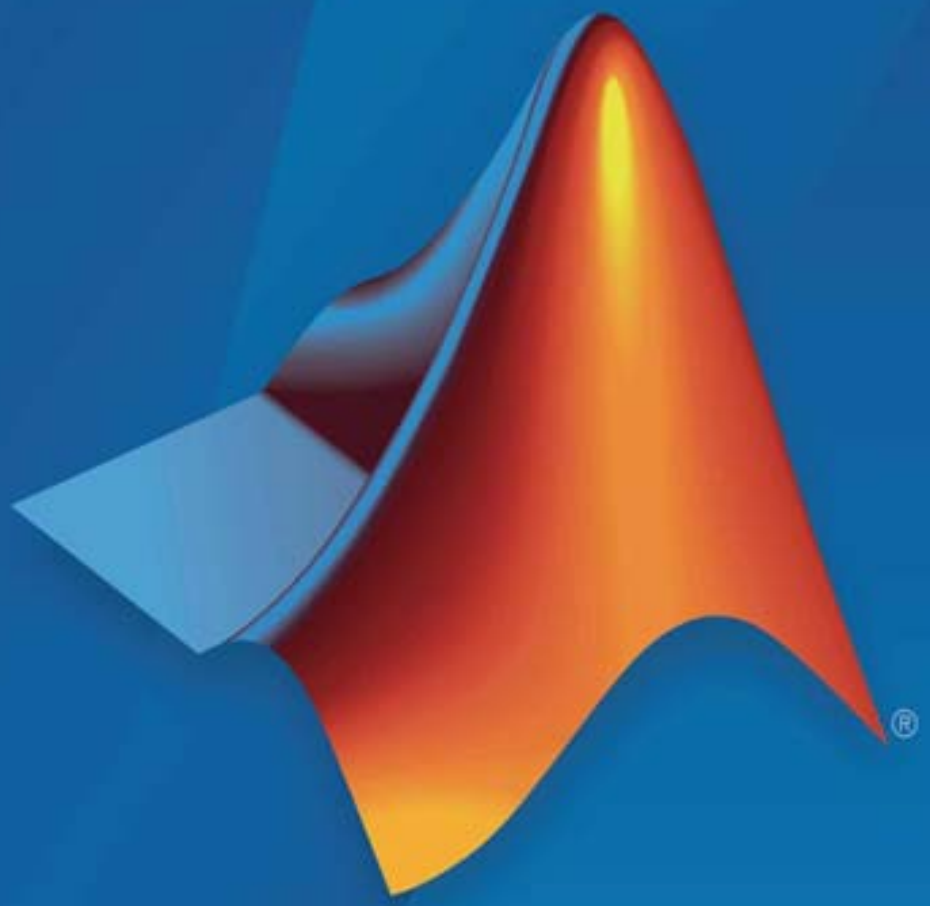
الجامعة المستنصرية
كلية الهندسة
قسم هندسة الموارد المائية



Lecture 8: Plotting in MATLAB

المحاضرة الثامنة: الرسم في ماتلاب

المنهاج الدراسي لمادة البرمجه والتطبيقات (ماتلاب)
الكورس الدراسي الاول / المرحلة الثانية

من اعداد د. صباح حسن لفته

Introduction

Large tables of data are difficult to interpret. Engineers use graphing techniques to make the information easier to understand. With a graph, it is easy to identify trends, pick out highs and lows, and isolate data points that may be measurement or calculation errors. Graphs can also be used as a quick check to determine whether a computer solution is yielding expected results. To do so, MATLAB provides tools to plot either two-dimensional figures or three-dimensional figures (out of the course syllabus), currently, we will focus on two-dimensional plots.

a- Two-Dimensional plots

The most useful plot for engineers is the *x*–*y* plot. A set of ordered pairs is used to identify points on a two-dimensional graph; the points are then connected by straight lines. The values of *x* and *y* may be measured or calculated. Generally, the independent variable is given the name *x* and is plotted on the *x*-axis, and the dependent variable is given the name *y* and is plotted on the *y*-axis.

Suppose we have a set of time versus distance data were obtained from an experiment. In this case, we can store the time values in a vector called **x** (the user can define any convenient name) and the distance values in a vector called **y**:

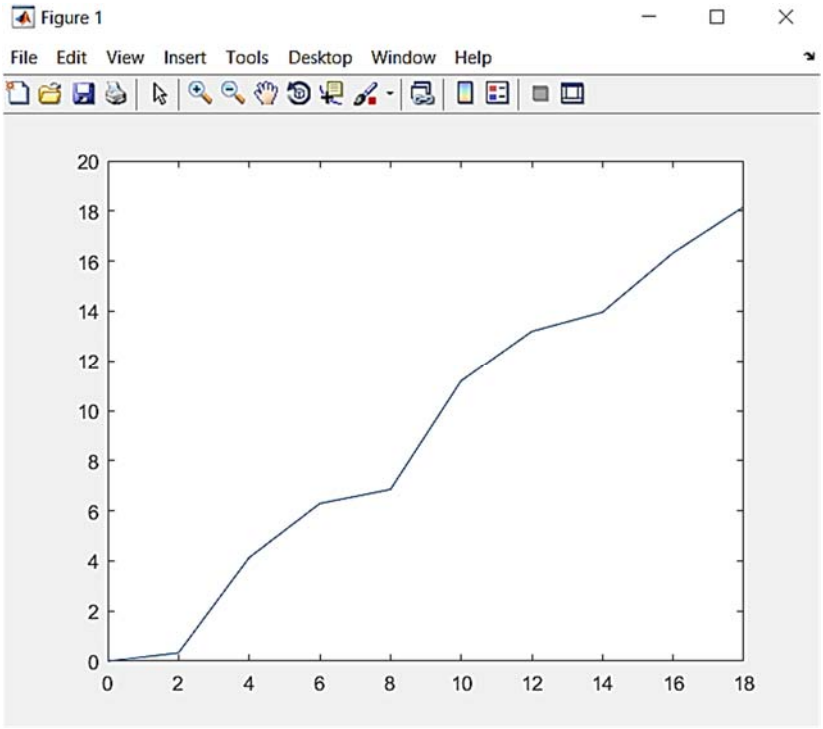
Time, s	Distance, cm
0	0
2	0.33
4	4.13
6	6.29
8	6.85
10	11.19
12	13.19
14	13.96
16	16.33
18	18.17

```
>> x = [0:2:18];
>> y = [0, 0.33, 4.13, 6.29, 6.85, 11.19, 13.19, 13.96, 16.33, 18.17];
```

To plot these points, we use the ***plot command***, with **x** and **y** as arguments:

```
>> plot(x,y)
```

After the execution of above statement, a graphic window automatically opens, which MATLAB calls Figure 1. The resulting plot is shown below.



b- Plot components

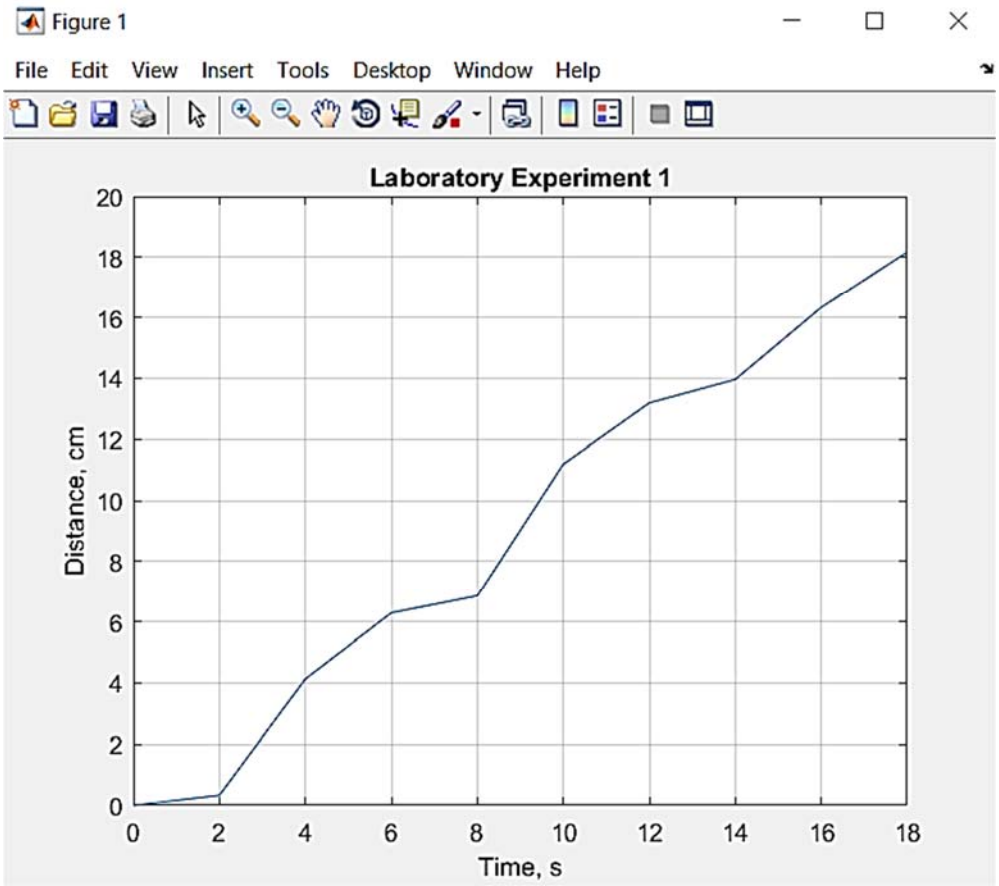
In order to make the plots in MATLAB more readable, its good engineering practice to include ***axis labels*** and ***a title*** in these plots. The following commands add a title, *x*- and *y*-axis labels, and a background grid:

```
>> plot(x,y)
>> title('Laboratory Experiment 1')
>> xlabel('Time, sec')
```

>> ylabel('Distance, ft')

>> grid on

The resulted figure will be like,



As with any MATLAB commands, they could also be combined onto one or two lines to save space.

>> plot(x,y) , title('Laboratory Experiment 1')

>> xlabel('Time, sec'), ylabel('Distance, ft'), grid on

As we type the previous commands into MATLAB, we can notice that the text color changes to red when we enter a single quote ('). This alerts us that we are starting a string. The color changes to purple when we type the final single quote ('), indicating

that we have completed the string. Its very important to pay attention to these visual aids will help us avoid coding mistakes.

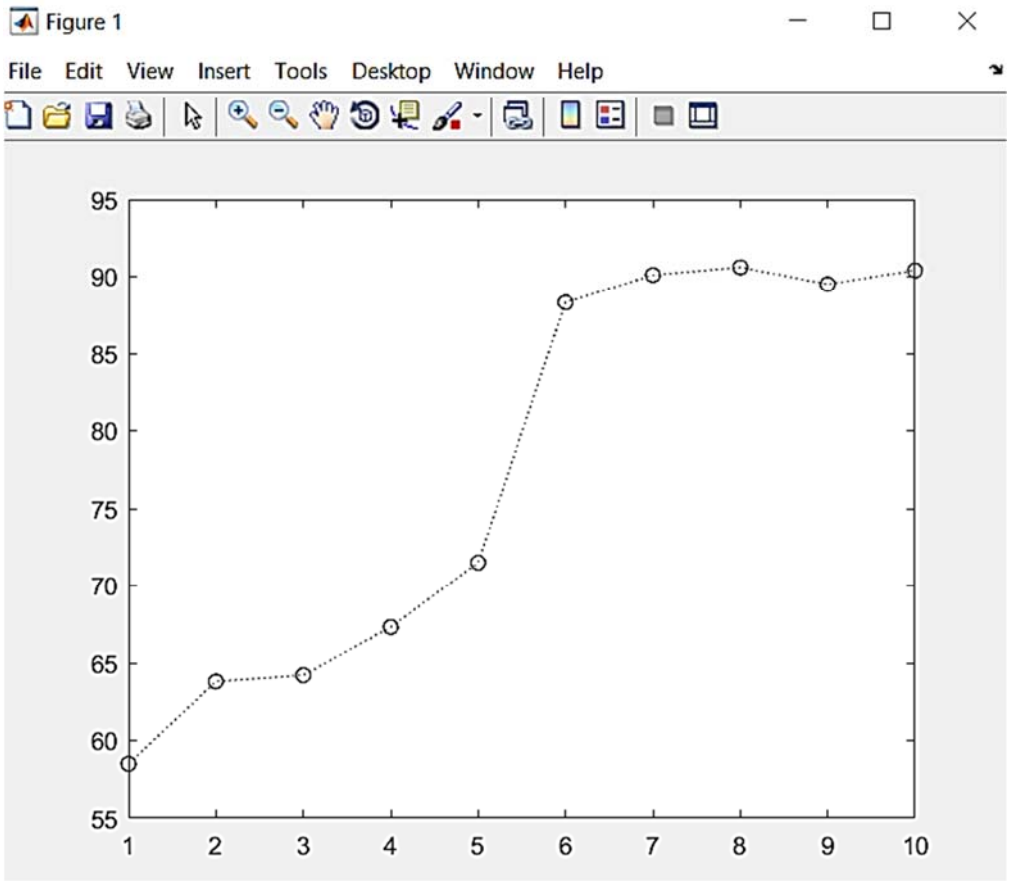
Lines, colors, and mark styles are useful plot components that change the appearance of our plots. The following commands are used to control the line, color, and mark styles in plots.

Line Type	Indicator	Mark Type	Indicator	Color	Indicator
solid	-	point	.	blue	b
dotted	:	circle	o	green	g
dashdot	- .	x-mark	x	red	r
dashed	--	plus	+	cyan	c
no line	none	star	*	magenta	m
		square	s	yellow	y
		diamond	d	black	k
		triangle up	^	white	w
		triangle down	v		
		triangle left	<		
		triangle right	>		
		pentagon	p		
		hexagram	h		

We can select solid (the default), dashed, dotted, and dash-dot line styles, and we can choose to show the points. The choices among marks include plus signs, stars, circles, and x-marks, among others. There are seven different color choices. The following commands illustrate the use of line, color, and mark styles:

```
>> x = [1:10];
>> y = [58.5, 63.8, 64.2, 67.3, 71.5, 88.3, 90.1, 90.6, 89.5, 90.4];
>> plot(x,y,':ok')
```

The resulting plot as shown below consists of a dashed line, together with data points marked with circles. The line, the points, and the circles are drawn in black.



Axis scaling and annotating plots are other options that can be used in MATLAB when plotting functions. MATLAB automatically selects appropriate *x*-axis and *y*-axis scaling, but sometimes, it is useful for the user to be able to control the scaling. Control is accomplished with the ***axis function***. Executing the axis function without any input

>> axis

freezes the scaling of the plot. The axis function also accepts input defining the *x*-axis and *y*-axis scaling. The argument is a single matrix, with four values representing:

• The minimum *x* value shown on the *x*-axis

• The maximum *x* value shown on the *x*-axis

• The minimum *y* value shown on the *y*-axis

• The maximum *y* value shown on the *y*-axis

Thus, the command

>> axis([-2, 3, 0, 10])

fixes the plot axes to *x* from - 2 to + 3 and *y* from 0 to 10.

It is often useful to create plots where the scaling is the same on the *x*- and *y*-axis. This is achieved with the statement:

>> axis equal

MATLAB offers several additional functions such as ***legend and text functions*** which are used to annotate plots. The ***legend function*** requires the user to specify a legend in the form of a string for each line plotted, and defaults to a display in the upper right-hand corner of the plot. The ***text function*** allows us to add a text box to our plot, which is useful for describing features on the graph. It requires the user to specify the location of the lower left-hand corner of the box in the plot window as the first two input fields, with a string specifying the contents of the text box in the third input field. The use of both legend and text is demonstrated in the following code, which modifies the graph from Figure 5.8b.

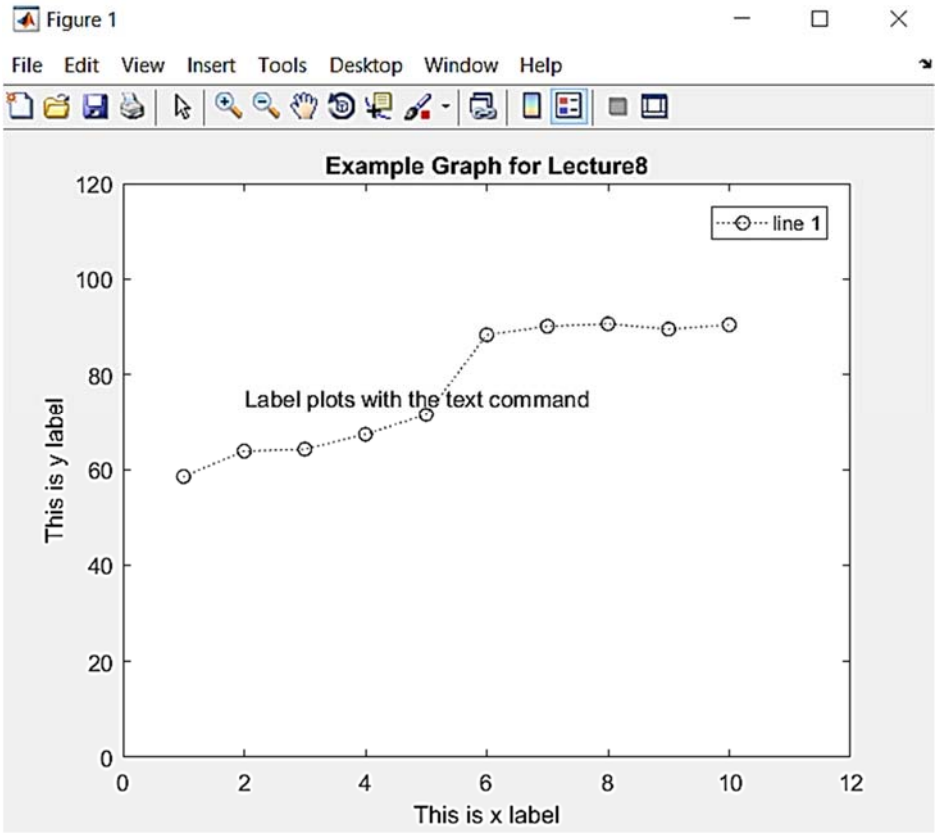
```
>> x = [1:10];
>> y = [58.5, 63.8, 64.2, 67.3, 71.5, 88.3, 90.1, 90.6, 89.5, 90.4];
>> plot(x,y,':ok')
```

```
>> legend('line 1')
>> text(1,75,'Label plots with the text command')
```

We added a title, *x* and *y* labels, and adjusted the axis with the following commands:

```
>> xlabel('This is x label'), ylabel('This is y label')
>> title('Example Graph for Lecture8')
>> axis([0,12,0,120])
```

The results are shown in the below figure:

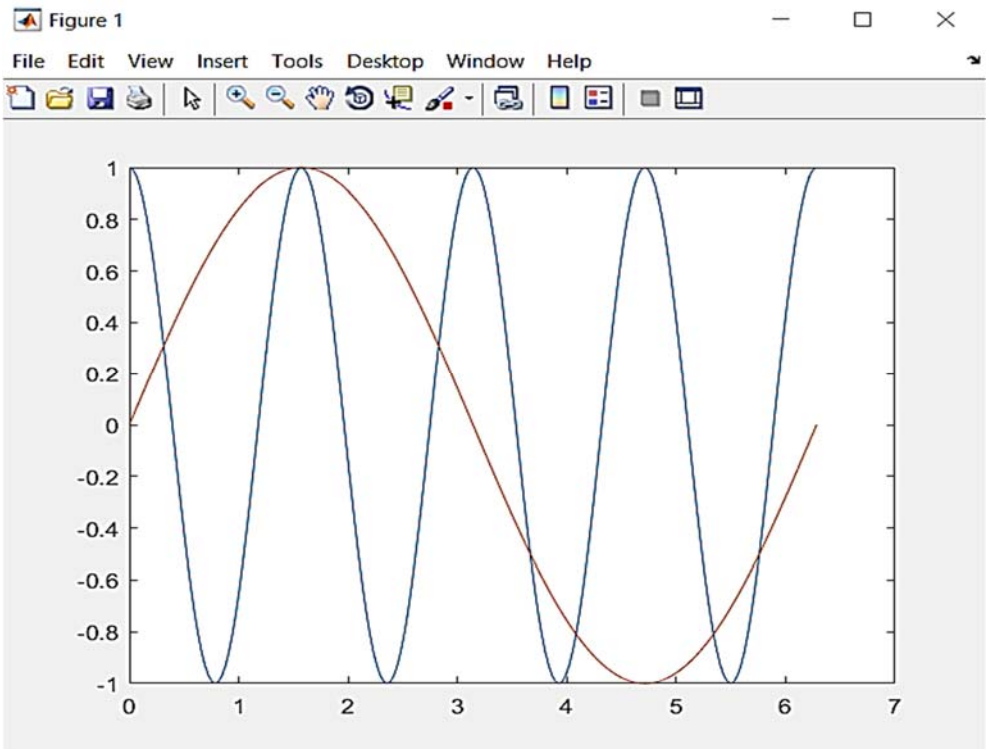


c- Plots with More than One Line

Plots with more than one line can be created in several ways. By default, the execution of a second plot statement will erase the first plot. However, we can layer plots on the top of one another by using the ***hold on command***. Execute the following

statements to create a plot with both functions plotted on the same graph, as shown below:

```
>> x = 0:pi/100:2*pi;
>> y1 = cos(x*4);
>> plot(x,y1)
>> y2 = sin(x);
>> hold on;
>> plot(x, y2)
```



Semicolons are optional on both the plot statement and the hold on statement. MATLAB will continue to layer the plots until the hold off command is executed:

```
>> hold off
```

Another way to create a graph with multiple lines is to add both lines in a single plot command. MATLAB interprets the input to plot as alternating *x* and *y* vectors, as in,

>> plot(X1, Y1, X2, Y2)

where the variables X1, Y1 form an ordered set of values to be plotted and X2, Y2 form a second ordered set of values. Using the data from the previous example,

>> plot(x, y1, x, y2)

produces the same graph as the figure plotted above, with one exception, the two lines are different colors. If plot is used with two arguments, one a vector and the other a matrix, MATLAB successively plots a line for each row in the matrix. For example, we can combine y1 and y2 into a single matrix and plot against x:

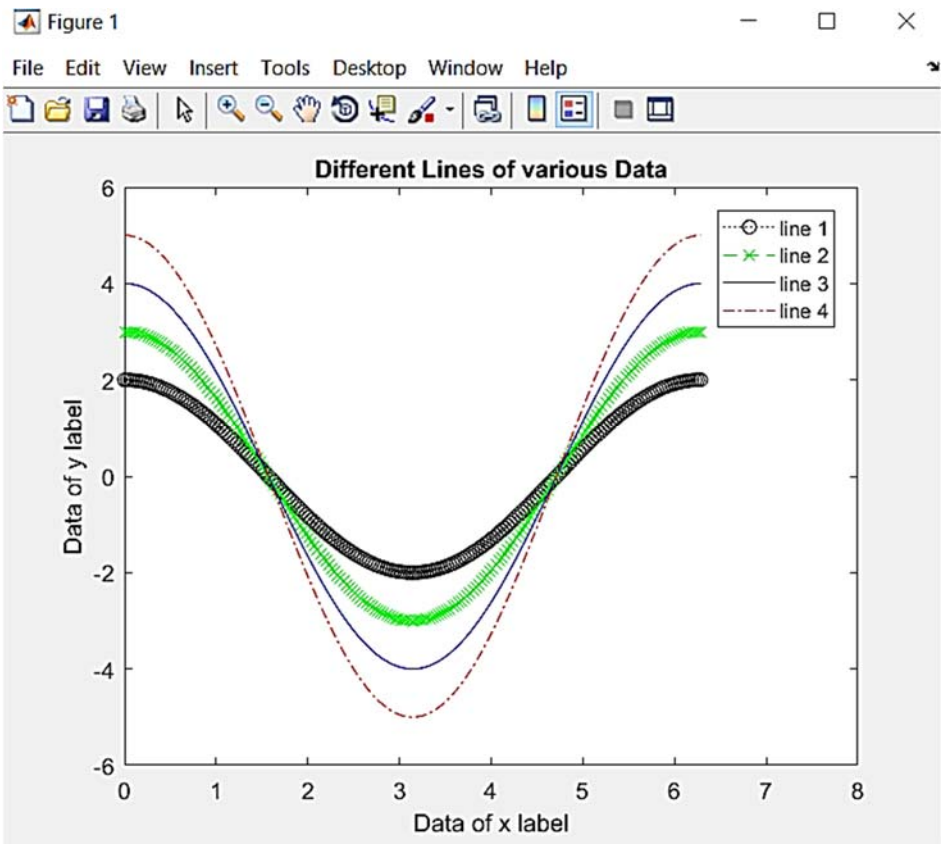
```
>> Y = [y1; y2];
>> plot(x,Y)
```

Here is another more complicated example:

```
>> X = 0:pi/100:2*pi;
>> Y1 = cos(X)*2;
>> Y2 = cos(X)*3;
>> Y3 = cos(X)*4;
>> Y4 = cos(X)*5;
>> Z = [Y1; Y2; Y3; Y4];
>> plot(X, Y1, ':ok', X, Y2, '--xg', X, Y3, '-b', X, Y4,'-.r',)
>> legend('line 1', 'line 2', 'line 3', 'line 4')
>> xlabel('Data of x label'), ylabel('Data of y label')
>> title('Different Lines of various Data')
>> axis([0,8,0,1.2])
```

This code produces the same result figure shown below as using,

>> plot(X, Z)



Hint:

To clear a figure, use the ***clf command***. To close the active figure window, use the ***close command***, and to close all open figure windows use ***close all command***.

d- Subplots

The ***subplot command*** allows us to subdivide the graphing window into a grid of ***m*** rows and ***n*** columns. The function,

>> subplot(m,n,p)

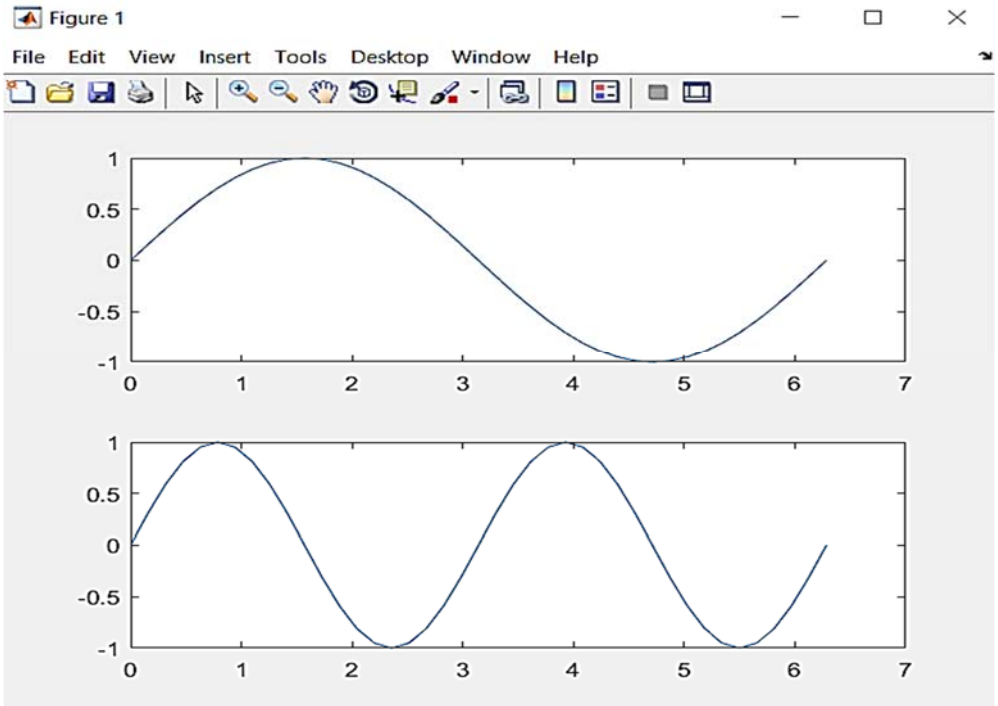
splits the figure into an **m * n matrix**. The variable ***p*** identifies the portion of the window where the next plot will be drawn. For example, if the command,

>> subplot(2,2,1)

is used, the window is divided into two rows and two columns, and the plot is drawn in the upper left-hand window. The windows are numbered from left to right, top to

bottom. Similarly, the following commands split the graph window into a top plot and a bottom plot:

```
>> x = 0:pi/20:2*pi;
>> subplot(2,1,1)
>> plot(x,sin(x))
>> subplot(2,1,2)
>> plot(x,sin(2*x))
```



The first graph is drawn in the top window since p = 1. Then the subplot command is used again to draw the next graph in the bottom window. Titles are added above each sub-window as the graphs are drawn, as are *x*- and *y*-axis labels and any annotation desired.

Example (1): A farmer has 50 sheep with different ages, the city requests him to count the sheep with age less than nine months and the number requested must not

exceed 25, which type of loops you think will be useful in this case to help the farmer to find the number of sheep requested by the city. Then plot the results using plot command.

```
%% This code is designed to count sheep numbers per city request
>> clear all;
>> clc;
>> close all;
%% Entering all sheep ages in months
>> Sh=[3 7 8 9 10 12 4 8 5 4 2 13 15 19 24 3.5 4.5 7 6 9 5.5 1 2.5 3.7 6.5 …
      11 7.3 9.2 1.9 5.6 3.8 2.9 1.5 14 18 21 10.5 7.7 8.5 9.4 4.8 3.3 7.5 15 16 …
      12 13.5 17 6.9 20];
>> x=1:25;  % this vector for plotting
>> R=9; % criterion for requested sheep age
>> i=1;n=1;
>> L=length(x);  % criterion for requested number of sheep
>> while i< 51
>>    if Sh(i) <=R
>>       y(n)=Sh(i);
>>       n=n+1;
>>    end
>>   if n> L
>>    break
>>   end
>>  i=i+1;
>>end
>>%% Display the results as a table
>> disp('The following are the sheep ages less than 9 months');
```

```
>> fprintf('Sheep # %4.0f is %4.0f age in month \n',[x;y])
>> %% Plotting the results
>> plot(x,y, ':ob')
>> legend('Sheep ages')
>> xlabel('Numbers'), ylabel('Sheep age (months) label')
>> title('Requested Sheep ages by the City ')
>> axis([0,26,0,10])
```

The results,

The following are the sheep ages less than 9 months

```
Sheep #    1 is    3 age in month
Sheep #    2 is    7 age in month
Sheep #    3 is    8 age in month
Sheep #    4 is    9 age in month
Sheep #    5 is    4 age in month
Sheep #    6 is    8 age in month
Sheep #    7 is    5 age in month
Sheep #    8 is    4 age in month
Sheep #    9 is    2 age in month
Sheep #   10 is    4 age in month
Sheep #   11 is    5 age in month
Sheep #   12 is    7 age in month
Sheep #   13 is    6 age in month
Sheep #   14 is    9 age in month
Sheep #   15 is    6 age in month
Sheep #   16 is    1 age in month
Sheep #   17 is    3 age in month
Sheep #   18 is    4 age in month
```

Sheep # 19 is 7 age in month

Sheep # 20 is 7 age in month

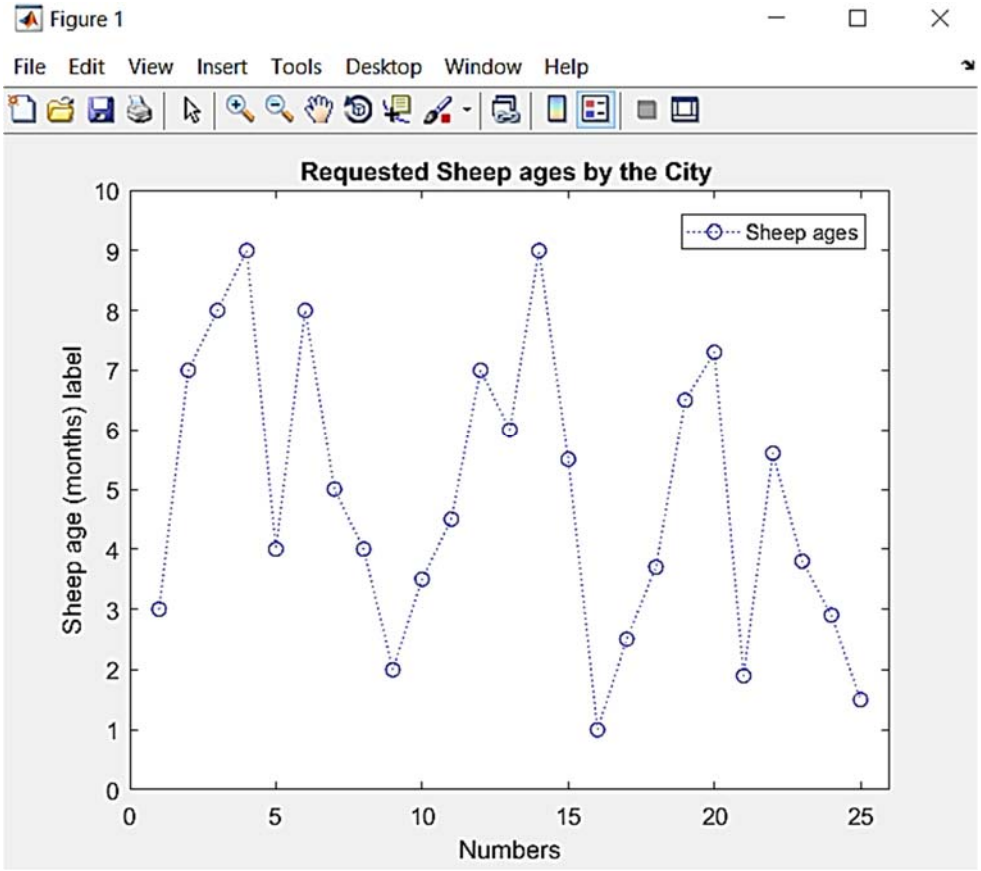
Sheep # 21 is 2 age in month

Sheep # 22 is 6 age in month

Sheep # 23 is 4 age in month

Sheep # 24 is 3 age in month

Sheep # 25 is 2 age in month



Example (2): A Dam in Iraq is discharging water quantities weekly as shown in the table below, the manager of the dam requests from one of the dam engineers to write a MATLAB code to find the quantities that less than 500 (m^3/week) and quantities between (600-800 m^3/week). He also requests the engineer to display the results like a table and plotting them in the same plot.

No.	Water Quantity (m^3/w)	Time (days)
1	300	7
2	350	14
3	250	23
4	400	30
5	750	37
6	550	44
7	375	51
8	650	58
9	450	65
10	900	72
11	200	79
12	675	86
13	600	93
14	150	100
15	800	107
16	850	114

```
>> %%% MATLAB code to find discharge water quantities%%%
>> clear all;
>> clc;
>> close all;
>> Entering Known Variables
>> disch=[300 350 250 400 750 550 375 650 450 900 200 675 600 …
          150 800 850];
>> W=7:7:114;   % Time in days
>> C1=500 % criterion #1
>> C2a=600; C2b=800 % criterion #2
```

```
>> n=1;i=1;j=1;
>> while n<17
>> %% Request # 1 less than 500 m^3/w
>> if disch(n)<= C1
>> Q1(i)=disch(n);
>> w1(i)=W(n);
>> i=i+1;
>> %% Request # 2 between (600-800) m^3/w
>>elseif disch(n)>= C2a && disch(n)<=C2b
>> Q2(j)=disch(n);
>> w2(j)=W(n);
>> j=j+1;
>>end
>> n=n+1;
>> end
>> I=i-1;J=j-1;
>> %% Display the results as a table
>> disp('The quantity #1 less than 500 m^3/week');
>> fprintf('%2.0f # Quantity# %4.0f m^3/w for week #%4.0f \n',[I;Q1;w1])
>> disp('The quantity #2 between (600-800) m^3/week');
>> fprintf('%2.0f # Quantity # %4.0f m^3/w for week #%4.0f \n',[J;Q2;w2])
>> %% Plotting the results
>> plot(w1,Q1, '--sk',w2,Q2, '-.dr')
>> legend('Quantity1', 'Quantity2')
>> xlabel('Time (days)'), ylabel('Water discharge quantity (m^3/w)')
>> title('Graph of discharged water quantities requested by the manager ')
>> axis([0,120,0,1000])
```

The results are,

The quantity #1 less than 500 m^3/week

1 # Quantity# 300 m^3/w for week # 7

2 # Quantity# 350 m^3/w for week # 14

3 # Quantity# 250 m^3/w for week # 21

4 # Quantity# 400 m^3/w for week # 28

5 # Quantity# 375 m^3/w for week # 49

6 # Quantity# 450 m^3/w for week # 63

7 # Quantity# 200 m^3/w for week # 77

8 # Quantity# 150 m^3/w for week # 98

The quantity #2 between (600-800) m^3/week

1 # Quantity # 750 m^3/w for week # 35

2 # Quantity # 650 m^3/w for week # 56

3 # Quantity # 675 m^3/w for week # 84

4 # Quantity # 600 m^3/w for week # 91

5 # Quantity # 800 m^3/w for week # 105