Name : Ahmad Baseer **Roll no :124**
Section : C

Lab # 12

Implementation of Basic Operations on Matrix

Theory:

The best way for you to get started with MATLAB is to learn how to handle matrices. This section shows you how to do that. In MATLAB, a matrix is a rectangular array of numbers. Special meaning is sometimes attached to 1-by-1 matrices, which are scalars, and to matrices with only one row or column, which are vectors. MATLAB has other ways of storing both numeric and nonnumeric data, but in the beginning, it is usually best to think of everything as a matrix. The operations in MATLAB are designed to be as natural as possible. Where other programming languages work with numbers one at a time, MATLAB allows you to work with entire matrices quickly and easily. MATLAB was written originally to allow mathematicians, scientists, and engineers to handle the mechanics of linear algebra — that is, vectors and matrices — as effortlessly as possible. In this section we introduce these concepts.

Scalar:

A scalar is 1-by-1 and appears in MATLAB as a single real or complex number.

Example 10.1

[7], [583.62], [-3.51], [5.46097e-14], [83+4i].

Vectors:

A vector is an ordered list of numbers. You can enter a vector of any length in MATLAB by typing a list of numbers, separated by commas and spaces, inside square bracket.
Or
A vector is 1-by-n or n-by-1, and appears in MATLAB as a row or column of real or complex numbers.

Example 10.2

```
>> Z = [2,4,6,8]
Z =
2 4 6 8
```

>> X = 1:9
X =
1 2 3 4 5 6 7 8 9

<u>Matrices:</u>

A matrix is a rectangular array of numbers. Row and column vectors, which we discussed above, are examples of matrices. Consider the 3 × 4 matrix. It can be entered in MATLAB with the command.

>> A = [1, 2, 3, 4; 5, 6, 7, 8; 9, 10, 11, 12]
A =

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$$

<u>Example 10.3</u>

- To enter Durer's Matrix
  A=[16 3 2 13 ; 5 10 11 8 ; 9 6 7 12 ; 4 15 14 1]

  Ans: $\begin{pmatrix} 16 & 3 & 2 & 13 \\ 5 & 10 & 11 & 8 \\ 9 & 6 & 7 & 12 \\ 4 & 15 & 14 & 1 \end{pmatrix}$

- Once you have entered the matrix, it is automatically remembered in the Matlab work space

Sum, Transpose, Drag

>> **Sum(A)**
Ans
34 34 34 34
>> A' (transpose)
Ans

$$\begin{pmatrix} 16 & 5 & 9 & 4 \\ 3 & 10 & 6 & 15 \\ 2 & 11 & 7 & 14 \\ 13 & 8 & 12 & 1 \end{pmatrix}$$

>> **Sum(diag(A))**
Ans
34

Exercise:

1. **Consider the given matrix and observe the output.**

Program	Output
C = [1 2 3] **size(C)**	`C =` `1     2     3` `ans =` `1(row)     3(columns)` //Matrix is of 1 by 3 size.
a = [1+j*2 1-j*7] **a’** **a.’**	`a = [1+j*2    1-j*7]` `a' //It will change rows into columns after taking conjugate of entities.` `a =` `1.0000 + 2.0000i` `1.0000 - 7.0000i` `ans =` `1.0000 - 2.0000i` `1.0000 + 7.0000i` `a.' //It will change rows into columns without taking conjugate of entities.` `ans =` `1.0000 + 2.0000i` `1.0000 - 7.0000i`
G = [1 2 3;4 5 6;7 8 9]	`G =` `1     2     3` `4     5     6` `7     8     9` //The command will simply shows a 3 by 3 matrix

2. Consider the given matrices and observe the output.

Program	Output
V = [2:0.25:4] **U = [0:-1:-4]**	`V = [2:0.25:4]  //Comand will start taking entities from 2.0 and it will continue taking enities by adding 0.25 until it reaches 4.0` `U = [0:-1:-4]  //Comand will start taking entities from 0 and it will continue taking enities by adding -1 until it reaches -4`

	`V =`
	`   2.0000  2.2500  2.5000  2.7500 3.0000 3.2500 3.5000 3.7500  4.0000`
	`U =`
	`     0    -1    -2    -3    -4`

V(1)	`V(1)  //It will print the entity which is at first place`
	`ans =`
	`     2`

V(1:3)	`V(1:3) //It will print entities from 1st column to 3rd column`
	`ans =`
	`    2.0000    2.2500    2.5000`

V(1:3)-V(2:4)	`V(1:3)-V(2:4) //It will subtract the entities of first 3 columns from The enittites of 2,,3,4 columns.`
	`ans =`
	`   -0.2500   -0.2500   -0.2500`

U+V	`U+V //The command is used for addition but both matrices are not of Equal size so operation is not possible`
	`Arrays have incompatible sizes for this operation.`

-2*V	`-2*V //It will simply Multiply -2 with V Matrix.`
	`ans =`
	`   -4.0000   -4.5000   -5.0000   -5.5000   -6.0000   -6.5000   -7.0000`
	`-7.5000   -8.0000`

V/3	`V/3  //It will simply divide V Matrix with 3.`
	`ans =`
	`    0.6667    0.7500    0.8333    0.9167    1.0000    1.0833    1.1667`
	`1.2500    1.3333`

-2*U+V/3	`-2*U+V/3 //As here ultimately addition is also taking place so both matrices are not of Equal size so operation is not possible`
	`Arrays have incompatible sizes for this operation.`

3. Consider the Matrix and observe the output.

Program	Output

G = [1 2 3;4 5 6;7 8 9]	`G = [1 2 3;4 5 6;7 8 9]`
	`G =`
	`     1     2     3`
	`     4     5     6`
	`     7     8     9`
	//The command will simply shows a 3 by 3 matrix

g*g	g*g
	`Unrecognized function or variable 'g'.`
	`G*G  //Simply multipy matrices`

	ans = 30 36 42 66 81 96 102 126 150
g*g’	g*g' Unrecognized function or variable 'g'. G*G' //Simply multipy matrices after taking transpose ans = 14 32 50 32 77 122 50 122 194
g.*g	g.*g Unrecognized function or variable 'g'. G.*G //It will just multiply Correspond entries. ans = 1 4 9 16 25 36 49 64 81
G*g	G*g Unrecognized function or variable 'g'. Did you mean: G*G ans = 30 36 42 66 81 96 102 126 150

G(1:2,1:2)	G(1:2,1:2) //It will just show first 2 rows and first 2 columns
	ans =
	1 2
	4 5

4. Consider the given matrices.

Program	Output

S = [2 2j;7j 3]
T = [-5 1+2j;-j 3]
	S = [2 2j;7j 3]
	T = [-5 1+2j;-j 3]

	S = 2.0000 + 0.0000i 0.0000 + 2.0000i 0.0000 + 7.0000i 3.0000 + 0.0000i T = -5.0000 + 0.0000i 1.0000 + 2.0000i 0.0000 - 1.0000i 3.0000 + 0.0000i //The command will simply shows a 2 by 2 matrix of complex entities
S.T	S.T Dot indexing is not supported for variables of this type. S*T //Simply multipy matrices ans = -8.0000 + 0.0000i 2.0000 +10.0000i 0.0000 -38.0000i -5.0000 + 7.0000i
T.S	T.S Dot indexing is not supported for variables of this type. T*S //Simply multipy matrices ans = -24.0000 + 7.0000i 3.0000 - 4.0000i 0.0000 +19.0000i 11.0000 + 0.0000i

inv(S)	inv(S) //Simply take the inverse.
	ans =
	0.1500 + 0.0000i 0.0000 - 0.1000i
	0.0000 - 0.3500i 0.1000 + 0.0000i

T’.inv(S)	T'.inv(S) T'.inv(S) ↑ Invalid use of operator. T’*inv(S) //Simply multipy matrices after taking transpose of first and inverse of Second. ans = -0.4000 + 0.0000i 0.0000 + 0.6000i 0.1500 - 1.3500i 0.1000 - 0.1000i

5. **Generate a 4*4 matrix P, whose first column is an array of 0, 2, 4 and 6; second column is an array of 1, 3, 5, and 7; third is the second column in reverse order and fourth column is the first column in reverse order.**
   Command:
   ```
   P = [0 1 7 6;2 3 5 4;4 5 3 2;6 7 1 0]
   ```
   Output:
   ```
   P =
      0    1    7    6
      2    3    5    4
      4    5    3    2
      6    7    1    0
   ```
6. **Generate using commands:**

   i. **Innermost 2*2 submatrix of P**.
      Command:
      ```
      P(2:3,2:3)
      ```
      Output:
      ```
      ans =
         3    5
         5    3
      ```
   ii. **Top rightmost 3*3 submatrix of P.**
      Command:
      ```
      P(1:3,2:4)
      ```
      Output:
      ```
      ans =
         1    7    6
         3    5    4
         5    3    2
      ```
   iii. **Bottom leftmost 2*2 submatrix of P**.
      Command:
      ```
      P(3:4,1:2)
      ```
      Output:
      ```
      ans =
         4    5
         6    7
      ```