

University of Engineering & Technology Lahore

Faculty of Engineering

Experiment # 2

Title: introduction of Arrays, Two dimensional plots and Elementary sequence.

Equipment Required: Personal computer (PC) with windows operating system and MATLAB software

Arrays

As mentioned earlier, the name MATLAB stands for MATrix LABoratory because MATLAB has been designed to work with *matrices*. A matrix is a rectangular object (e.g. a *table*) consisting of rows and columns. A *vector* is a special type of matrix, having only one row, or one column.

MATLAB handles vectors and matrices in the same way, but since vectors are easier to think about than matrices.

A one dimensional array is a list of number that is placed in a row or a column. The vector is created by typing the elements inside the square brackets []

Variable_name = [type vector elements]

A vector is created with constant spacing by specifying the first term, the spacing, and the last term

Variable_name = [firstterm : spacing : lastterm]

A vector in which the first element is x_i , the last element is x_f , and the number of elements is n is created by typing the linspace command:

Variable_name =linspace (xi, xi, n)

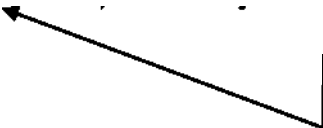
A matrix is created by assigning the elements of the matrix to a variable. This is done by typing the elements, row by row, inside square brackets []. First type the left bracket [, then type the first row separating the elements with spaces or commas. To type the next row type a semicolon or press Enter. Type the right bracket] at the end of the last row.

variable_name = [1st row elements; 2nd row elements; 3rd row elements; ;

last row elements]

Example:

```
>> a = [5 35 43; 4 76 81; 21 32 40]
```



A semicolon is typed before a new line is entered.

Rows of a matrix can also be entered as vectors using the notation for creating vectors with constant spacing, or the `linspace` command.

Example:

```
>> A = [1:2:11; 0:5:25; linspace(10,60,6); 67 2 43 68 4 13]
```

A =

```
1   3   5   7   9  11
0   5  10  15  20  25
10  20  30  40  50  60
67   2  43  68   4  13
```

In this example the first two rows were entered as vectors using the notation of constant spacing, the third row was entered using the `linspace` command, and in the last row the elements were entered individually.

The zeros, ones and eye Commands

The `zeros` (m, n), the `ones`(m, n), and `eye` (n) commands can be used to create matrices that have elements with special values. The `eye` (n) command creates a square matrix with n rows and n columns in which the diagonal elements are equal to **1**, and the rest of the elements are 0. This matrix is called the identity matrix.

Examples:

```
>> zr = zeros(3,4)
```

zr =

```
0   0   0   0
0   0   0   0
0   0   0   0
```

```
>> ne=ones(3,5) ne =
```

```
1   1   1   1   1
1   1   1   1   1
1   1   1   1   1
```

```
>> idn= eye(5)
```

idn =

```
0   0   0   0   0
0   1   0   0   0
0   0   1   0   0
0   0   0   1   0
0   0   0   0   1
```

The Transpose Operator:-

The transpose operator is applied by typing a single quote' following the variable to be transposed.

Examples:

```
>> C = [2 5 14 8 ; 9 5 32 1; 1 2 3 4]
```

```
C =  
    2     5    14     8  
    9     5    32     1  
    1     2     3     4
```

```
>> D = C'
```

```
D =  
    2     9    14  
    5     5    32  
    1     2     3  
    8     1     4
```

Array Addressing

Elements in an array (either vector or matrix) can be addressed individually or in subgroups.

Vector:-

The address of an element in a vector is its position in the row (or column). For a vector named *ve*, *ve(k)* refers to the element in position *k*. The first position is 1. For example, if the vector *ve* has nine elements: *ve*=35 46 78 23 5 14 81 3 55 then

ve(4) = 23, *ve(7)* = 81, and *ve(1)* = 35.

Example:

```
>> VCT = [35 46 78 23 5 14 81 3 55]
```

```
VCT =  
    35 46 78 23 5 14 81 3 55
```

```
>> VCT(4)
```

```
ans= 23
```

```
>> VCT(2)+VCT(8)
```

```
ans= 49
```

```
>> MAT = [3 11 6 5; 4 7 10 2; 13 9 0 8]
```

```
MAT =
```

```
3    11 6    5
```

```
4     7 10   2
```

```
13   9    0   8
```

```
>> MAT(3,1)
```

```
ans =
```

```
13
```

```
>> MAT = [3 11 6 5; 4 7 10 2; 13 9 0 8]
```

```
MAT =
```

```
3    11 6    5
```

```
4     7 10   2
```

```
13   9    0   8
```

```
>> MAT(3,1)=20
```

```
MAT =
```

```
3    11 6    5
```

```
4     7 10   2
```

```
20   9    0   8
```

```
>> MAT(2,4)-MAT(1,2)
```

```
ans =
```

```
-9
```

Using A Colon: In Addressing Arrays

A colon can be used to address a range of elements in a vector or a matrix.

For a vector:

$va(:)$ Refers to all the elements of the vector va (either a row or a column vector). $va(m:n)$ Refers to elements m through n of the vector va .

Example:

```
>> v = [4 15 8 12 34 2 50 23 11]
v =
15 8 12 34 2 50 23 11
>> u = v(3:7)
u =
8 12 34 2 50
```

For a matrix:

$A(:,n)$ Refers to the elements in all the rows of column n of the matrix A .

$A(n,:)$ Refers to the elements in all the columns of row n of the matrix A .

$A(:,m:n)$ Refers to the elements in all the rows between columns m and n of the matrix A .

$A(m:n,:)$ Refers to the elements in all the columns between rows m and n of the matrix A .

$A(m:1},p:q)$ Refers to the elements in rows m through n and columns p through q of the matrix A .

Adding Elements to Existing Variables:

A variable that exists as a vector, or a matrix, can be changed by adding elements to it.

Example:

```
>> DF = 1:4
DF =
1 2 3 4
>> DF(5:10)=10:5:35
DF =
1 2 3 4 10 15 20 25 30 35
```

Adding elements to a matrix:

Rows and/or columns can be added, to an existing matrix by assigning values to the new rows or columns. This can be done by assigning new values, or by appending existing variables. This must be done carefully since the size of the added rows or columns must fit the existing matrix. Examples are given below.

```
>> E = [1 2 3 4; 5 6 7 8]
```

```
    1    2    3    4
    5    6    7    8
```

```
>> E(3,:)=[10:4:22]
```

```
E =
```

```
    1    2    3    4
    5    6    7    8
   10  14  18  22
```

```
>> K = eye(3)
```

```
K =
```

```
    1    0    0
    0    1    0
    0    0    1
```

```
>> G = [E K]
```

```
G =
```

```
    1    2    3    4    1    0    0
    5    6    7    8    0    1    0
   10  14  18   22    0    0    1
```

Built-in Functions for Handling Arrays:-

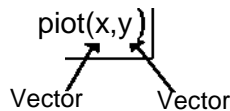
MATLAB has many built-in functions for managing and handling arrays. Some of these are listed below:

Function	Description	Example
length (A)	Returns the number of elements in the vector A.	>> A = [5 9 2 4]; >> length(A) ans = 4
size(A)	Returns a row vector [m, n] , where m and n are the size $m \times n$ of the array A.	>> A = [6 1 4 0 12; 5 19 6 8 2]; >> size(A) ans = 2 5
diag(A)	When A is a matrix, creates a vector from the diagonal elements of A.	>> A = [1 2 3; 4 5 6; 7 8 9] A = 1 2 3 4 5 6 7 8 9 >> vec = diag(A) vec = 1 5 9

Two Dimensional Plots

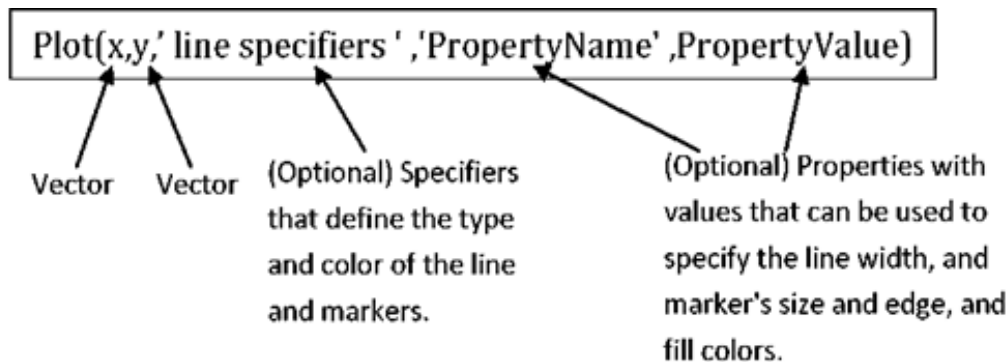
The plot Command:-

The plot command is used to create two-dimensional plots. The simplest form of the command is: The arguments x and y are each a vector (one-dimensional array). Both vectors must have the same



number of elements.

The plot command has additional optional arguments that can be used to specify the color and style of the line and the color and type of markers, if any are desired. With these options the command has the Graphs *in* the Same Plot:-



In many situations there is a need to make several graphs in the same plot. There are three methods to plot multiple graphs in one figure.

- By using the plot command
- By using the hold on, hold off commands
- By using the line command.

Formatting a Plot Using Commands

The formatting commands are entered after the plot or the fplot commands. The various formatting commands are:

The xlabel and ylabel commands:-

Labels can be placed next to the axes with the xlabel and ylabel commands which have the form:

`xlabel('text as string')`

ylabel ('text as string')

The title command:-

A title can be added to the plot with the command:

title ('text as string') [The text is placed at the top of the figure as a title.]

The text command:-

A text label can be placed in the plot with the text or gtext commands:

text(x,y,'text as string')

gtext('text as string')

The text command places the text in the figure such that the first character is Positioned at the point with the coordinates x, y (according to the axes of the figure).

The gtext command places the text at a position specified by the user. When the command is executed, the Figure Window opens and the user specifies the Position with the mouse.

Legend command:-

The legend command places a legend on the plot. The legend shows a sample of the line type of each graph that is plotted, and places a label, specified by the user, beside the line sample. The command is:

legend('string1', 'string1', ,pos)

The strings are the labels that are placed next to the line sample

The axis command:-

When the plot (x, y) command is executed, MATLAB creates axes with limits that are based on the minimum and maximum values of the elements of x and y. The axis command can be used to change the range and the appearance of the axes. In many situations a graph looks better if the range of the axes extend beyond the range of the data. The following are some of the possible forms of the axis command:

axis([x_{min}, x_{max}]) Sets the limits of the x axis (xmin and xmax are numbers).

axis([x_{min},x_{max},y_{min},y_{max}]) Sets the limits of both the x and y axes.

Plotting multiple plots on the same page:-

Command is: subplot(m,n,p)

The command divides the Figure Window (page when printed) into $m \times n$ rectangular subplots where plots will be created. The subplots are arranged like elements in a $m \times n$ matrix where each element is a subplot. The subplots are numbered from 1 through $m \times n$. The upper left is 1 and the lower right is the number $m \times n$. The numbers increase from left to right within a row, from the first row to the last. The command subplot(m, n, p) makes the subplot P current.

Elementary Sequence

A discrete time signal is represented as a sequence of numbers, called samples. These samples are denoted by $x(n)$ where the variable n is integer valued and represents in discrete instances in time. An example of a discrete time signal is:

$$x(n) = \{2, 1, -1, \underset{\uparrow}{0}, 1, 4, 3, 7\} \dots (1)$$

where the up arrow indicates the sample at $n = 0$

In MATLAB, a finite duration sequence is represented by a row vector. However, such a vector does not have any information about sample position n . Therefore a correct representation of $x(n)$ would require two vectors, one each for x and n ,

To represent the sequence defined in eq1, the following MATLAB command can be used:

$$>> n = [-3, -2, -1, 0, 1, 2, 3, 4] \qquad x = [2, 1, -1, 0, 1, 4, 3, 7]$$

We use several elementary sequences in digital signal processing for analysis purposes. Their definitions and MATLAB representations are given below.

Unit Sample Sequence:

$$\delta(n) = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases} = \left\{ \dots, 0, \underset{\uparrow}{0, 1, 0, 0, \dots} \right\}$$

In MATLAB the function `zeros (1, N)` generates a row vector of N zeros, which can be used to implement $\delta(n)$ over a finite interval. However, the logical relation $n==0$ is an elegant way of implementing $\delta(n)$. For example, to implement

$$\delta(n - n_0) = \begin{cases} 1, & n = n_0 \\ 0, & n \neq n_0 \end{cases}$$

over the $n_1 \leq n_0 \leq n_2$ interval, we will use the following MATLAB function.

```
function [x,n] = impseq(n0,n1,n2)
% Generates x(n) = delta(n-n0);      n1 <= n <= n2
% -----
% [x,n] = impseq(n0,n1,n2)
%
n= [n1:n2];
x = [(n-n0) == 0];
```

MATLAB Script:-

```
% Generation of a Unit Sample Sequence
% Generate a vector from -10 to 20
[x,n]=impseq(1,-10,20)
%plot the unit sample sequence
stem(n,u);
xlabel('time index n');ylabel('Amplitude');
title('Unit Sample Sequence');
axis([-10 20 0 1.2]);
```



Unit Step Sequence:

$$u(n) = \begin{cases} 1, & n \geq 0 \\ 0, & n < 0 \end{cases} = \left\{ \dots, 0, 0, \underset{\uparrow}{1}, 1, 1, \dots \right\}$$

In MATLAB the function **ones(1,N)** generates a row vector of N ones. It can be used to generate $u(n)$ over a finite interval. Once again an elegant approach is to use the logical relation $n \geq 0$. To implement

$$u(n-n_0) = \begin{cases} 1, & n \geq n_0 \\ 0, & n < n_0 \end{cases}$$

over the $n_1 \leq n_0 \leq n_2$ interval, we will use the following MATLAB function.

```
function [x,n] = stepseq(n0,n1,n2)
% Generates x(n)= u(n-n0); n1 <= n <= n2
% -----
% [x,n] = stepseq(n0,n1,n2)
%
n = [n1:n2]; x = [(n-n0) >= 0];
```