step 1) Create two signals using rand (1, 2500). Step 2) Add the two matrices.

Step 3) Display the resultant matrix.

**Screen shot of Source Code:**



**Screenshot of Output:**



# ------------------------TASK 03--------------------------

Using colon notation, generate the following sequence: -65.25, -57.75, -50.25. ,

54.75, 62.25, 69.75

**Problem Analysis:**

After little analysis, we see that it is arithmetic series starting from -65.25 and increments by 7.5. Matlab provides a built-in series command.

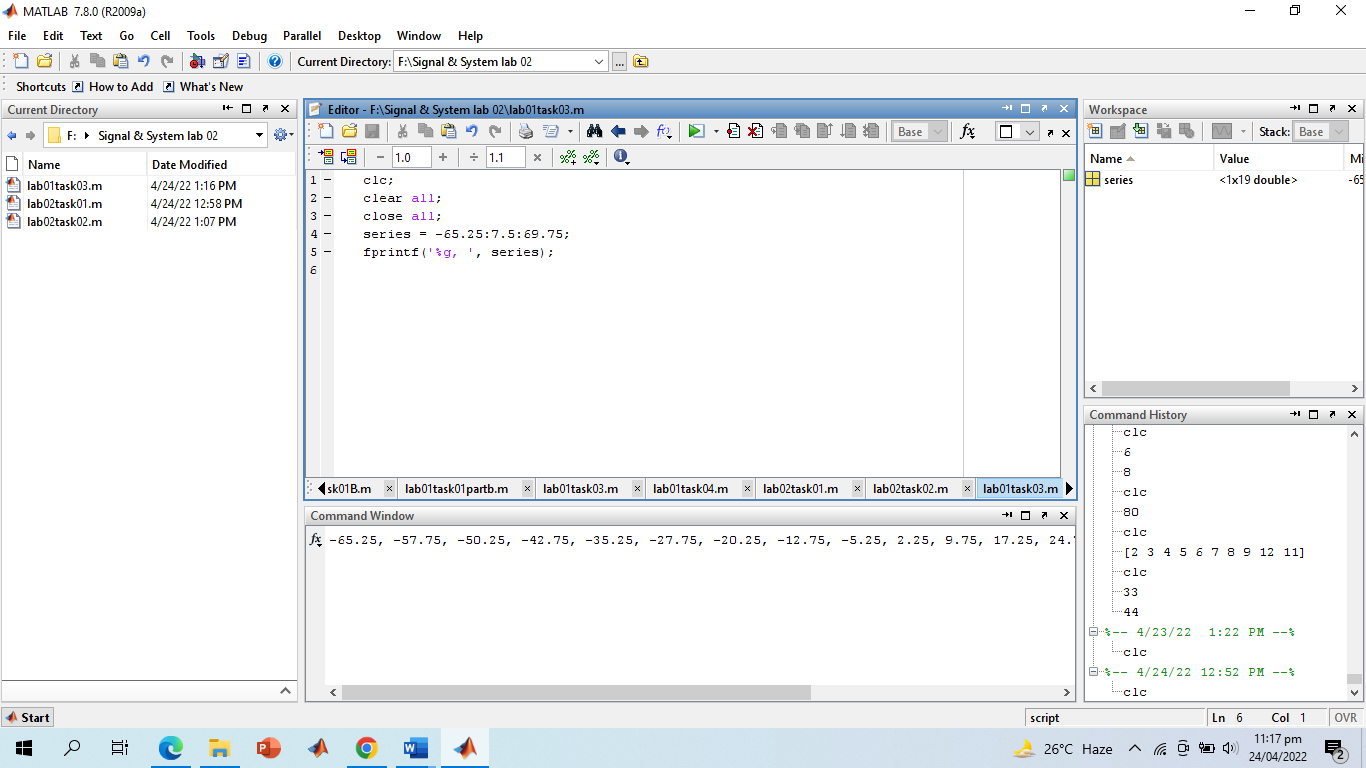
M = initial: step: final

The above store an arithmetic series in M that starts from initial value, increments by step until final.

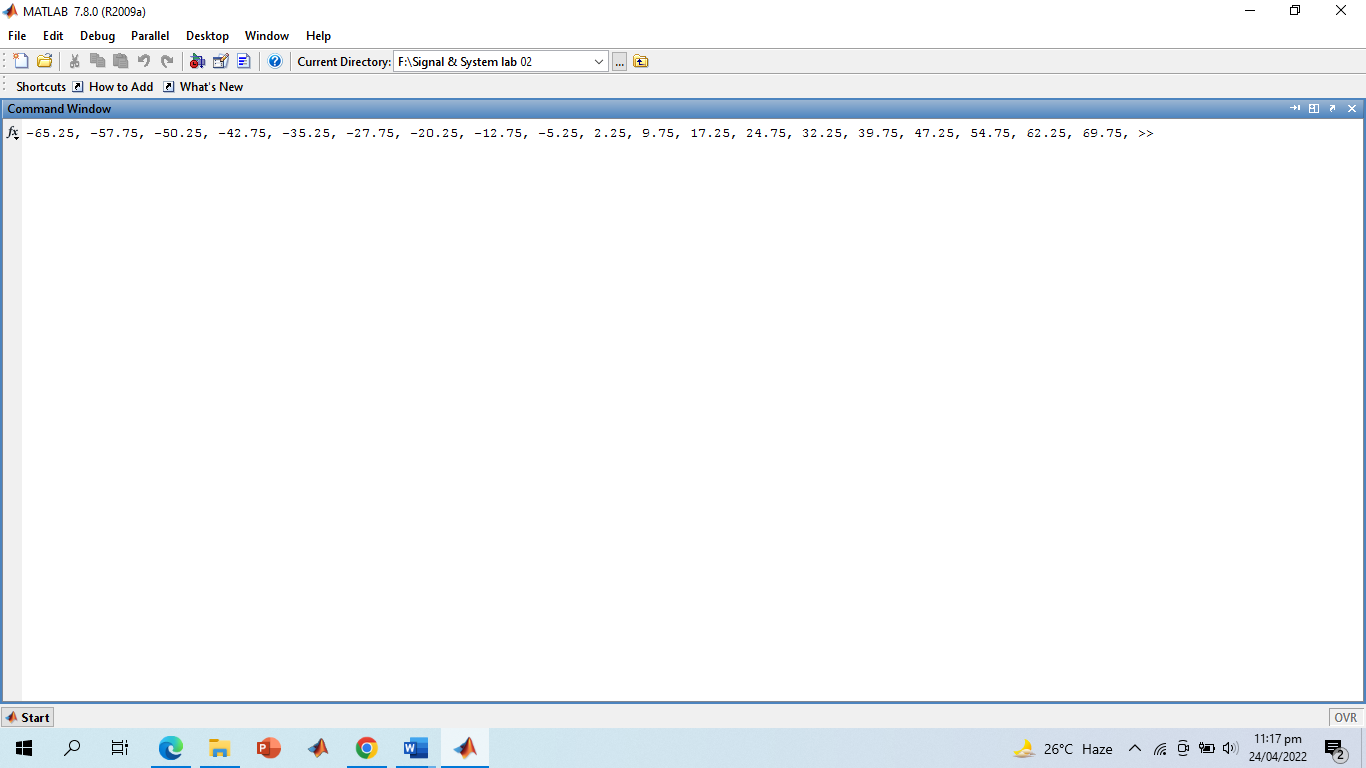
**Algorithm:**

Step 1) Using the built-in MATLAB series, store -65.25:7.5:69.75 in M. Step 2) Display M

**Screenshot of Source Code:**



**Screenshot of Output:**



# -------------------------TASK 04--------------------------

Write a Given the matrices:

A= [-12,34,61,-9;65,78,90,12; 14,78,45,12; 60,25,3,8]

B= [34,67,8,9; 12,-91,12,9; 89,-8,0,2; 16,9,23,67]

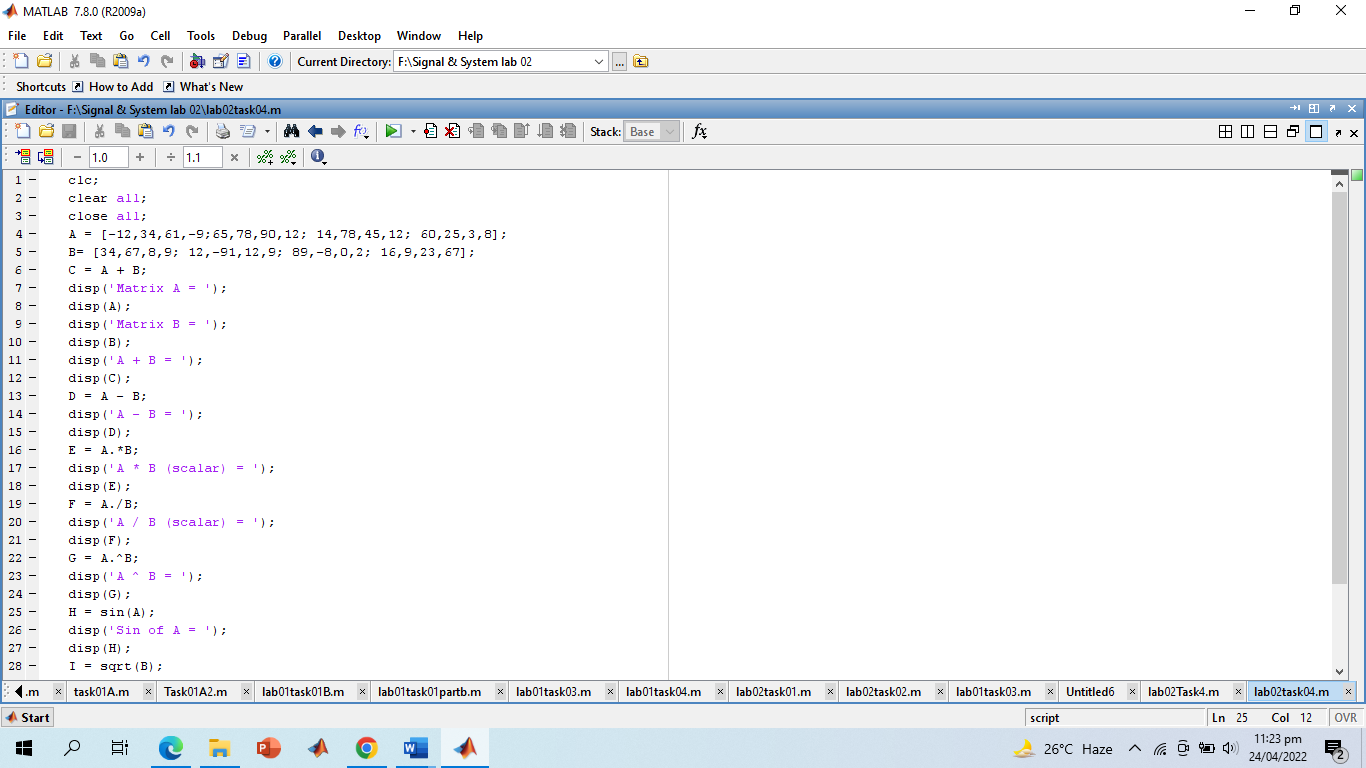
Find the following:

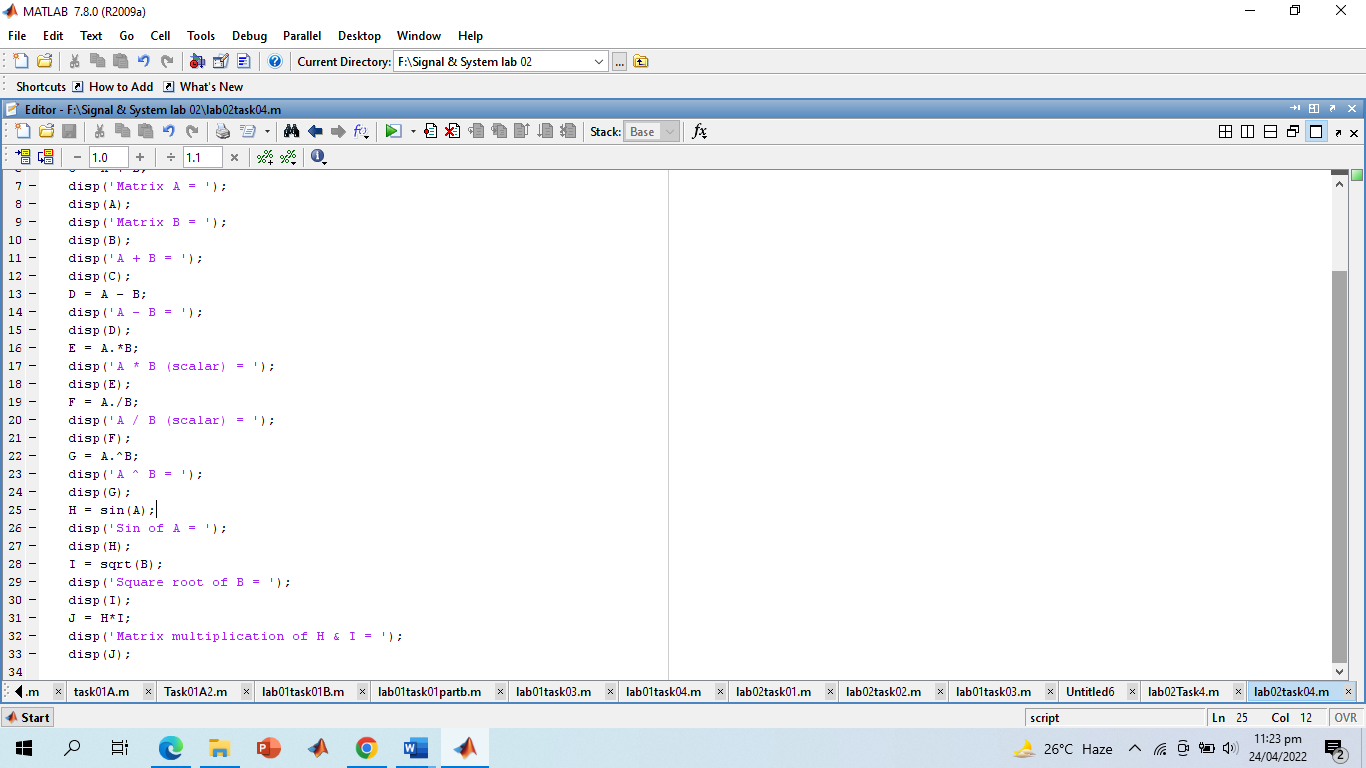
1. Array addition; store the result in matrix C
2. Array subtraction; store the result in matrix D
3. Array multiplication using .\* operator; store the result in matrix E
4. Array division using ./ operator; store the result in matrix F
5. Array exponentiation using .^ operator; store the result in matrix G
6. Take sin of A and store the result in H, Take sqrt of B and store the result in I. Find H\*I and store the result in J

**Problem Analysis:**

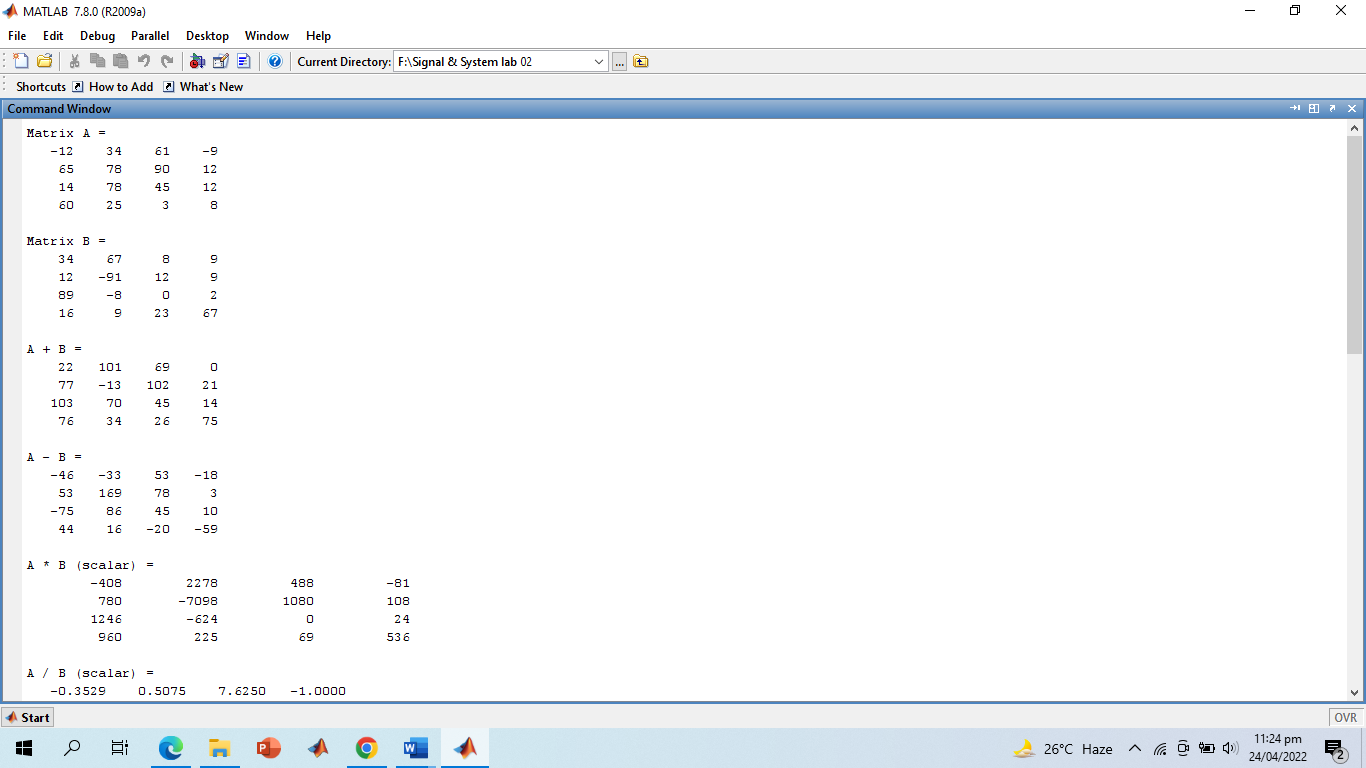
For most part, this task is a straight forward coding task. The only thing that needs to be explained is the difference between .\* and \* operation. The .\* operation is array multiplication operator and it simply multiplies the corresponding elements of two operand arrays. That is it multiplies element at index i, j of right side matrix with element at index i,j of left side index for all i and j. The \* operator, on the other hand, follows the matrix multiplication procedure. This program really does not need an additional algorithm section.

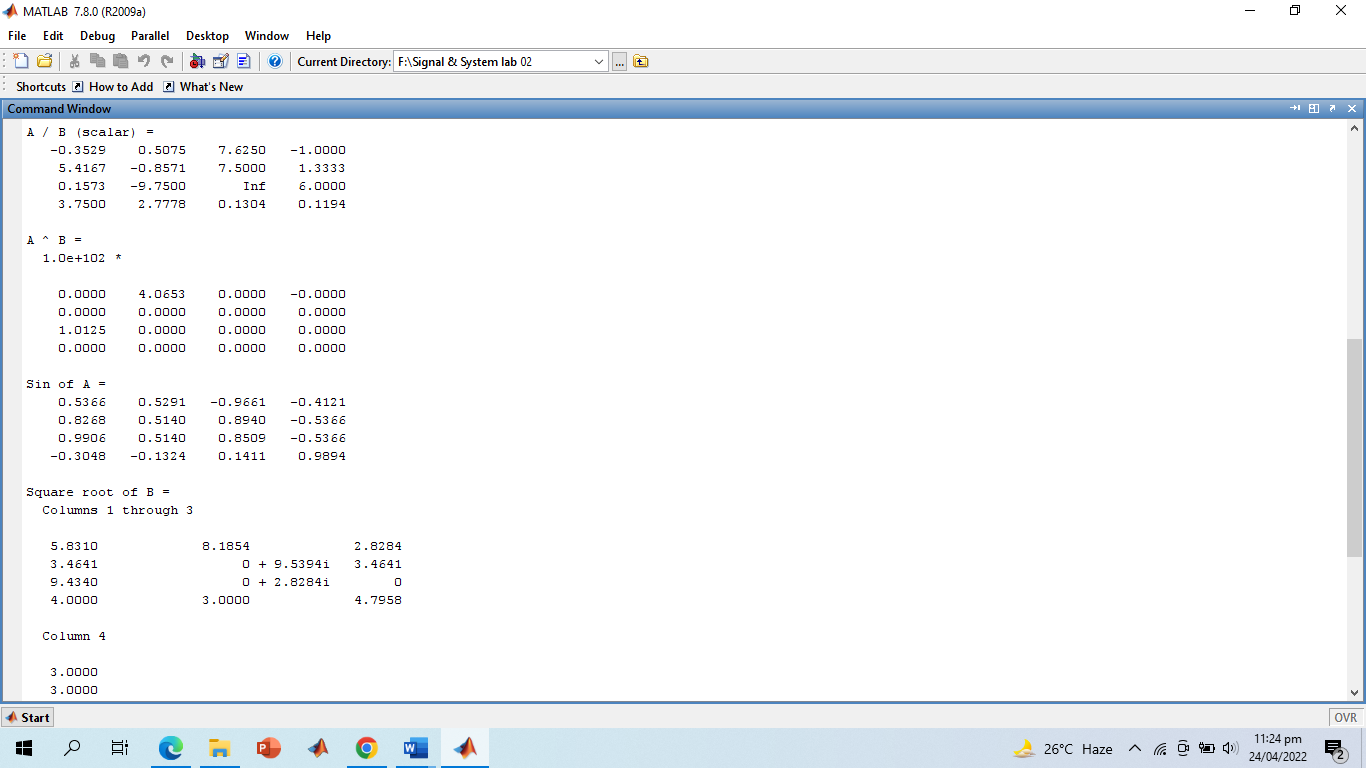
**Screenshot of Output:**

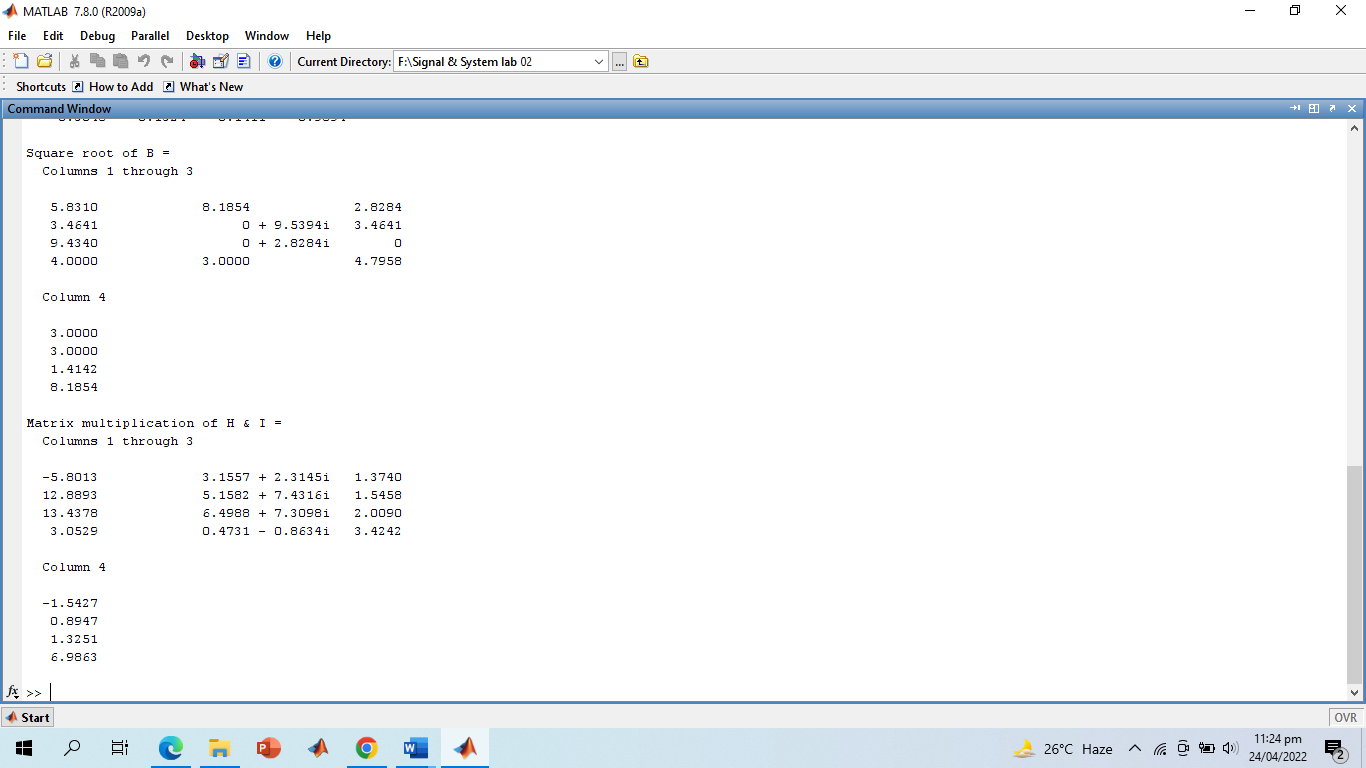




**Screenshot of Output:**







**-------------------------TASK 05--------------------------**

Type the given matrix in Matlab:

3 7 − 4 12

-5 9 10 2

A = 6 13 8 11

15 5 4 1

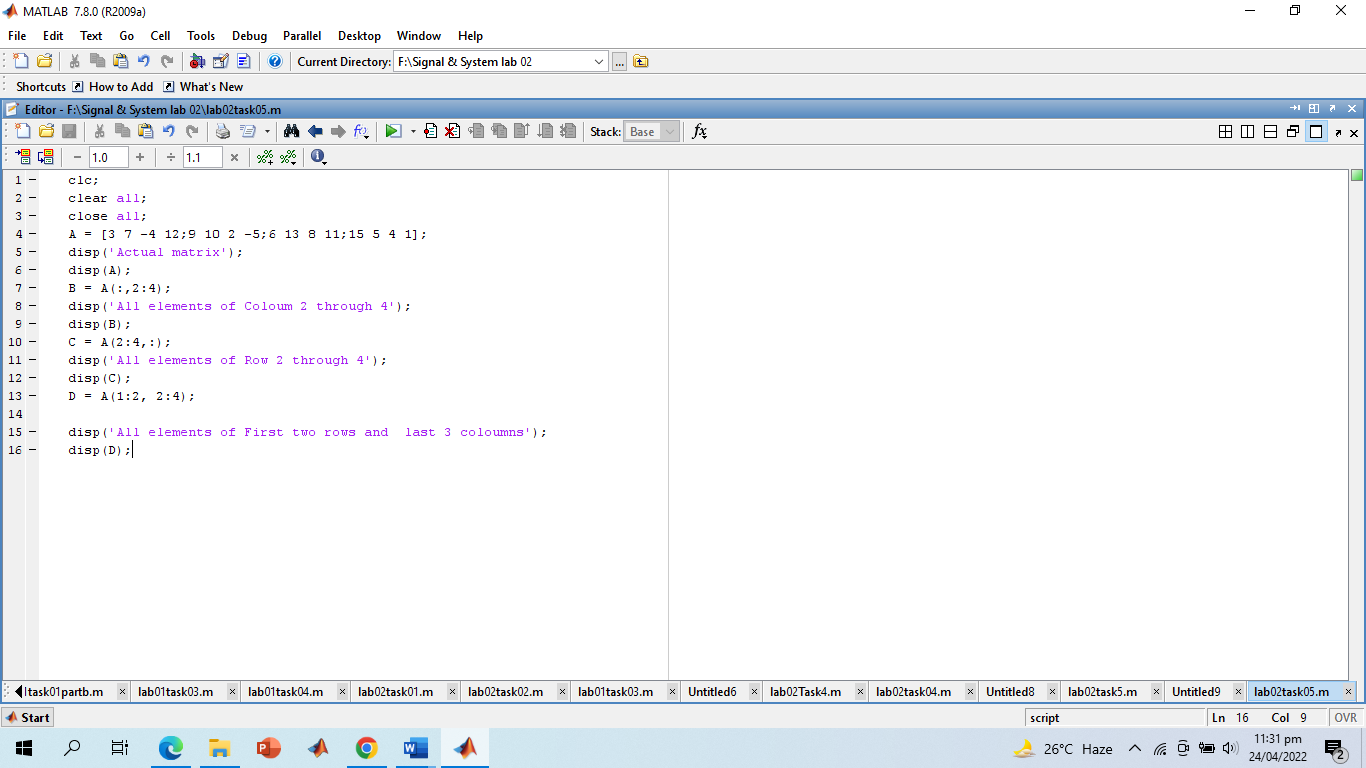
Find the following:

1. Create 4x3 array B consisting of all elements in the second through fourth columns of A
2. Create 3x4 array C consisting of all elements in the second through fourth rows of A
3. Create 2x3 array D consisting of all elements in the first two rows and the last three columns of A

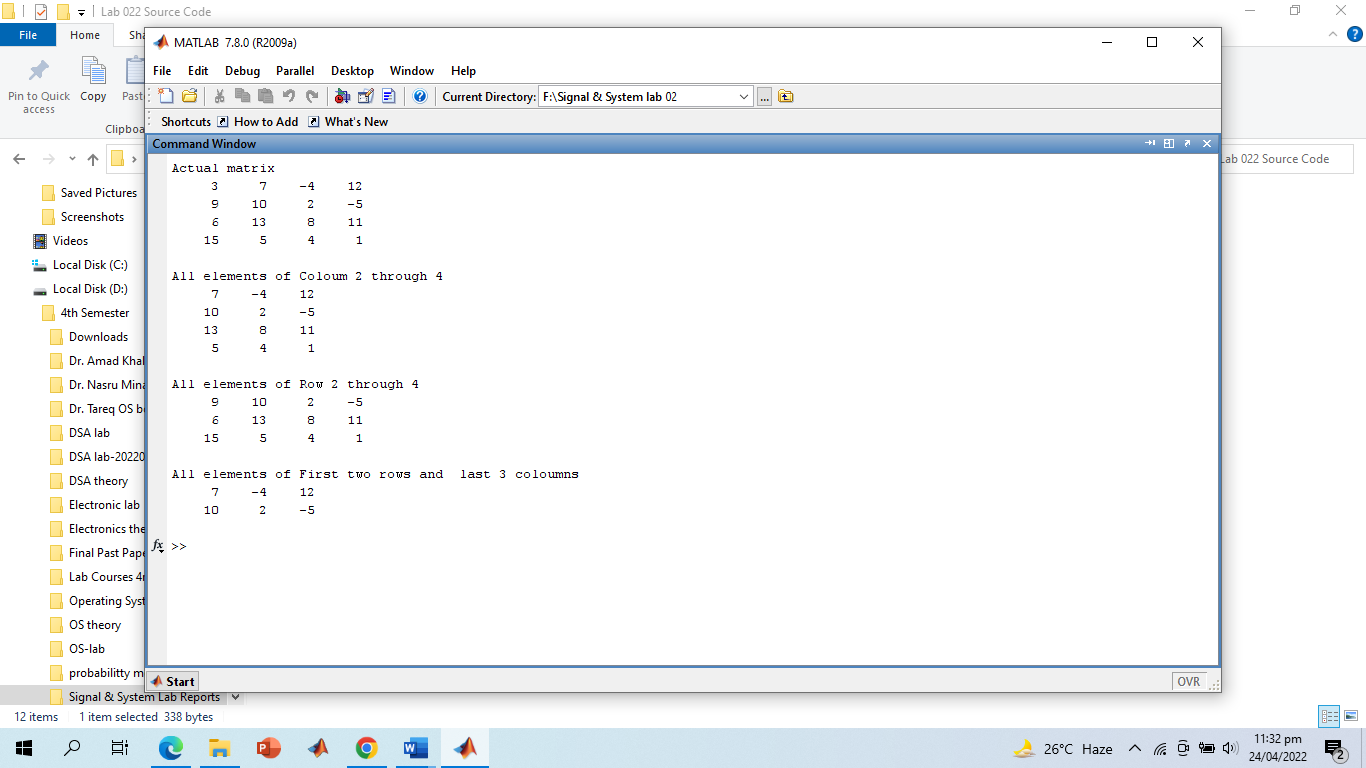
**Problem Analysis:**

The only thing we need to understand is the concept of slicing matrix in MATLAB. Say we have a matrix AB then the commands AB(x:y, t:s) will return rows x to y of columns t to s. If we omit rows than all rows are returned and if we omit columns, then all columns are returned. For first part, we will use A(:, 2:4). Notice that the rows part is omitted so all rows will be returned. For columns part, column 2 to 4 will get retuned. For Second part, we will just swap the ranges of columns an rows A(2:4,:). For third part will use A(1:2, 2:4). This once again requires no special algorithm section.

**Screenshot of Source Code:**



**Screenshot of Output:**



# -------------------------TASK 06--------------------------

Write a MATLAB has functions to round floating point numbers to integers. These are round, fix, ceil, and floor. Test how these functions work. Determine the output of the following:

>> f = [‐.5 .1 .5];

>> round(f)

>> fix(f)

>> ceil(f)

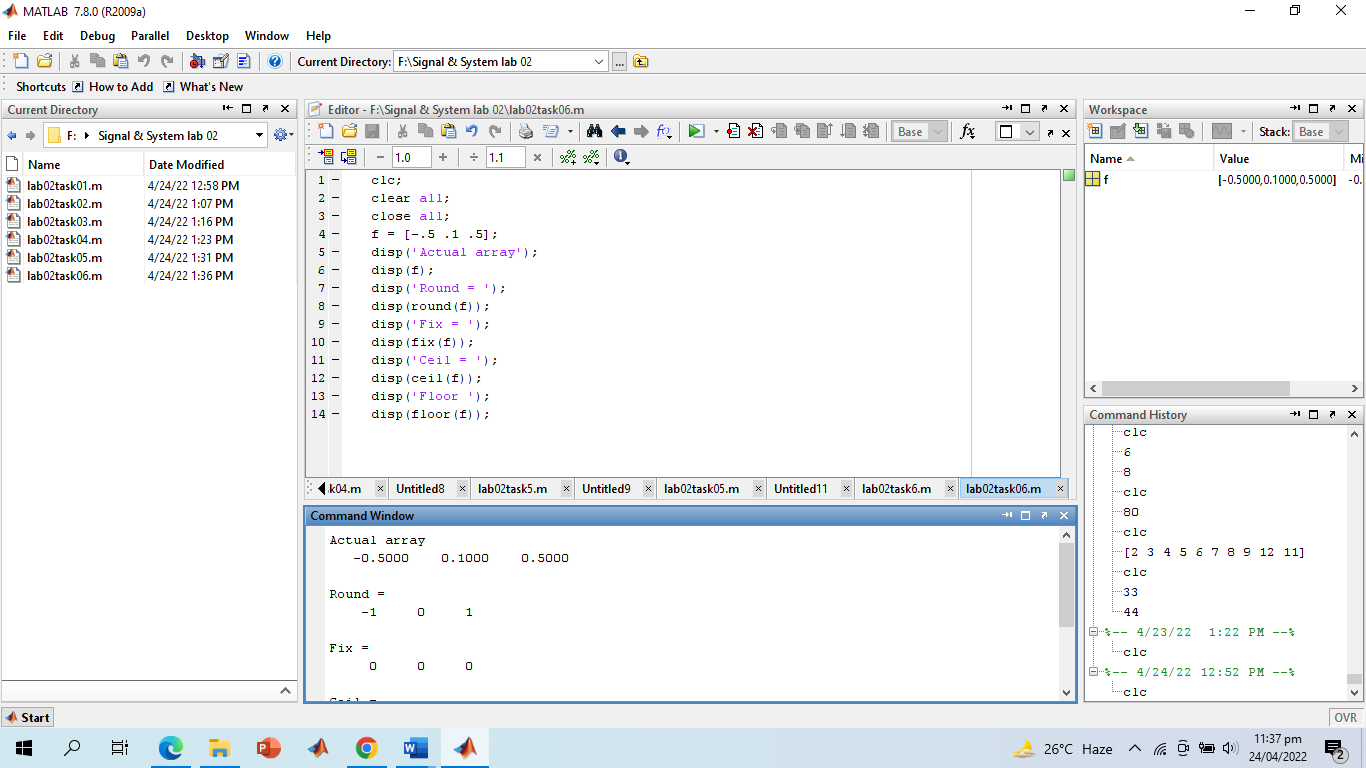
>> floor(f)

**Problem Analysis:**

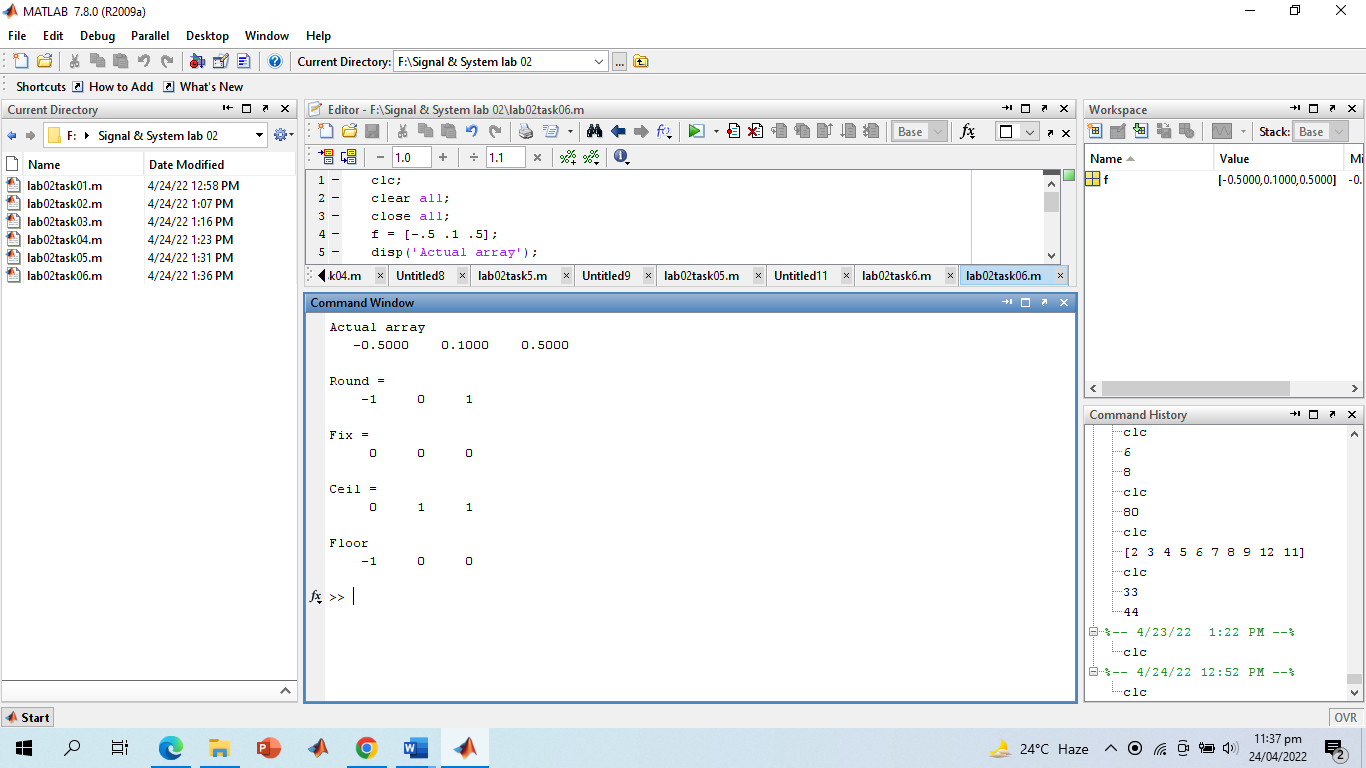
The floor function rounds off the given number to the greatest integer that is, less than or equal to, the input number. Floor(34.34) = 34. Floor(34) = 34. Similarly, the ceil function rounds of the given number to the smallest integer that is, greater than or equal to, the input number.

Ceil(34.25) = 35. Ceil(34) = 34. The round function performs the general rounding. The fix functions floors the inputs number to the closest integer towards zero. Fix(2) = 1 and fix(1) = 0.

**Screenshot of Source Code:**



**Screenshot of Output:**



**Discussion and Conclusion:**

According to general rounding rules, -0.5 equals -1, 0.1 equals 0 and 0.5 equals 1. For fix, the closes integer towards 0 from -0.5, 0.1 and 1 is 0 and therefore all three elements of fix are 0. For ceil the smallest integer greater or equal to -0.5 is 0, and to 0.1 and 0.5, the value is 1. For floor, the greatest integer less than or equal to -0.5 is -1, and to 0.1 and 0.5 the value is 0. We see that the output is consistent with our expectations.

# -------------------------TASK 07--------------------------

Given the following matrix:

A= −3 5

8

4

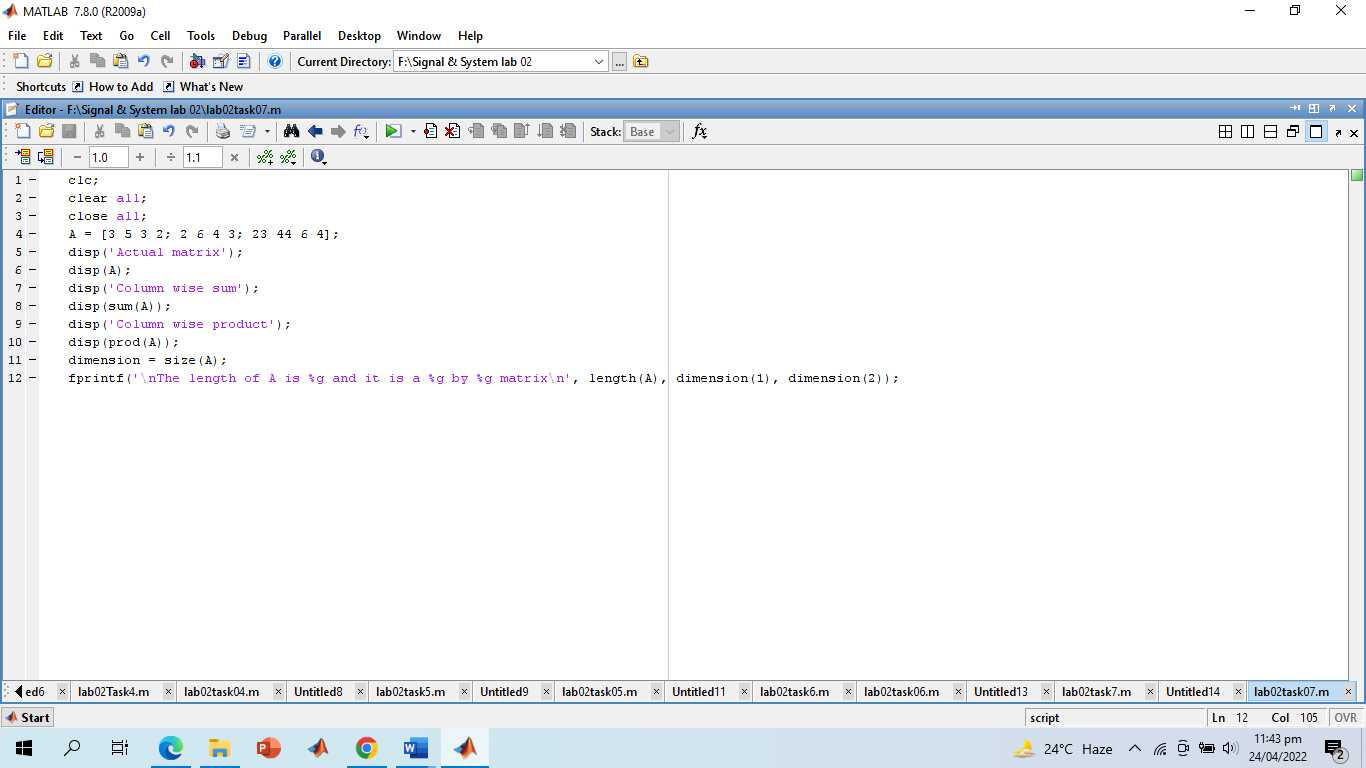
Find the following:

1. Column‐wise sum of all elements of A using sum function; for information about sum function, type help sum in matlab
2. Column‐wise product of all elements of A using prod function; for information about prod function, type help prod in matlab
3. Length of matrix A
4. Size of matrix A

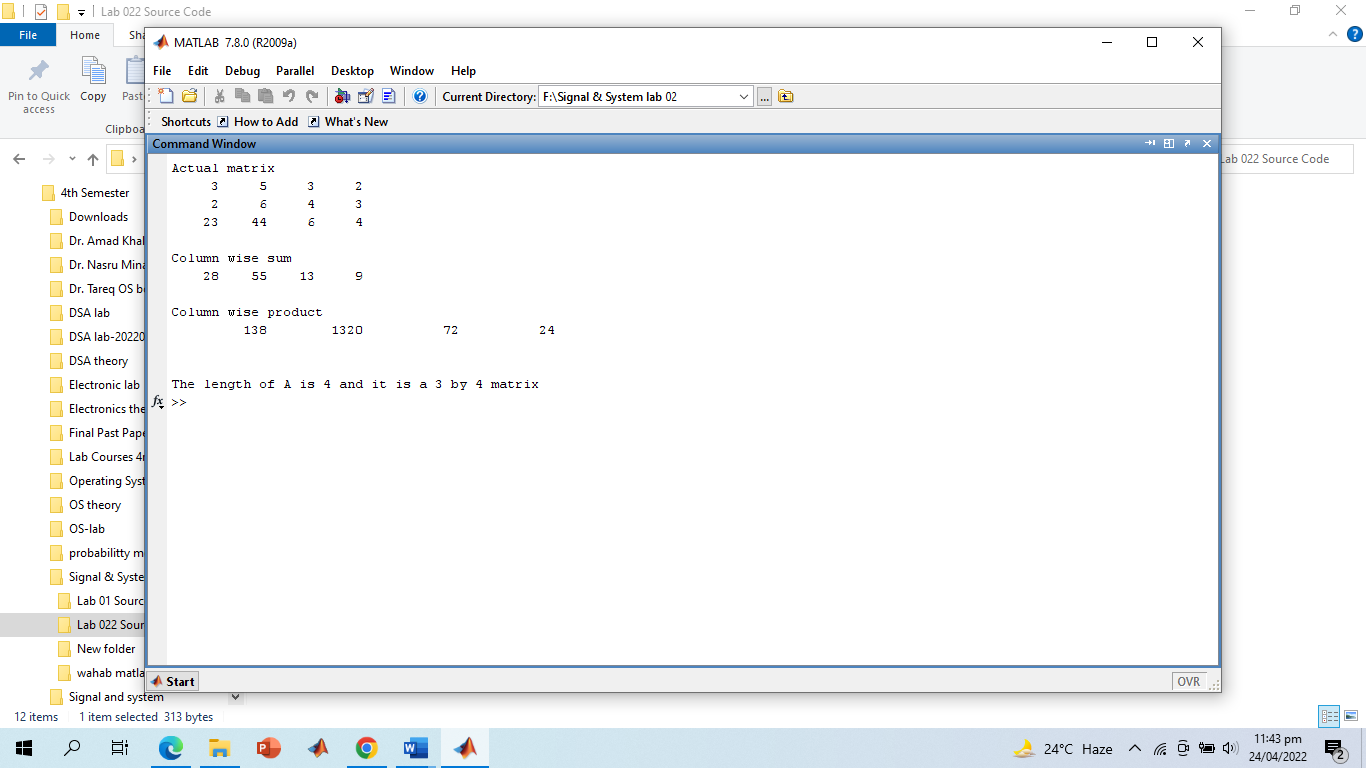
**Problem Analysis:**

All these can be performed using the built-in sum, prod, length and size functions.

**Screenshot of Source Code:**



**Screenshot of Output:**



**Discussion and Conclusion:**

The size functions returns a matrix containing dimensions. In our case, since our matrix was 2 dimensional, the size returned a 2-element matrix.

# -------------------------TASK 08--------------------------

# The end command is used to access the last row or column of a matrix. Use the end command to delete and update the last row and column of the following matrix.

Given the following matrix:

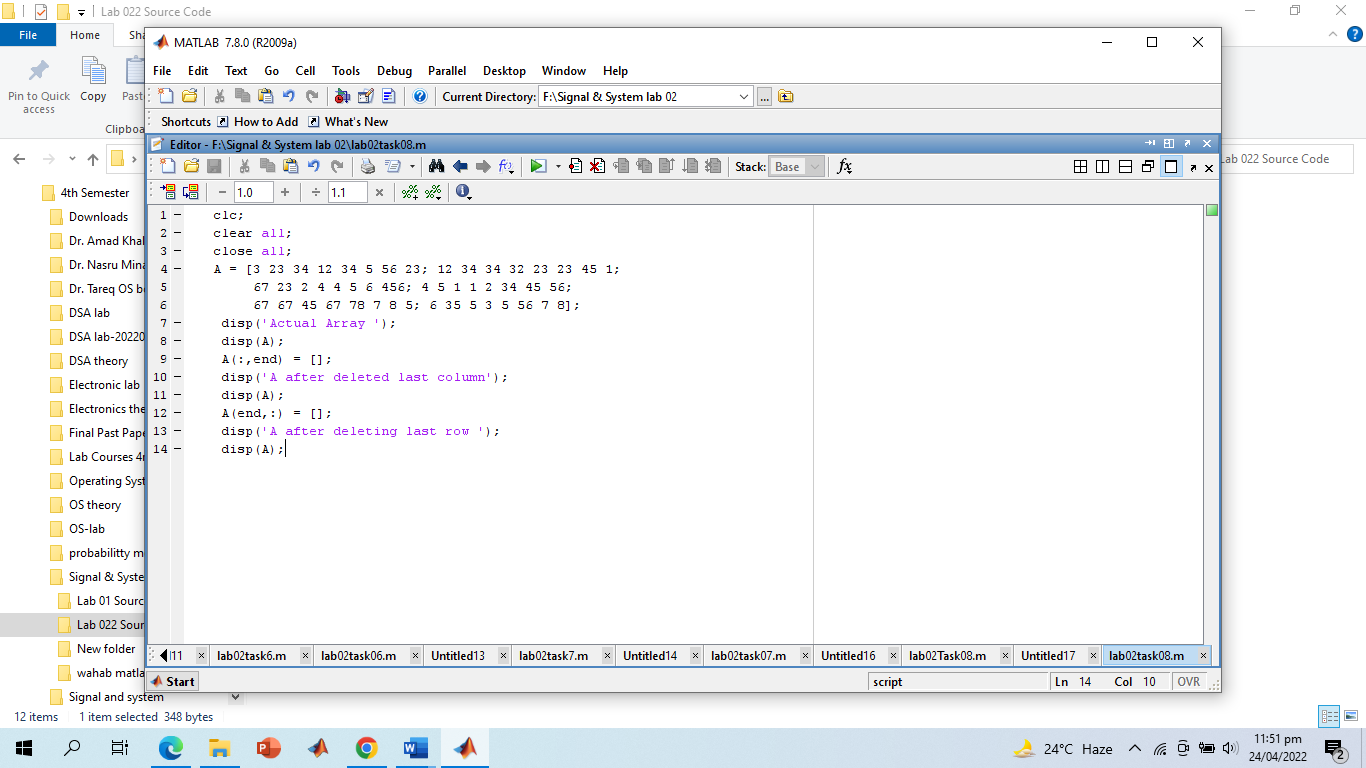
Find the following:

1. Column‐wise sum of all elements of A using sum function; for information about sum function, type help sum in matlab
2. Column‐wise product of all elements of A using prod function; for information about prod function, type help prod in matlab
3. Length of matrix A
4. Size of matrix A

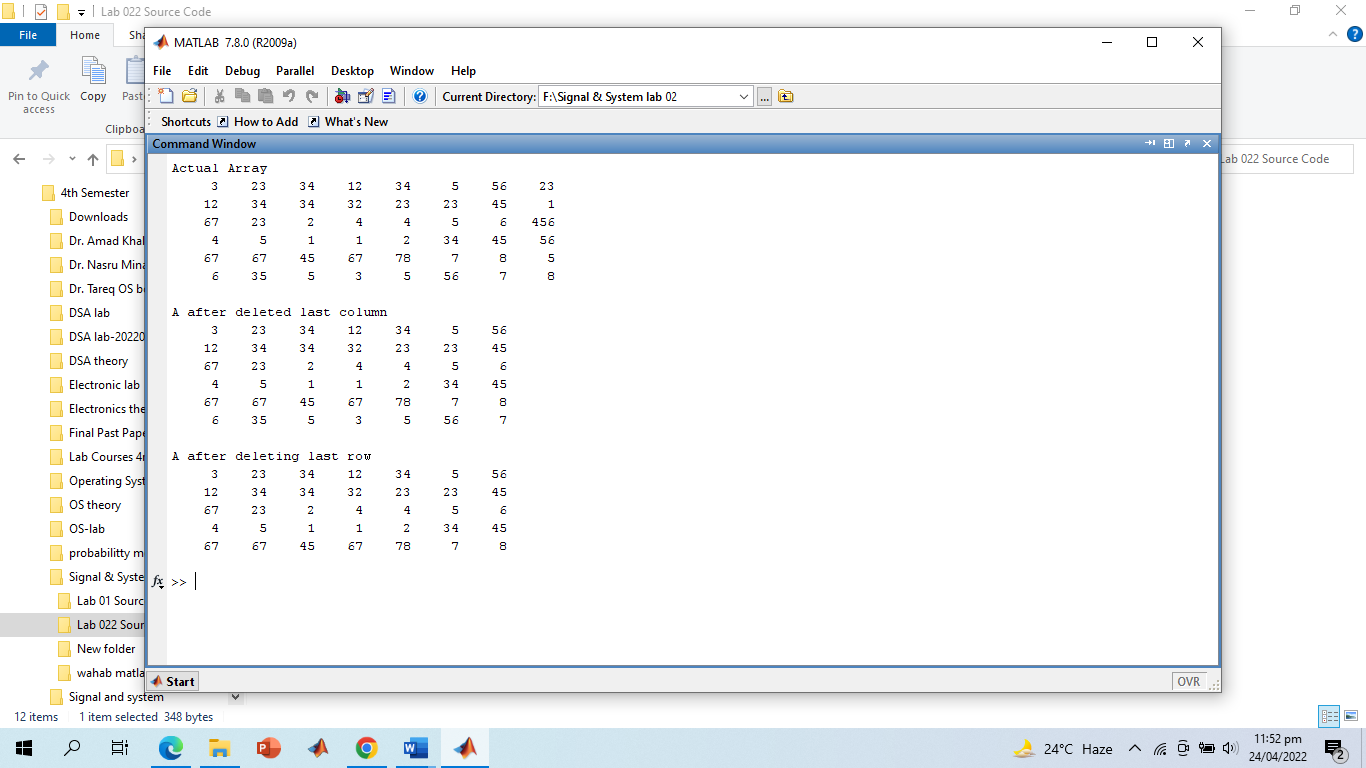
**Problem Analysis:**

If we write, A(;, end) than it will return us all the rows in last column which is the entire last column. Similarly, writing A(end,:) will return all columns in the last row, which is the last entire last row. Setting them to [] will essentially cause them to delete.

**Screenshot of Source Code:**



**Screenshot of Output:**



# -------------------------TASK 09--------------------------

Given the following matrix

Try the following commands in Matlab and comment on them:

1. A(3,end)
2. A(:)
3. A (: , end)
4. Y = linspace(20,100)

(v) Y = linspace(20,100,50)

**Problem Analysis:**

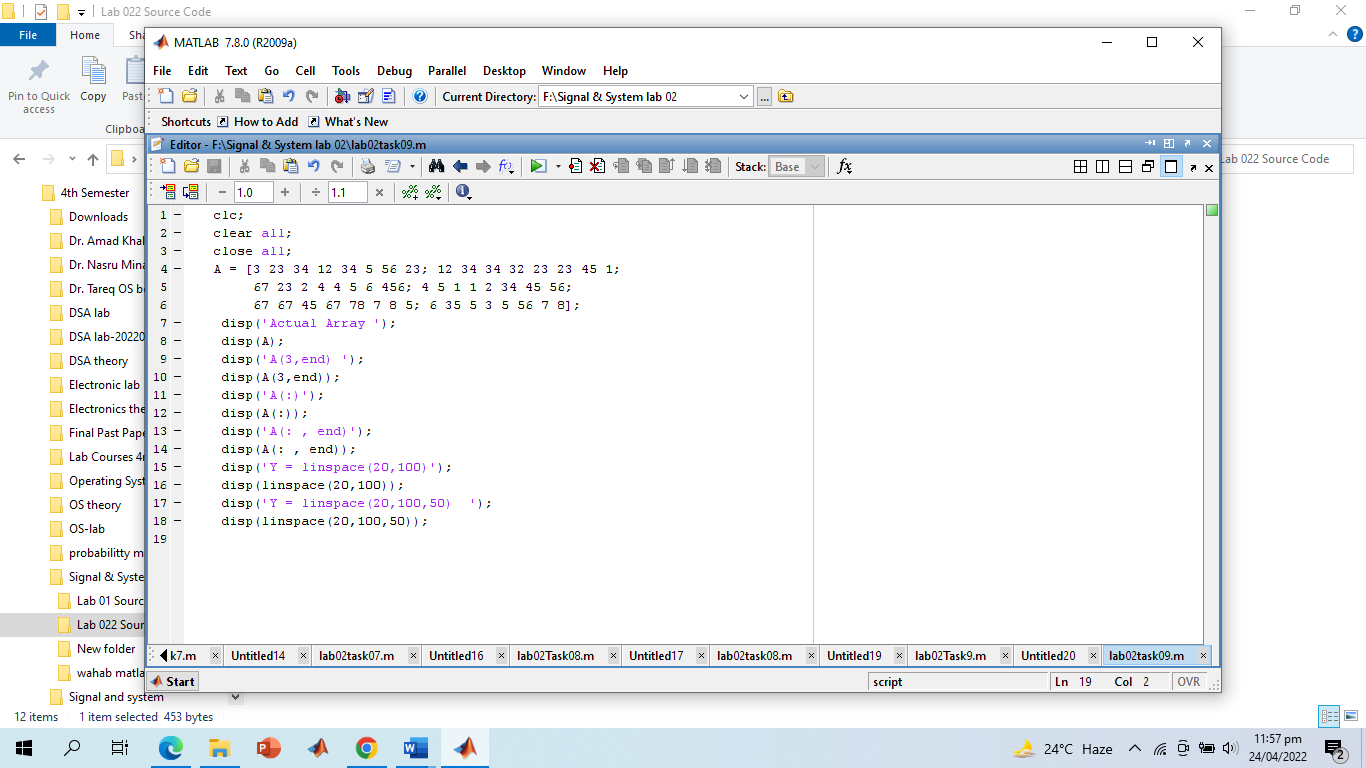
A(3, end) will return element at row 3 of the last column.

A(:) will columnwise return all the columns

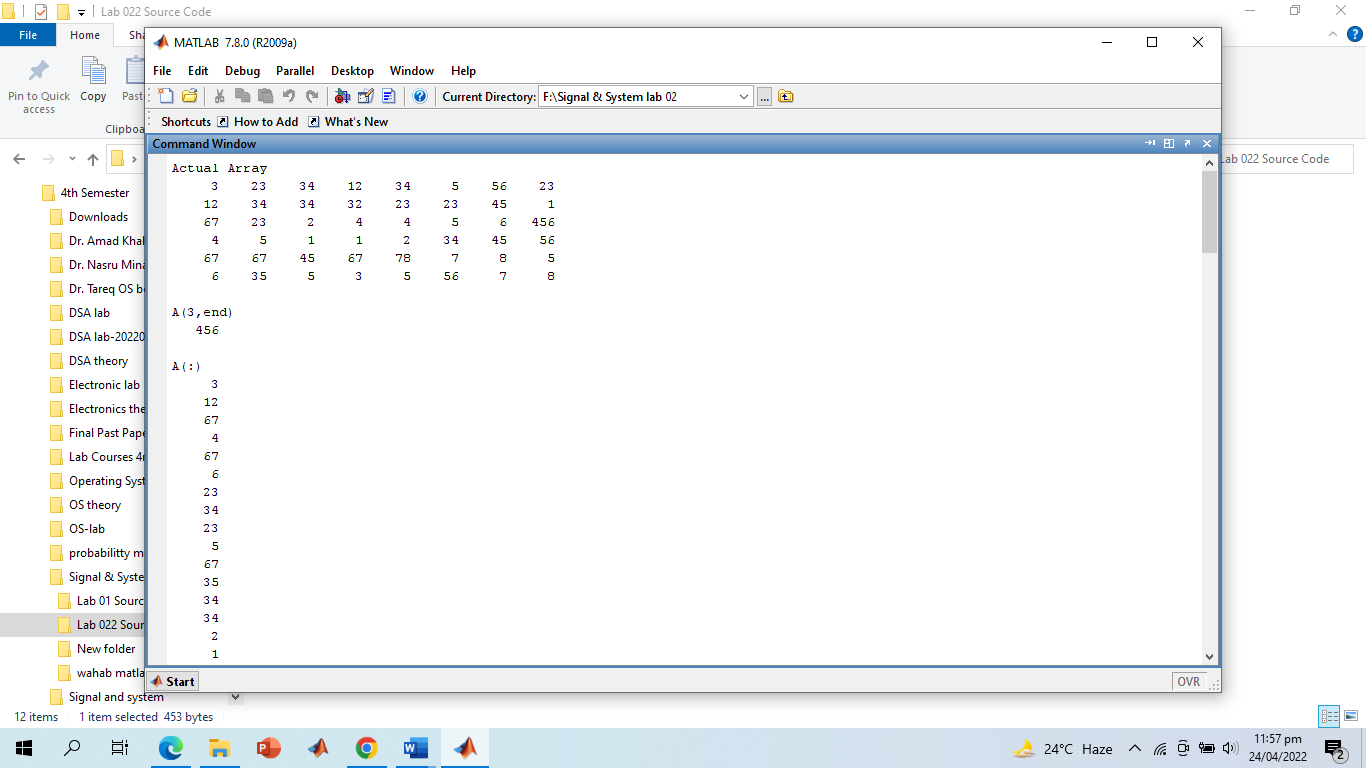
A(:,end) will return all rows of last column which is the entire last column.

Linespace (a, b) returns a vector of 100 linearly spaced points between a and b and linespace(a,x,b) returns a vector of x linearly spaced points between a and b.

**Screenshot of Output:**



**Screenshot of Output:**



# -------------------------TASK 10--------------------------

Use the inverse (inv(A)) function to find the inverse of A for finding the unknowns for Linear equation.

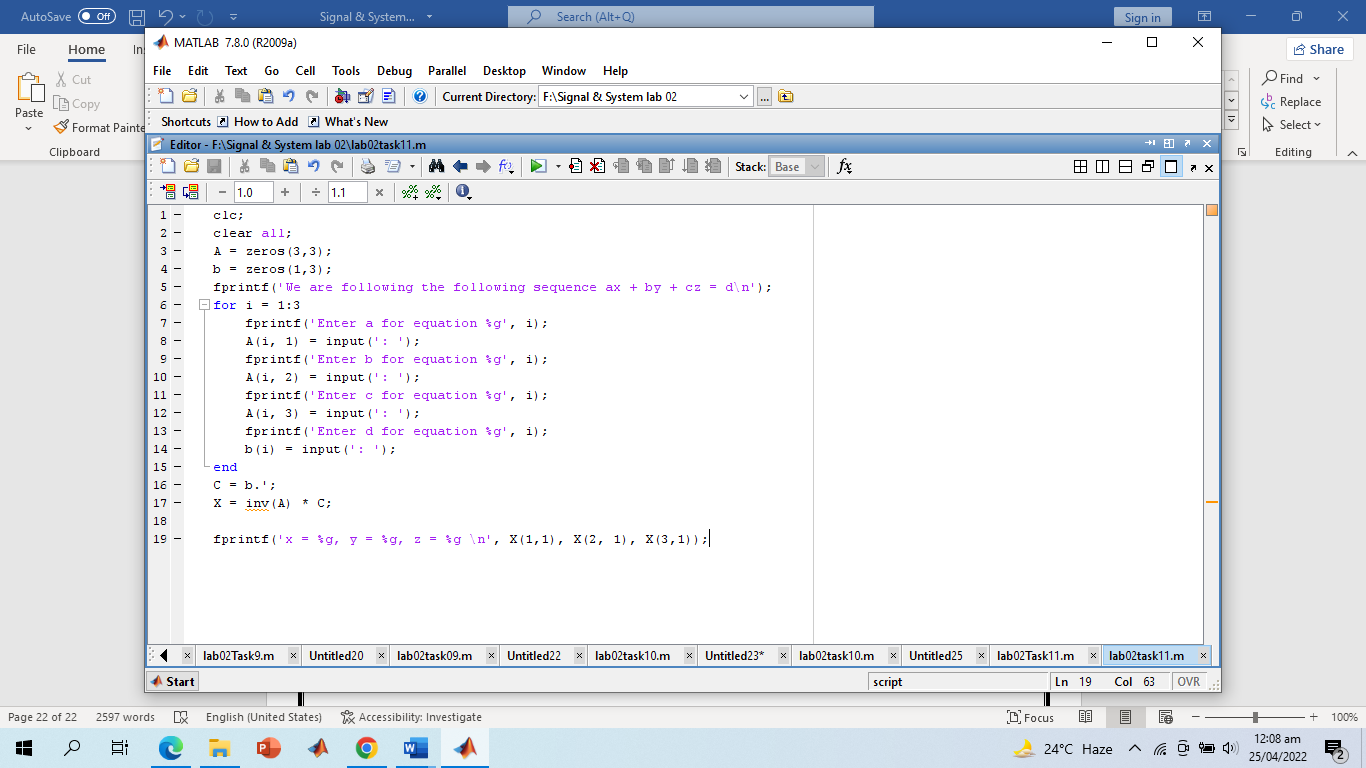
# Screenshot of Source Code and Output:

# 

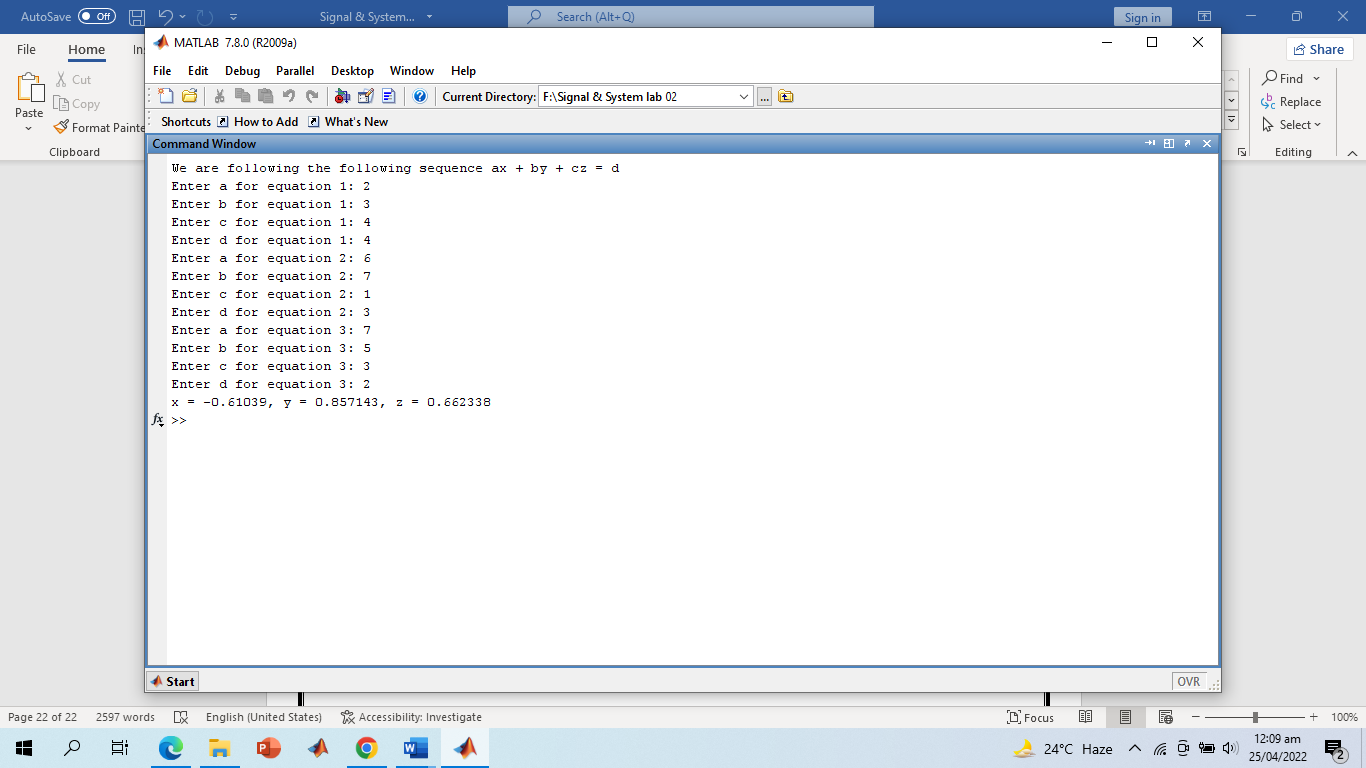
# -------------------------TASK 11--------------------------

Solve 10 by taking input from user.

**Screenshot of Source Code and Output:**



**Screenshot of Output:**



**-------------------------THE END-------------------------**