INTRODUCTION TO MATRICES

LAB # 02

Spring 2023

CSE301L Signals & Systems Lab

Submitted by: **Ali Asghar**

Registration No. : **21PWCSE2059**

Class Section: **C**

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

Submitted to:

Engr. Sumayyea Salahuddin

March 16, 2023

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

Lab Objective(s):

Objectives of this Lab are;

- Built in Matrix Functions
- Indexing Matrices
- Sub Matrices
- Matrix element level operations
- Round Floating Point numbers to Integers

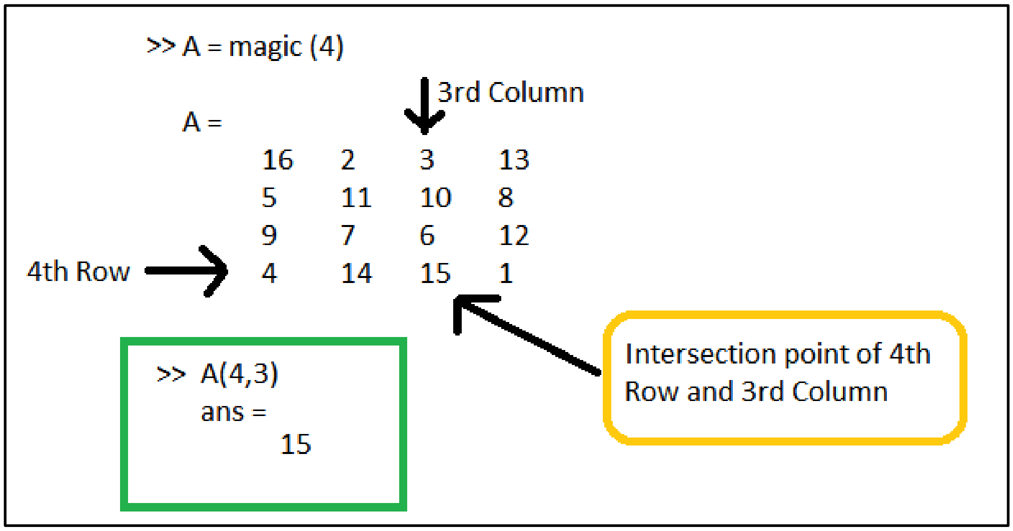
INTRODUCTION:

Built-in Matrix Functions:

In many branches of mathematics and computer science, matrices constitute an essential type of data structure. For performing various operations on matrices, such as matrix addition, multiplication, transposition, and determinant, several computer languages have built-in functions. These operations are frequently accuracy- and performance-optimized, making them perfect for use in engineering and science.

Indexing Matrices:

Matrix indexing refers to the process of accessing specific elements of a matrix. In many programming languages, matrices are represented as arrays, and indexing is done using row and column indices. For example, to access the element at row i and column j of a matrix A, you can use the notation A(i,j).



```
>> A = magic (4)

A =
        16    2    3    13
        5     11   10   8
        9     7    6    12
        4     14   15   1
```

```
>> A(4,3)
   ans =
         15
```

Figure 2-1: Accessing Elements of Matrix A by row and column subscript in MATLAB

Sub Matrices:

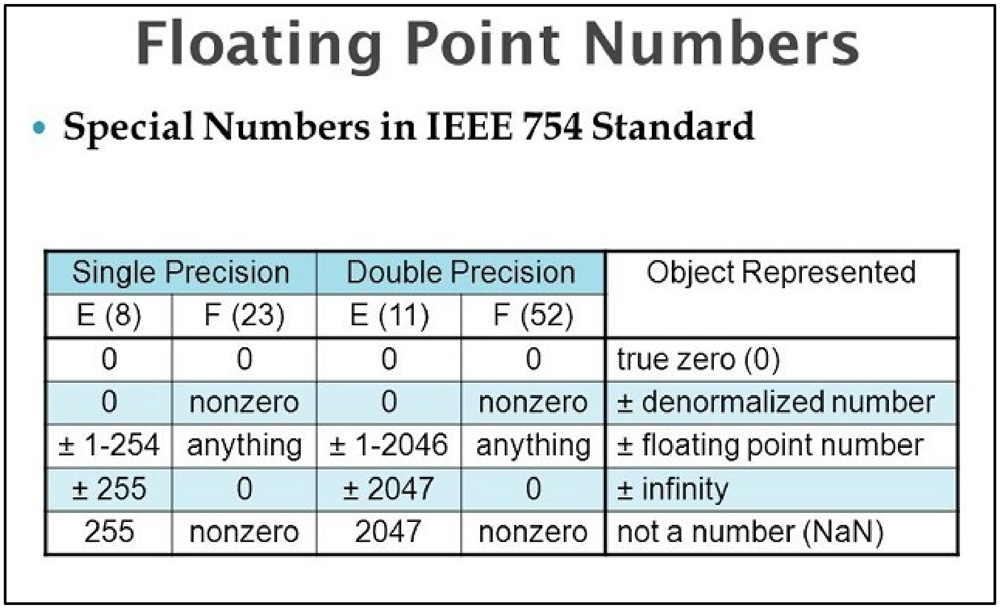
A submatrix is a subset of a matrix's constituent elements that creates a smaller matrix. In linear algebra and other branches of mathematics like graph theory and optimization, submatrices are frequently employed. You can give the row and column indices of the components you want to include in order to extract a submatrix from a bigger matrix.

Matrix element level operations:

Matrix element-level operations are operations that are applied to individual elements of a matrix. Examples of such operations include addition, subtraction, multiplication, division, exponentiation, and more. Element-level operations are important in many scientific and engineering applications, such as image processing, signal analysis, and machine learning.

Floating-Point numbers:

Real numbers are represented by floating-point numbers in several computer languages. To round floating-point numbers to integers, though, may occasionally be necessary for presentation or other reasons. Using built-in operations like round(), floor(), and ceiling (). While the floor() function rounds down to the nearest integer and the ceil() function rounds up to the nearest integer, round() rounds a floating-point number to the nearest integer.



Floating Point Numbers

- Special Numbers in IEEE 754 Standard

Single Precision		Double Precision		Object Represented
E (8)	F (23)	E (11)	F (52)	
0	0	0	0	true zero (0)
0	nonzero	0	nonzero	± denormalized number
± 1-254	anything	± 1-2046	anything	± floating point number
± 255	0	± 2047	0	± infinity
255	nonzero	2047	nonzero	not a number (NaN)

Figure 3-1:Floating-Point numbers: Special Numbers in IEEE 754 Standard

RESULTS AND EXPLANATION:

Task 1:

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.

$$A = \begin{bmatrix} 3 & 6 & 9 \\ 1 & 4 & 8 \\ 2 & 8 & 7 \end{bmatrix}$$

Problem Analysis:

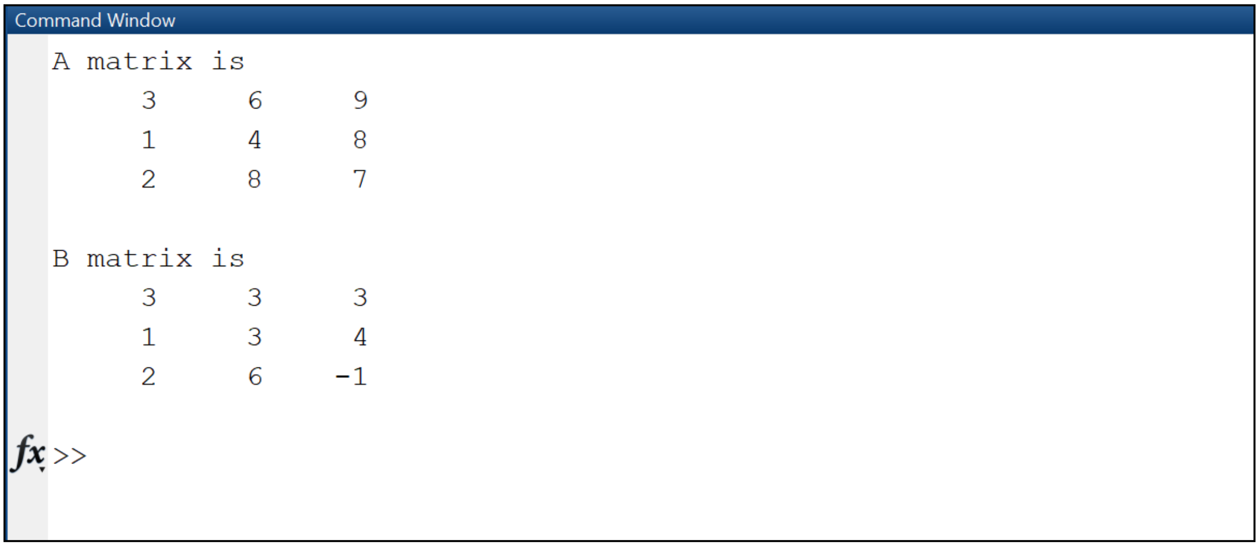
With MATLAB, manipulating matrices is simple. Here, a matrix will be replaced with built-in formulas in MATLAB.

Algorithm:

- Write the matrix A.
- Produce a zero matrix B;
- Switch out row one of B for row one of A.
- Keep the difference between the first and second columns in c2.
- Keep the third-and-second-column difference in c3.
- Change the second and third columns of B to c2 and c3, respectively.
- Display A
- Display B

Code:

Output / Graphs / Plots / Results:



```
A matrix is
     3     6     9
     1     4     8
     2     8     7

B matrix is
     3     3     3
     1     3     4
     2     6    -1

>>
```

Discussion and Conclusion:

With MATLAB, we can quickly replace columns and rows in matrices.

Task 2:

Generate two 5000 samples random discrete time signals (1 dimensional) using rand () function i.e. rand (1, 5000). Write a program to add two signals together using simple vector addition.

Problem Analysis:

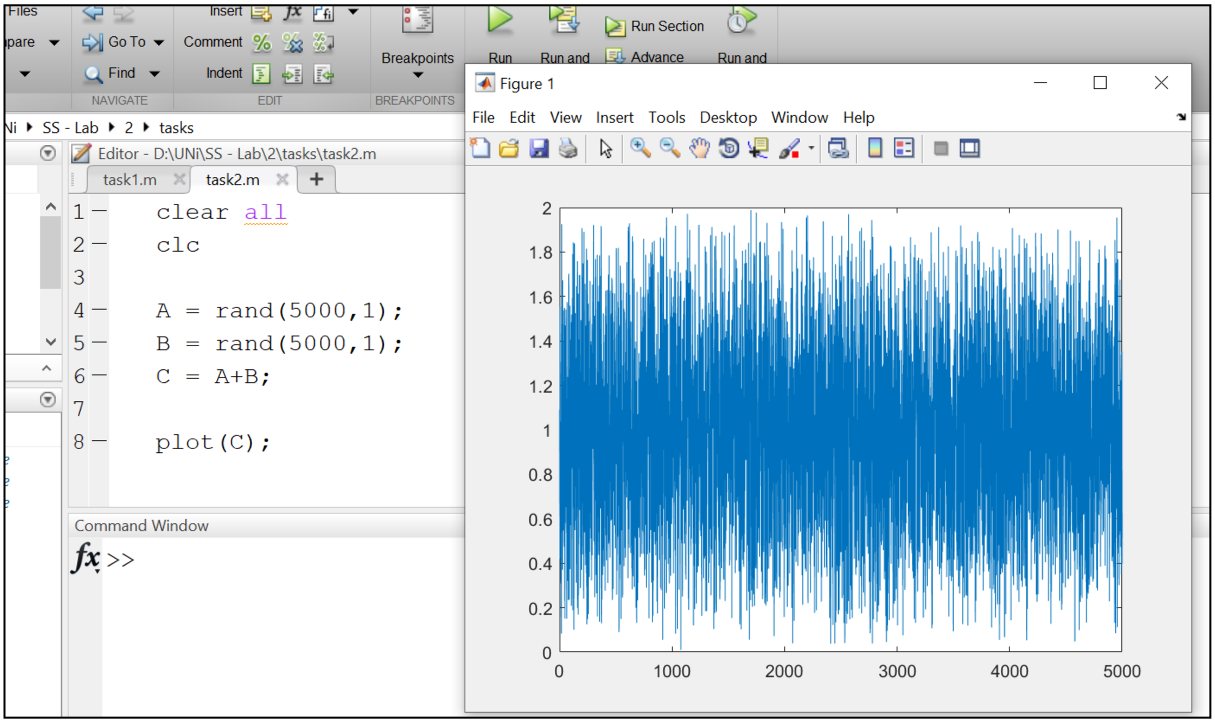
MATLAB is able to work with vectors and signals as well. Here, we use rand to create signals, which we then put in a vector for vector addition.

Algorithm:

- Create a random signal of 1-5000 discrete values and store it in A
- Create a random signal of 1-5000 discrete values and store it in B
- Add A and B and store it in C
- Display C

Code:

Output / Graphs / Plots / Results:



Editor - D:\UNi\SS - Lab\2\tasks\task2.m

```
clear all
clc

A = rand(5000,1);
B = rand(5000,1);
C = A+B;

plot(C);
```

Discussion and Conclusion:

MATLAB is particularly effective at handling both vectors and temporal signals. We can quickly convert discrete time domain values to vectors by applying a few straightforward formulas and routines.

Task 3:

Using colon notation, generate the following sequence:
-120, -116, -112, . . ., -4, 0, 4, 8, . . ., 112, 116, 120

Problem Analysis:

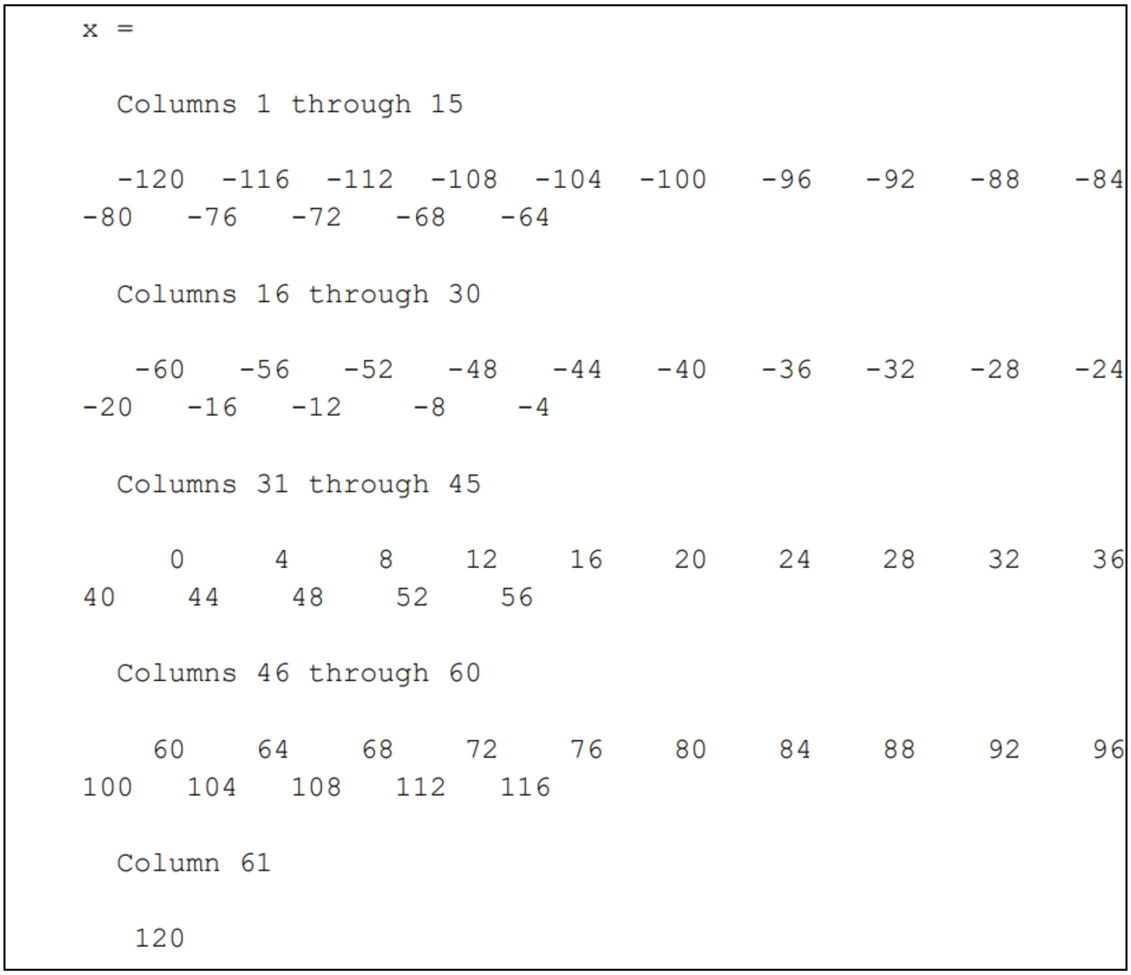
In addition to signal synthesis, MATLAB makes it simple to create random and fixed sequences.

Algorithm:

- Make a sequence x which ranges from -120 to 120 with and increment of 4
- Display the sequence

Code:

Output / Graphs / Plots / Results:



```
x =

  Columns 1 through 15

  -120  -116  -112  -108  -104  -100   -96   -92   -88   -84
-80   -76   -72   -68   -64

  Columns 16 through 30

   -60   -56   -52   -48   -44   -40   -36   -32   -28   -24
-20   -16   -12    -8    -4

  Columns 31 through 45

     0     4     8    12    16    20    24    28    32    36
40    44    48    52    56

  Columns 46 through 60

    60    64    68    72    76    80    84    88    92    96
100   104   108   112   116

  Column 61

   120
```

Discussion and Conclusion:

Generating a sequence in computer programming takes much more time and effort than in MATLAB. As a result, MATLAB has also made the production of sequences more time efficient.

Task 4:

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:

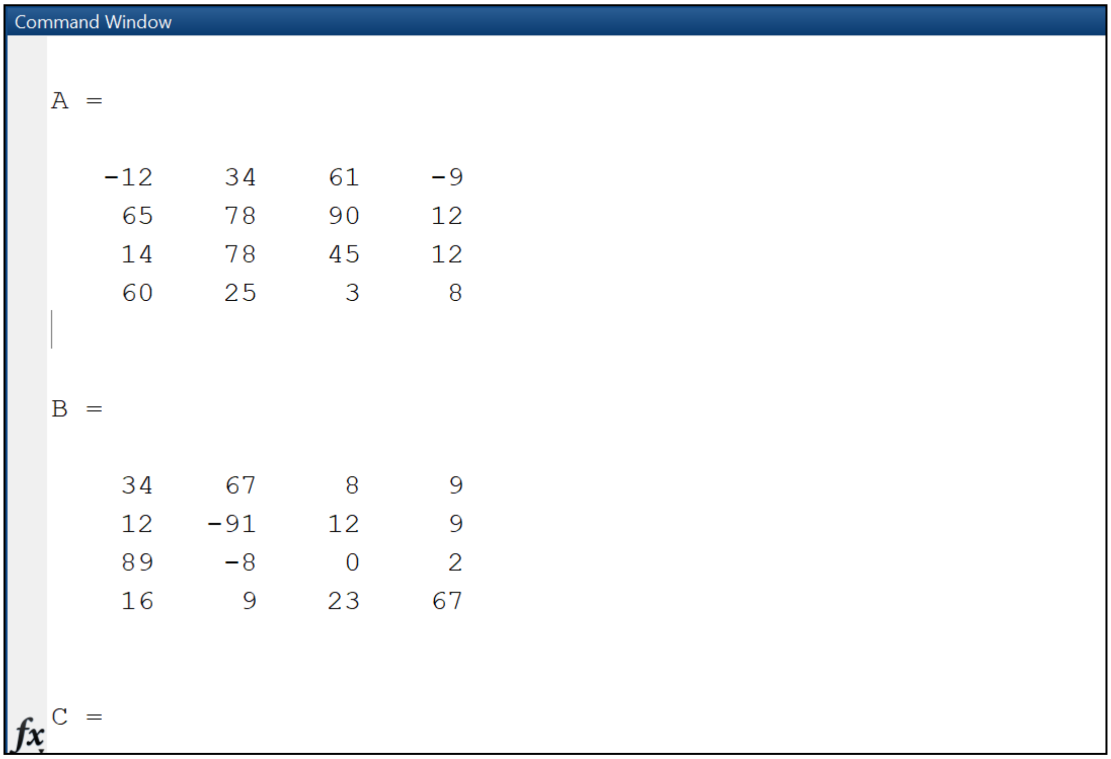
The use of matrices in mathematical computations is crucial. Matrix calculations are just as efficient at solving problems as linear and pure computations. Here, we demonstrate how MATLAB facilitates calculations and matrix computation.

Algorithm:

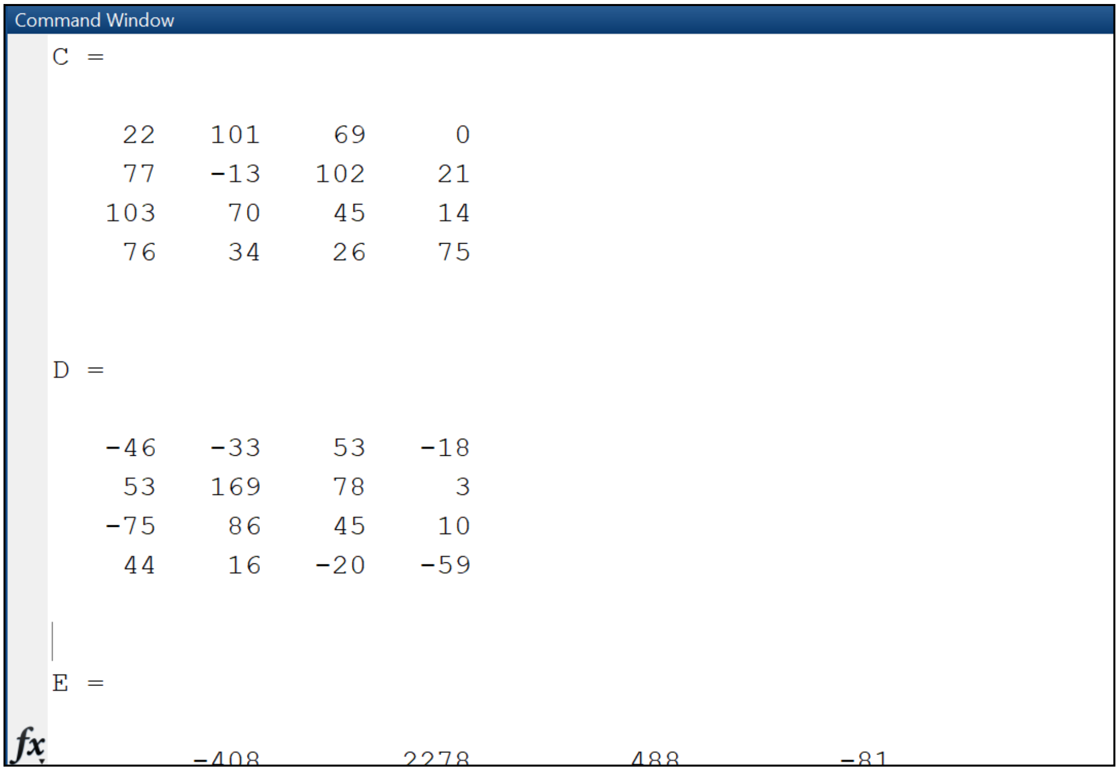
- Write matrices/arrays A and B
- Sum A and B and store it in C
- Subtract B from A and store it in D
- Multiply A with B using .* operator and store it in E
- Divide A by B using ./ operator and store it in F
- Use .^ operator to perform exponentiation of A and B and store it in G
- Take sin(A) and store it in H
- Take sqrt(B) and store their product in I
- Multiply H and I and then store it in J
- Display A,B,C,D,E,F,G,H,J

Code:

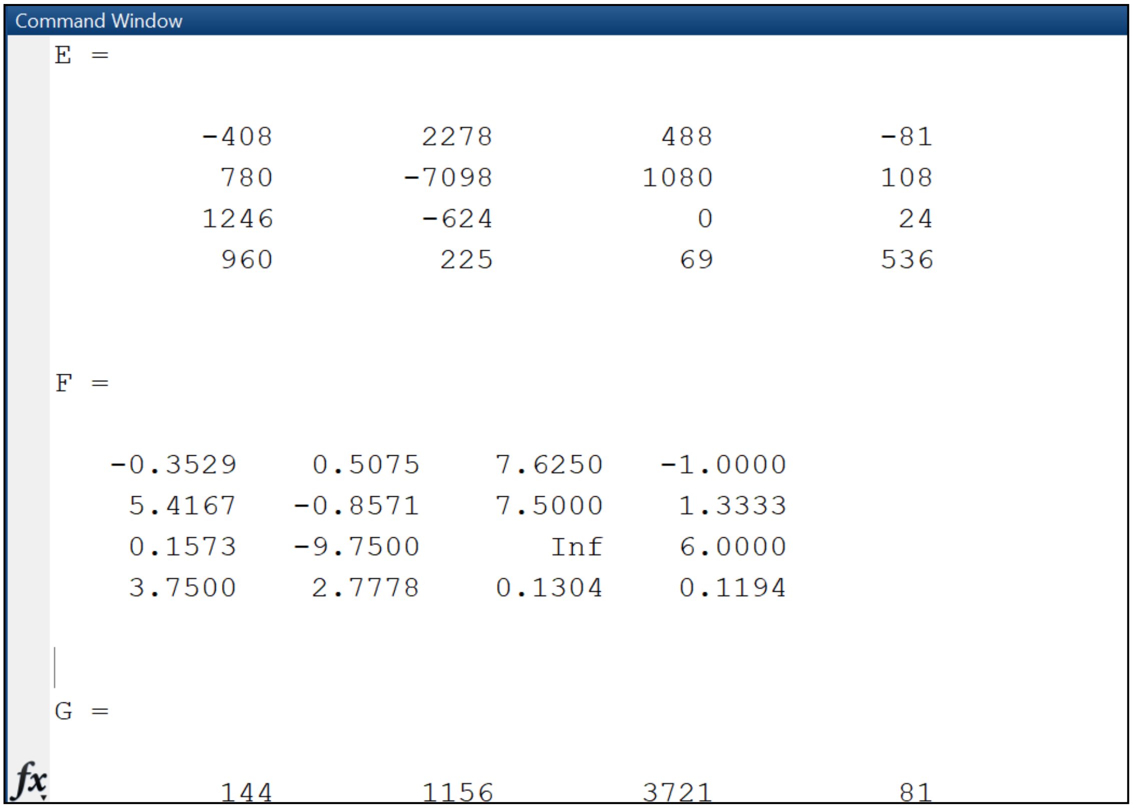
Output / Graphs / Plots / Results:



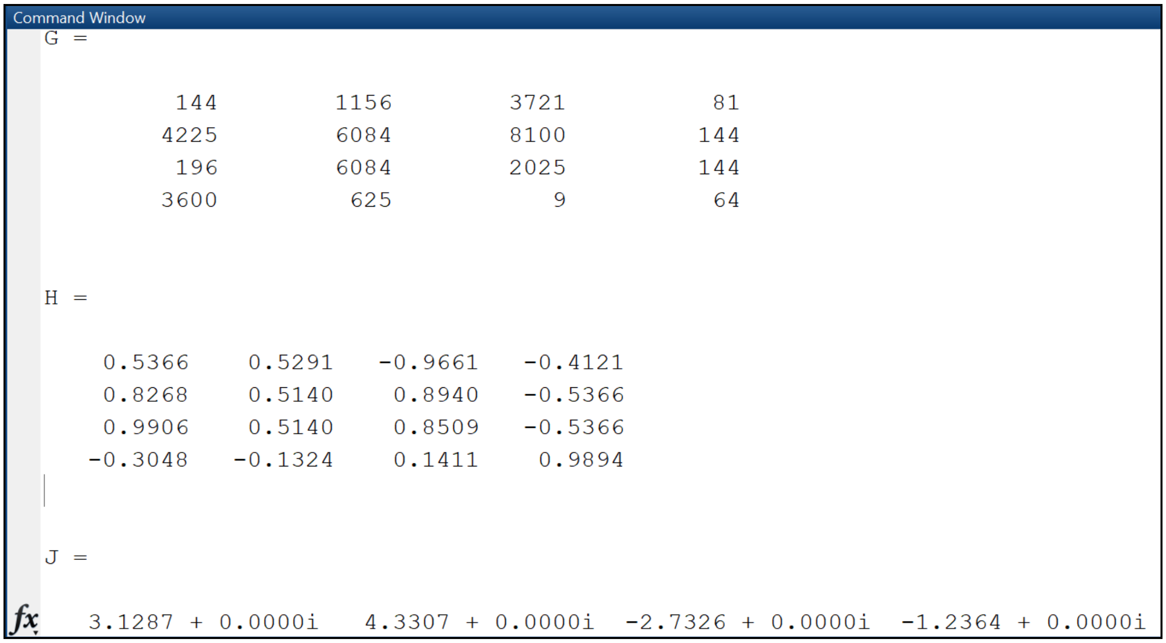
```
A =

   -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


C =
```



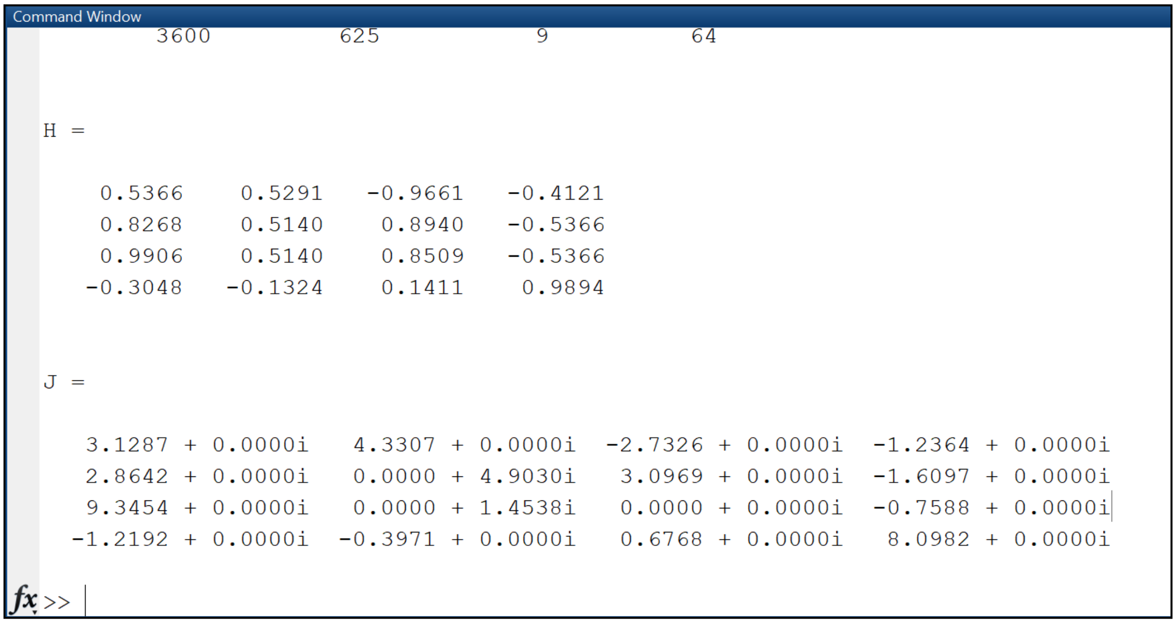
```
C =

    22   101    69     0
    77   -13   102    21
   103    70    45    14
    76    34    26    75


D =

   -46   -33    53   -18
    53   169    78     3
   -75    86    45    10
    44    16   -20   -59


E =

        -408         2278          488          -81
```



```
E =

        -408         2278          488          -81
         780        -7098         1080          108
        1246         -624            0           24
         960          225           69          536


F =

   -0.3529    0.5075    7.6250   -1.0000
    5.4167   -0.8571    7.5000    1.3333
    0.1573   -9.7500       Inf    6.0000
    3.7500    2.7778    0.1304    0.1194


G =

         144         1156         3721           81
```



```
G =

         144        1156        3721          81
        4225        6084        8100         144
         196        6084        2025         144
        3600         625           9          64


H =

    0.5366    0.5291   -0.9661   -0.4121
    0.8268    0.5140    0.8940   -0.5366
    0.9906    0.5140    0.8509   -0.5366
   -0.3048   -0.1324    0.1411    0.9894


J =

   3.1287 + 0.0000i   4.3307 + 0.0000i  -2.7326 + 0.0000i  -1.2364 + 0.0000i
```



```
        3600         625           9          64


H =

    0.5366    0.5291   -0.9661   -0.4121
    0.8268    0.5140    0.8940   -0.5366
    0.9906    0.5140    0.8509   -0.5366
   -0.3048   -0.1324    0.1411    0.9894


J =

   3.1287 + 0.0000i   4.3307 + 0.0000i  -2.7326 + 0.0000i  -1.2364 + 0.0000i
   2.8642 + 0.0000i   0.0000 + 4.9030i   3.0969 + 0.0000i  -1.6097 + 0.0000i
   9.3454 + 0.0000i   0.0000 + 1.4538i   0.0000 + 0.0000i  -0.7588 + 0.0000i
  -1.2192 + 0.0000i  -0.3971 + 0.0000i   0.6768 + 0.0000i   8.0982 + 0.0000i

>>
```

Discussion and Conclusion:

General arithmetic operations on matrices are time-consuming and prone to error, making them highly difficult to complete. On the other hand, MATLAB computes matrix operations very precisely while using a great deal less time and energy.

Task 5:

Type the given matrix in matlab:

$$A = \begin{matrix} 3 & 7 & -4 & 12 \\ -5 & 9 & 10 & 2 \\ 6 & 13 & 8 & 11 \\ 15 & 5 & 4 & 1 \end{matrix}$$

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:

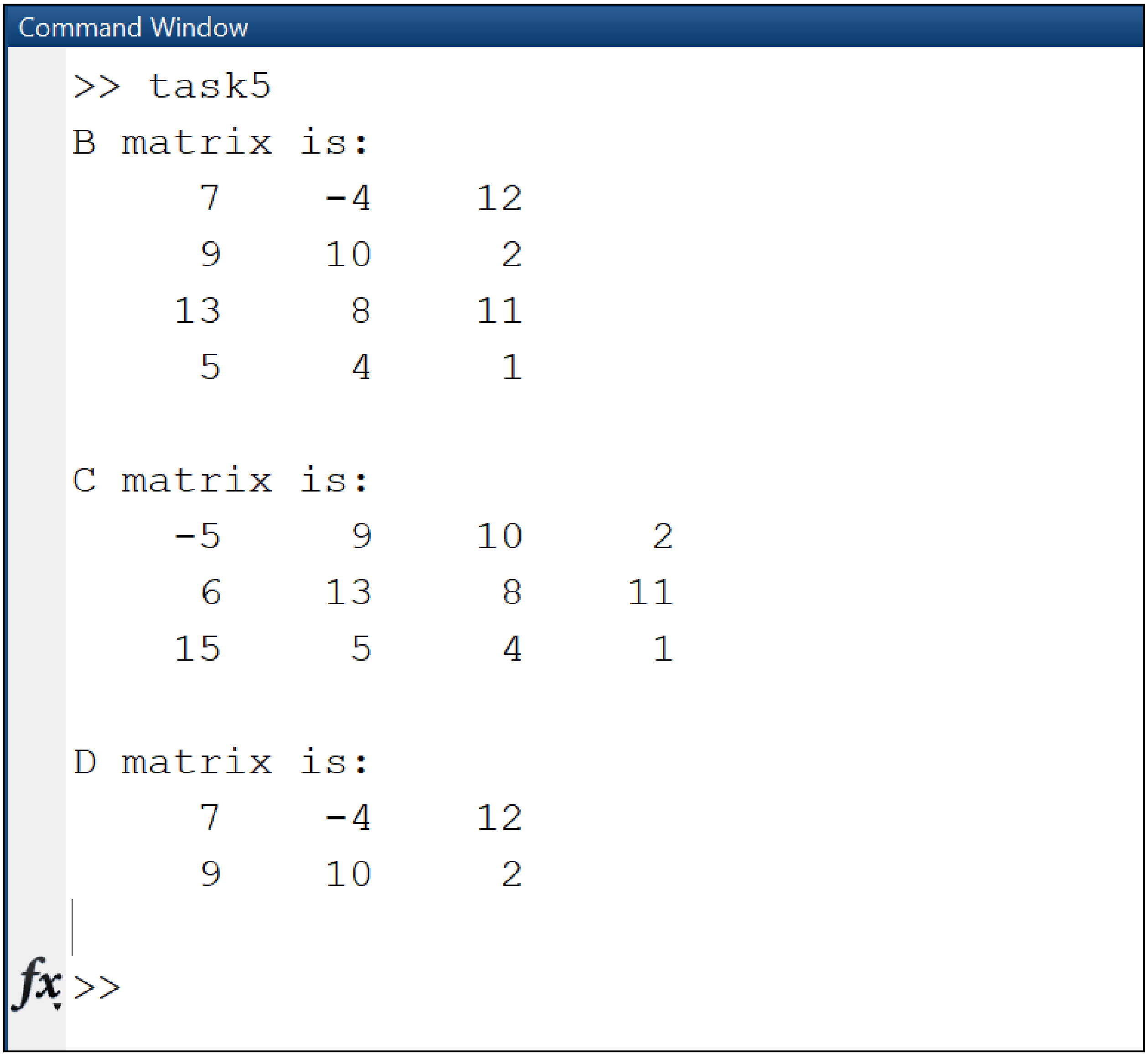
Several mathematical models and issues make use of matrix conversion. Here, we'll use row and column replacement techniques to build new matrices from existing matrices.

Algorithm:

- Write matrix A
- Create a matrix B and store all the elements from second through fourth columns of A
- Create a matrix C and store all the elements from second through fourth rows of A
- Create a matric D and store all elements in the first two rows and the last three columns of A
- Display A,B,C,D

Code:

Output / Graphs / Plots / Results:



```
>> task5
B matrix is:
     7    -4    12
     9    10     2
    13     8    11
     5     4     1

C matrix is:
    -5     9    10     2
     6    13     8    11
    15     5     4     1

D matrix is:
     7    -4    12
     9    10     2

>>
```

Discussion and Conclusion:

On paper, creating new matrices from existing matrices requires a variety of complex strategies that are difficult to recall; however, MATLAB easily handles this operation.

Task 6:

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:

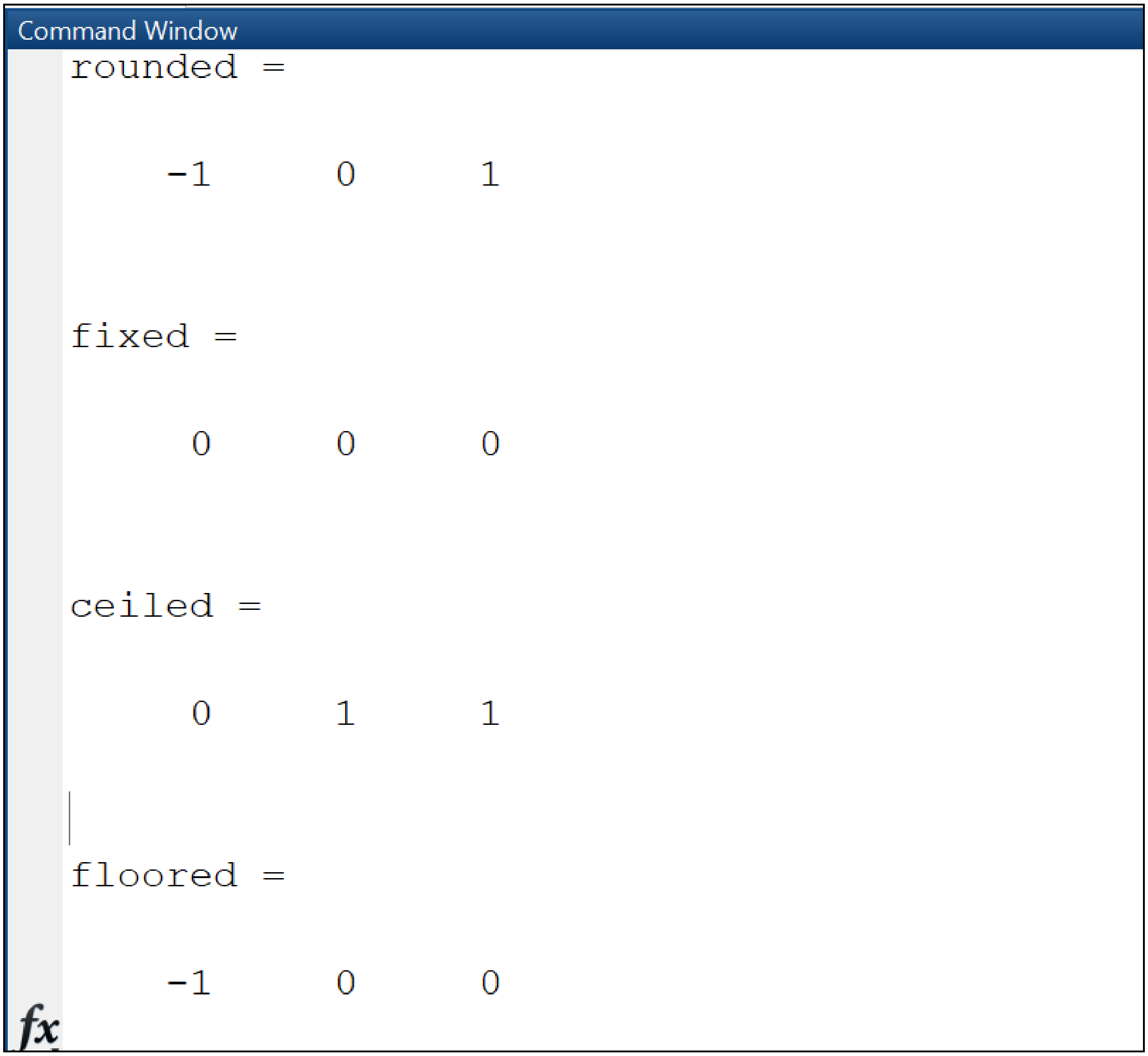
The values must be restricted to a certain number of decimal places for many mathematical calculations. Here, we'll look at how to round floating points to specific decimal places.

Algorithm:

- Write the Matrix f
- Pass matrix into round function
- Pass matrix into fix function
- Pass matrix into ceil function
- Pass matrix into floor function

Code:

Output / Graphs / Plots / Results:



```
rounded =

    -1     0     1


fixed =

     0     0     0


ceiled =

     0     1     1


floored =

    -1     0     0
```

Discussion and Conclusion:

- Each element of X is rounded to the nearest integer by Y = round(X). In the event of a tie, the round function rounds away from zero to the nearest integer with a greater magnitude when an element has a fractional portion of 0.5 (within roundoff error) in decimal form.
- When Y = fix(X), each element of X is rounded to the nearest integer heading towards zero. By eliminating the decimal place from each number, this method effectively converts the numbers in X to integers: Fix behaves similarly to floor for positive values.

- Every element of X is rounded to the nearest integer bigger than or equal to that element by the formula Y = ceil(X). Every member of the duration array t is rounded to the next number of seconds larger than or equal to that element by the formula Y = ceil(t).
- When Y = floor(t), the duration array's elements are rounded to the nearest number of seconds that is less than or equal to that element. Each element of t is rounded to the nearest number of the specified unit of time that is less than or equal to that element by Y = floor(t, unit).

Task 7:

Given the following matrix:

−3 5

A = 4 8

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:

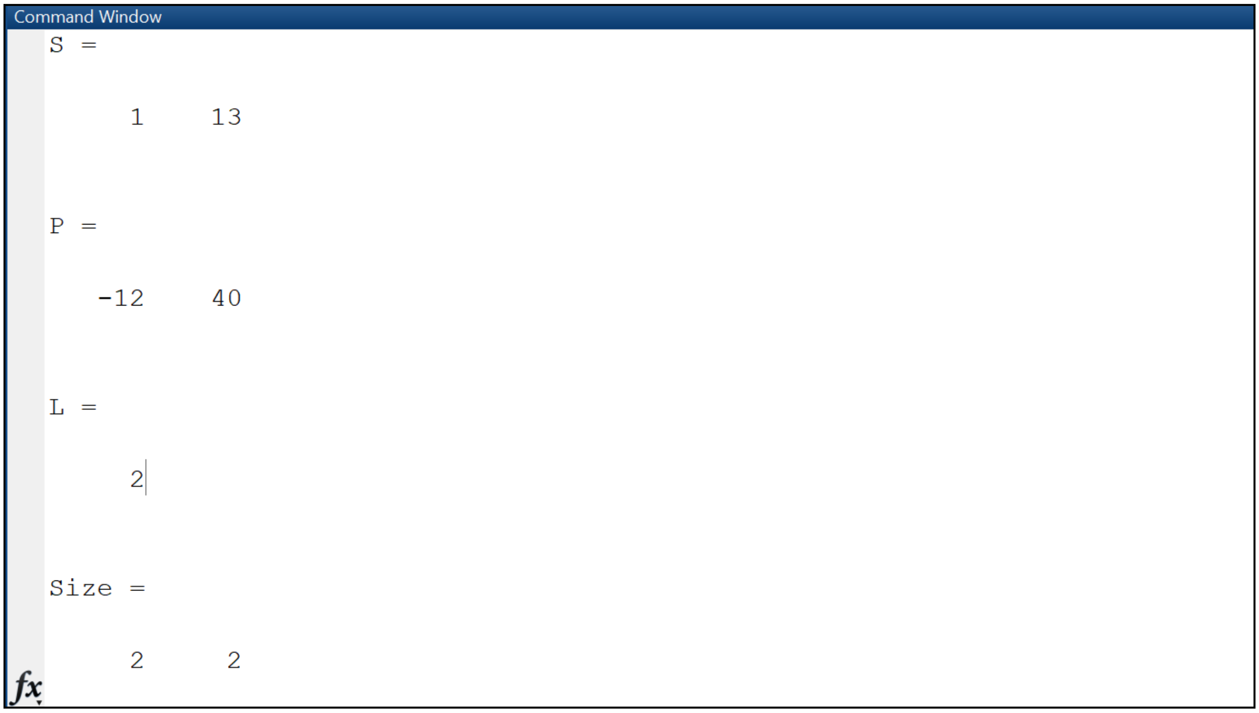
Another very crucial idea in matrices is the concept of column and row wise operations. Here, we'll use MATLAB to compute the sum and product row- and column-wise.

Algorithm:

- Write matrix A
- Pass matrix A into sum function with proper parameters and store it in S for column operation
- Pass matrix A into prod function with proper parameters and store it in P for column operation
- Pass matrix A into length function and store it in L.
- Pass matrix A into size function and store it in Size.
- Display all matrices.

Code:

Output / Graphs / Plots / Results:



```
S =

     1    13


P =

   -12    40


L =

     2


Size =

     2     2
```

Discussion and Conclusion:

With MATLAB, it is simple to calculate a matrix's length and size as well as the sum and product of its rows and columns.

Task 8:

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.
Matrix A = [3 23 34 12 34 5 56 23; 12 34 34 32 23 23 45 1; 67 23 2 4 4 5 6 456; 4 5 1 1 2 34 45 56; 67 67 45 67 78 7 8 5; 6 35 5 3 5 56 7 8]

Problem Analysis:

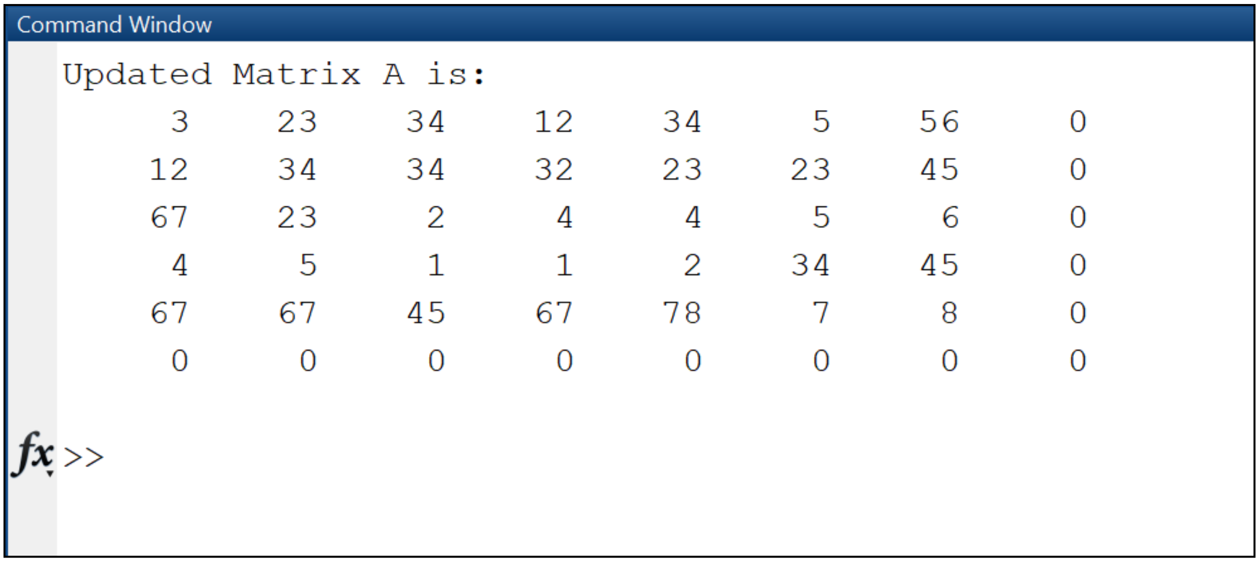
During a variety of matrix operations, we frequently need to replace matrix rows and columns. Here, we attempt to implement it using the MATLAB end command.

Algorithm:

- Create matrix A
- Delete last row
- Delete last column
- Add last column +1 and pass zeros to it
- Add last row +1 and pass zeros to it
- Display A

Code:

Output / Graphs / Plots / Results:



```
Updated Matrix A is:
      3     23     34     12     34      5     56      0
     12     34     34     32     23     23     45      0
     67     23      2      4      4      5      6      0
      4      5      1      1      2     34     45      0
     67     67     45     67     78      7      8      0
      0      0      0      0      0      0      0      0

fx >>
```

Discussion and Conclusion:

Hence, MATLAB makes it simple to replace columns and rows in matrices.

Task 9:

Try the following commands in MatLab and comment on them:
(i) A(3,end)
(ii) A(:)
(iii) A(: , end)
(iv) Y = linspace(20,100)
(v) Y = linspace(20,100,50)

Problem Analysis:

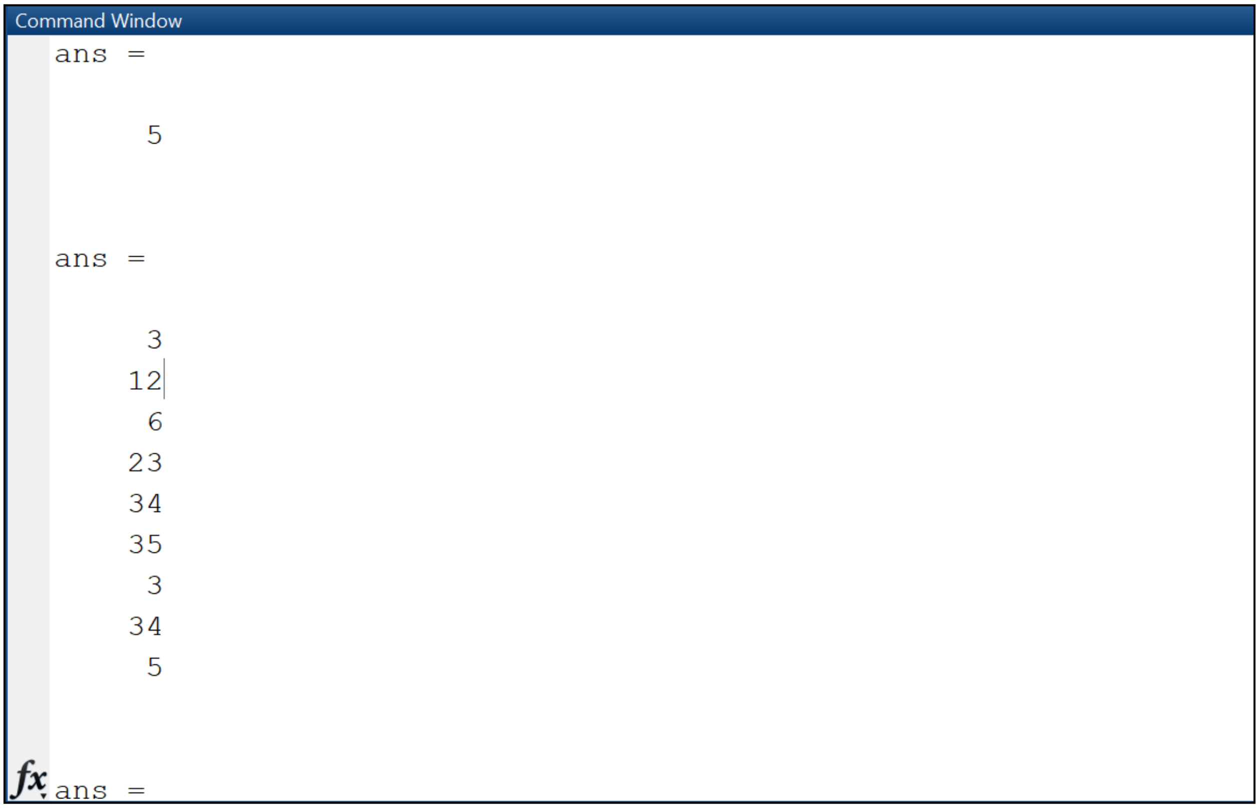
We attempt to carry out selected matrix operations in MATLAB apart from other matrix computations.

Algorithm:

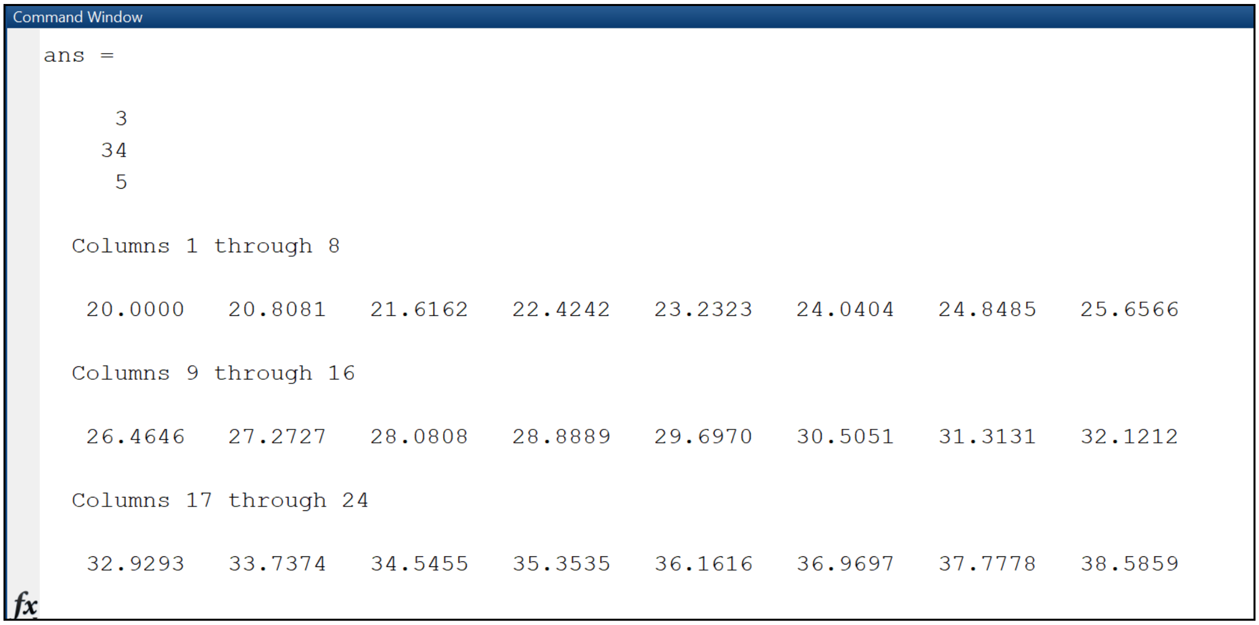
- Define the matrix A.
- Access the value at the intersection of the third row and last column of matrix A using A(3,end).
- Access all elements in matrix A using A(:).
- Access the last column of matrix A using A(:,end).
- Use the linspace function to create a row vector Y with 50 equally spaced values between 20 and 100.
- Display the values in Y using the disp function.
- Use the linspace function again to create a row vector Y with 50 equally spaced values between 20 and 100 and store it in the variable Y.
  Display the values in Y using the disp function.

Code:

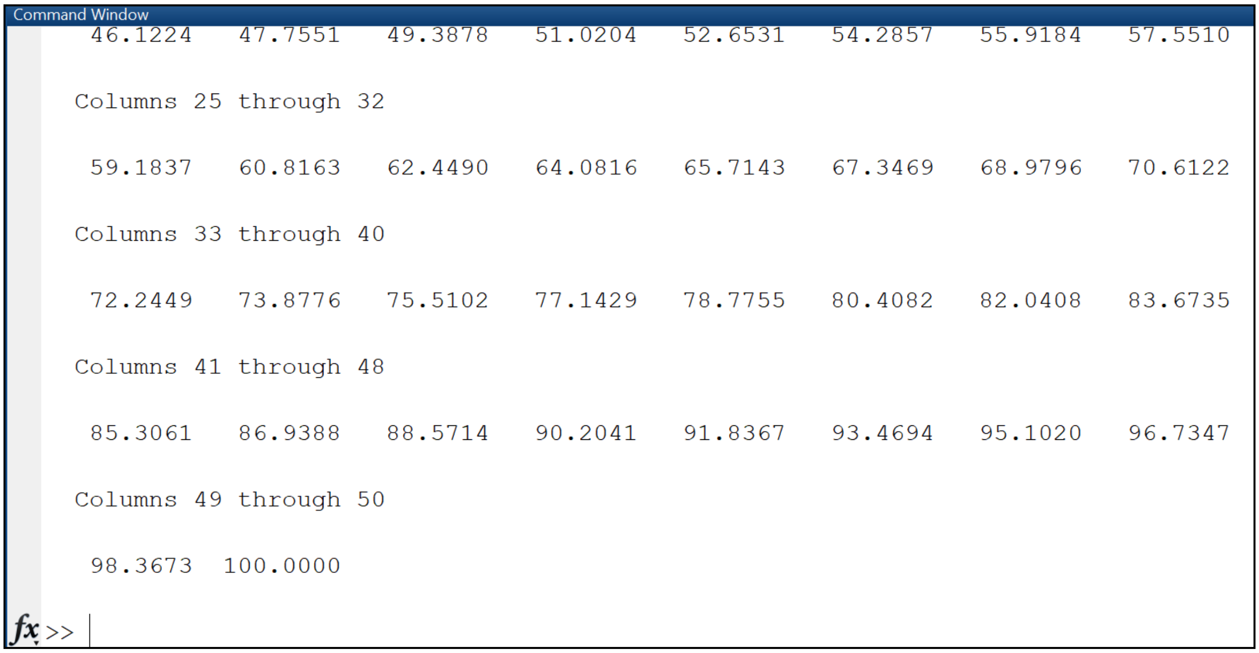
Output / Graphs / Plots / Results:



```
ans =

     5


ans =

     3
    12
     6
    23
    34
    35
     3
    34
     5


ans =
```



```
ans =

     3
    34
     5

  Columns 1 through 8

   20.0000   20.8081   21.6162   22.4242   23.2323   24.0404   24.8485   25.6566

  Columns 9 through 16

   26.4646   27.2727   28.0808   28.8889   29.6970   30.5051   31.3131   32.1212

  Columns 17 through 24

   32.9293   33.7374   34.5455   35.3535   36.1616   36.9697   37.7778   38.5859
```



```
   46.1224   47.7551   49.3878   51.0204   52.6531   54.2857   55.9184   57.5510

  Columns 25 through 32

   59.1837   60.8163   62.4490   64.0816   65.7143   67.3469   68.9796   70.6122

  Columns 33 through 40

   72.2449   73.8776   75.5102   77.1429   78.7755   80.4082   82.0408   83.6735

  Columns 41 through 48

   85.3061   86.9388   88.5714   90.2041   91.8367   93.4694   95.1020   96.7347

  Columns 49 through 50

   98.3673  100.0000

>>
```

Discussion and Conclusion:

- In the first section, the entire matrix A was displayed.
- In the second, the columns below row 3 were shown.
- In the third, all columns were shown; in the fourth, the final column was shown.
- In the second-to-last section, 100 linearly spaced points between 20 and 100 were generated.
- In the final section, we created 50 evenly spaced linear points between 20 and 100.

TASK:10

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

$x + 2y + 3z = 1$
$4x + 5y + 6z = 2$
$7x + 8y = 1$

where:
X= [x y z]
A= [1 2 3 ;4 5 6; 7 8 9]
b= [1;2;3]
Ax=b

Problem Analysis:

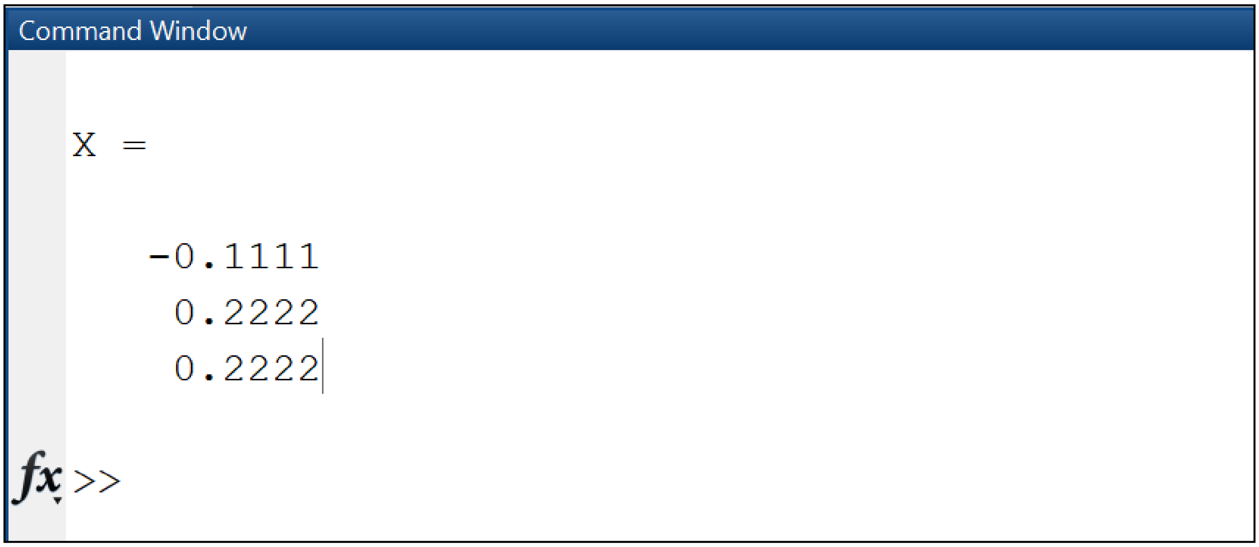
Using MATLAB, we need to solve the linear system of three unknowns using the inverses of matrices.

Algorithm:

- Write matrices
- Put matrices in inverse code
- Output resultant matrix

Code:

Output / Graphs / Plots / Results:



```
X =

   -0.1111
    0.2222
    0.2222

fx >>
```

Discussion and Conclusion:

Using MATLAB, we were able to effectively find the matrix's inverse and solve the linear system of equations.

TASK:11

Solve Task 10 by taking the equations from user.

Problem Analysis:

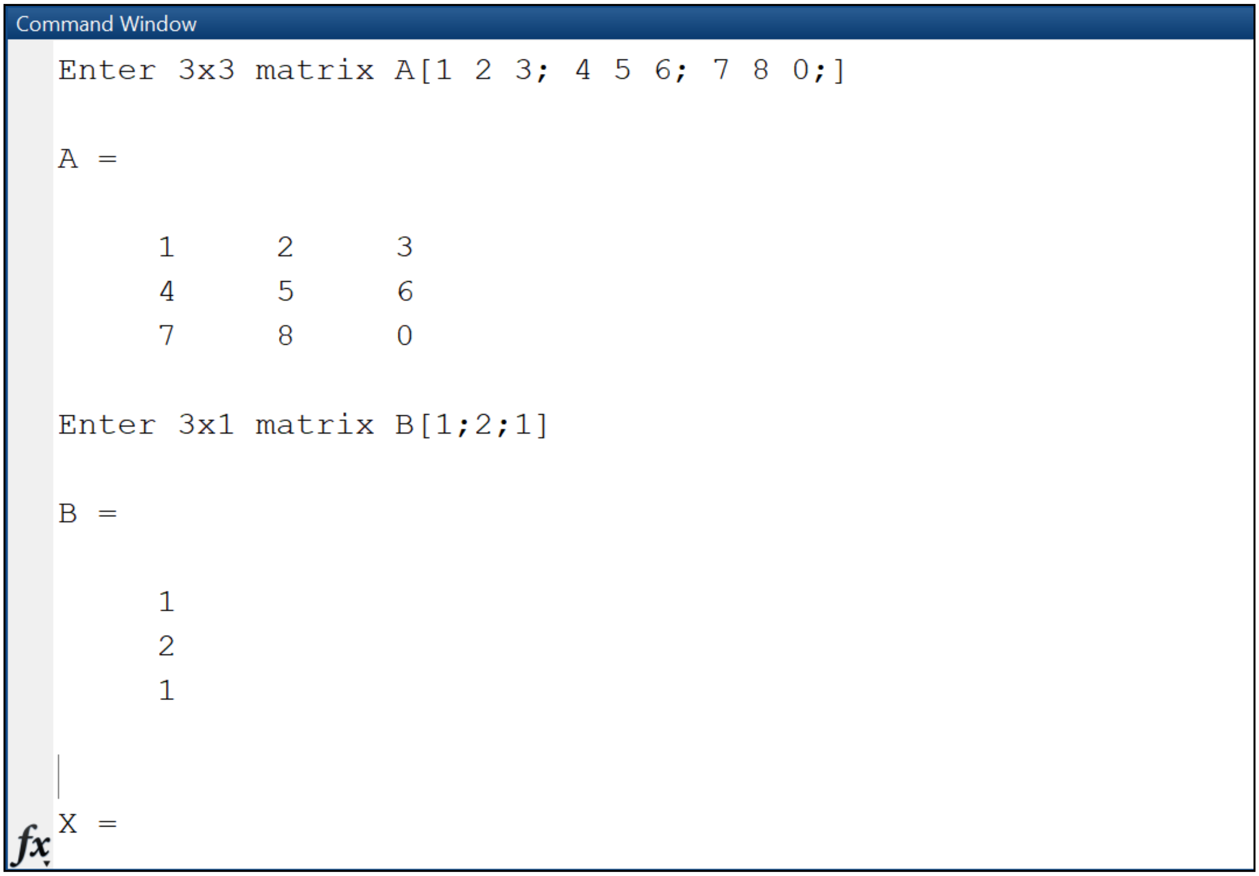
Here we will input the matrix from user and apply matrix computations upon it.

Algorithm:

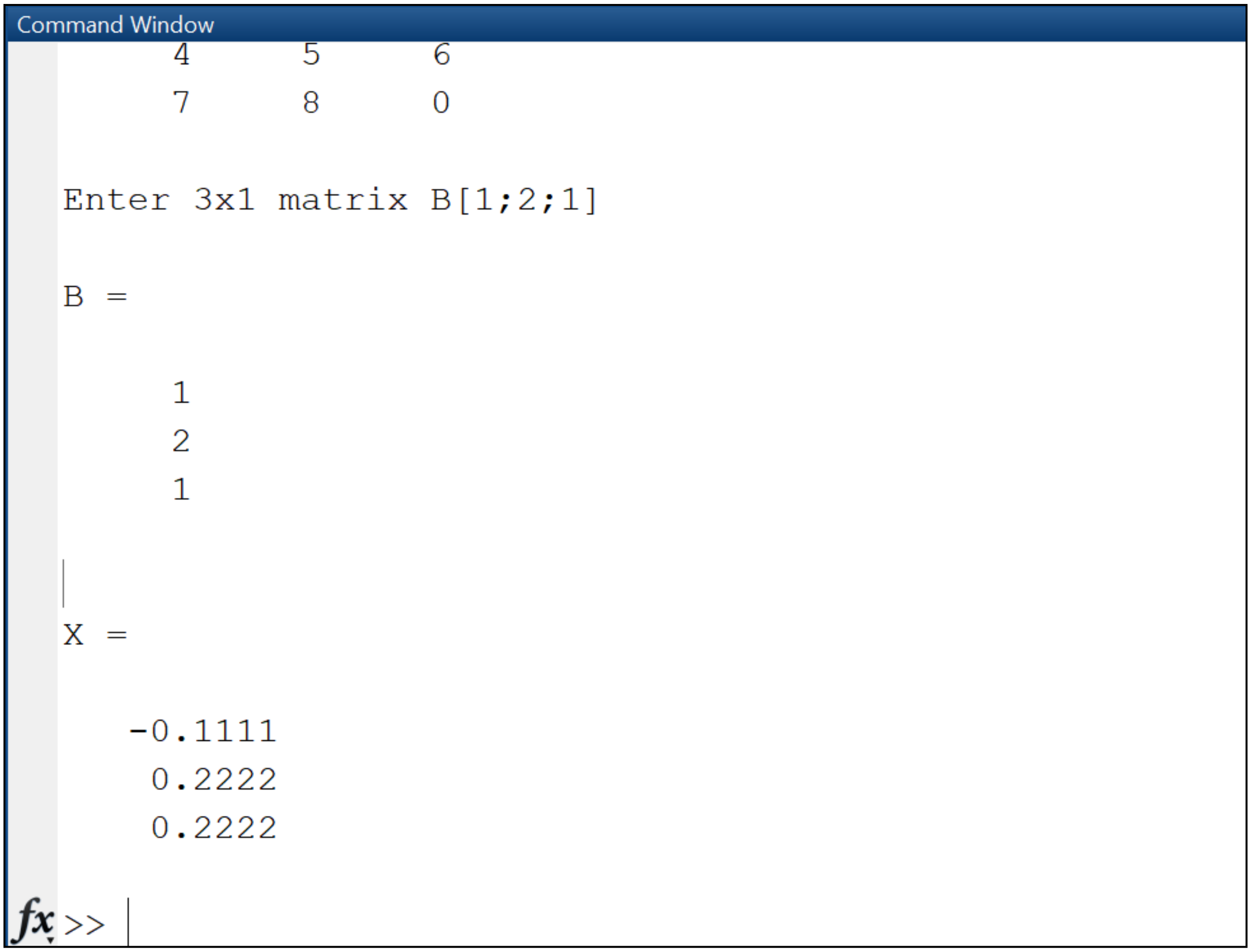
- Input the coefficient matrix from user
- Input the constant matrix from the user
- Perform inversion formula and store it in a matrix X
- Display X

Code:

Output / Graphs / Plots / Results:



```
Enter 3x3 matrix A[1 2 3; 4 5 6; 7 8 0;]

A =

     1     2     3
     4     5     6
     7     8     0

Enter 3x1 matrix B[1;2;1]

B =

     1
     2
     1


X =
```



```
     4     5     6
     7     8     0

Enter 3x1 matrix B[1;2;1]

B =

     1
     2
     1


X =

   -0.1111
    0.2222
    0.2222

fx >>
```

Discussion and Conclusion:

We can also input the matrix and perform matrix computations on it.

Conclusion:

I concluded that MATLAB is a useful tool for dealing with matrices and computing with them on computers. I acquired new skills for manipulating matrices and applying various mathematical operations to them. I discovered several definite sequences and how to produce a random discrete time signal. Along with that, I also learnt how to create sub-matrices from matrices that were provided. I also learnt how to use some of MATLAB's built-in functions, such as floor, ceiling, round, and others, on a given matrix. The final thing I learnt was how to use matrices and built-in MATLAB functions to solve a linear system with three unknowns.