

## CHAPTER 1

### INTRODUCTION TO MATLAB

Matlab is short for Matrix Laboratory, which is a special-purpose computer program optimized to perform engineering and scientific calculations. It started life to perform matrix mathematics, but over the years it has grown into a flexible computing system capable of solving essentially any technical problem. Matlab implements a Matlab programming language and provides an extensive library of predefined functions that make technical programming tasks easier and more efficient.

#### **1 *The Advantage of Matlab***

Matlab has many advantages compared with the conventional programming language like fortran, C or Pascal for technical problem solving. Among of them are the following:

1. Easy to Use. Matlab is interpreted language, so it is not needed compiling when we want to execute the program. Program may be easily written and modified with the built-in integrated development environment and debugged with the Matlab debugger.
2. Platform Independence. Matlab is supported on many computer systems. At the time of this writing, Matlab is supported on Windows 2000/ XP/Vista and many version of UNIX. Programs written in any platform can run on all the other platforms and data files written on any platform may be read on any

other program.

3. **Predefined Functions.** Matlab comes complete with an extensive library of predefined functions that provide tested and prepackaged solutions to many technical problems. For example, suppose that you are writing a program that calculate the statistics associated with input data set. In most languages, you need to write your own subroutines or functions to implement calculations such as arithmetic mean, median, standard deviation and so forth. These and hundreds of functions are built right into Matlab language, making your job easier.
4. **Device Independent Plotting.** Unlike most other languages, Matlab has many integral plotting and imaging commands. The plots and images can be displayed on any graphical output device supported by computer on which Matlab running. This capability makes Matlab an outstanding tool for visualizing technical data.
5. **Graphical User Interface (GUI).** Matlab include tools that allow us to interactively construct a Graphical User Interface (GUI) for his or her program. With capability, the programmer can design a sophisticated data analysis program that can be operated by relatively inexperienced users.

## ***2 The Disadvantage of Matlab***

There are two principal disadvantage of Matlab. First, because of Matlab is interpreted language, therefore Matlab can run slowly compared with compiled language such as pascal, c or fortran. Second, the disadvantage of Matlab is cost, full

copy of Matlab may be 5 to 10 times more expensive than conventional compiler c, pascal or fortran.

### **3 The Matlab Dekstop**

When we start Matlab version 6.5, the special window called The Matlab Dekstop will appear. The default configuration of Matlab Dekstop Matlab is shown in Figure 1.1. It integrates many tools for managing files, variables and application within Matlab environment.

The major tools within Matlab Dekstop are the following:

- a) command window
- b) command history window
- c) launch pad
- d) edit/debug window
- e) figure window
- f) workspace browser and array editor
- g) help browser
- h) current directory browser

## **Exercise**

1. Get help on Matlab function exp using: the “help exp” command typed on the command window and get it with help browser.

2.

## CHAPTER 2

### MATLAB BASICS

#### 1 *Funncion and Constants*

Matlab provides a large number of standard elementary mathematical functions, including `abs`, `sin`, `cos`, `sqrt`, `exp` and so forth. Taking the square root of negative number is not an error, because the appropriate complex number will be produced automatically. Matlab also provide a large number of advanced mathematical function, including `bessel`, `gamma`, `beta`, `erf`, and so forth. For a list of standard elementary mathematical function, type

```
help elfun
```

and for a list of advanced mathematical functions, type

```
help specfun
```

Some functions, like `sqrt`, `cos`, `sin`, `abs` are built in. It means that they are has been compiled, so we just apply them and computational details are not readily accessible.

Matlab also provide some special constants that they may be useful to solve any technical problem. Some constants are the following:

Tabel 2.3 Special Constants

No	Constants	Description
1	pi	3.14159265...

2	i	Imajinary unit, $\sqrt{-1}$
3	j	Same as i
4	eps	Floating-point relative precision, $10^{-52}$
5	realmin	Smallest floating-point number
6	realmax	Largest floating-point number
7	inf	Infinite number
8	NaN	Not-a-Number

```
>> pi
ans =
    3.1416

>> i
ans =
    0 + 1.0000i

>> j
ans =
    0 + 1.0000i

>> realmin
ans =
    2.2251e-308

>> realmax
ans =
    1.7977e+308

>> eps
ans =
```

```
2.2204e-016
>> 1/0
Warning: Divide by zero.
ans =
    Inf
>> 0/0
Warning: Divide by zero.
ans =
    NaN
```

## 2 Using Meshgrid

Meshgrid is used to generate X and Y matrix for three dimensional plot. The syntax of meshgrid are

```
[X,Y] = meshgrid(x,y)
```

```
[X,Y] = meshgrid(x)
```

```
[X,Y,Z] = meshgrid(x,y,z)
```

`[X,Y]=meshgrid(x,y)` transforms the domain specified by vector `x` and `y` into arrays `X` and `Y`, which can be used to evaluate the function of two variables and three dimensional mesh/surface plots. The rows of the output array `X` are copies of vector `x`, and the column of the array `Y` are copies of the vector `y`.

Ex.

```
[X,Y] = meshgrid(1:3,10:14)
```

X =

1 2 3

1 2 3

1 2 3

1 2 3

1 2 3

Y =

10 10 10

11 11 11

12 12 12

13 13 13

14 14 14

### Contoh

Plot the function graphic of  $z = x^2 - y^2$  specified by domain  $0 \leq x \leq 5$  dan  $0 \leq y \leq 0$

### Solution

Firstly, we must specify the grids on the surface x-y using meshgrid function

```
>> x=0:5;
```

```
>> y=0:5;
```



```
>> [X Y]=meshgrid(x,y)
```

```
X =
```

```
0 1 2 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5
0 1 2 3 4 5
```

```
Y =
```

```
0 0 0 0 0 0
1 1 1 1 1 1
2 2 2 2 2 2
3 3 3 3 3 3
4 4 4 4 4 4
5 5 5 5 5 5
```

Then, the values of  $z$  can be obtained by replacing  $x$  and  $y$  in the original function into  $X$  and  $Y$

```
>> z=X.^2-Y.^2
```

```
z =
```

```
0 1 4 9 16 25
-1 0 3 8 15 24
-4 -3 0 5 12 21
-9 -8 -5 0 7 16
-16 -15 -12 -7 0 9
-25 -24 -21 -16 -9 0
```

Finally, we have the graphic of the function

```
>> mesh(X,Y,z)
```

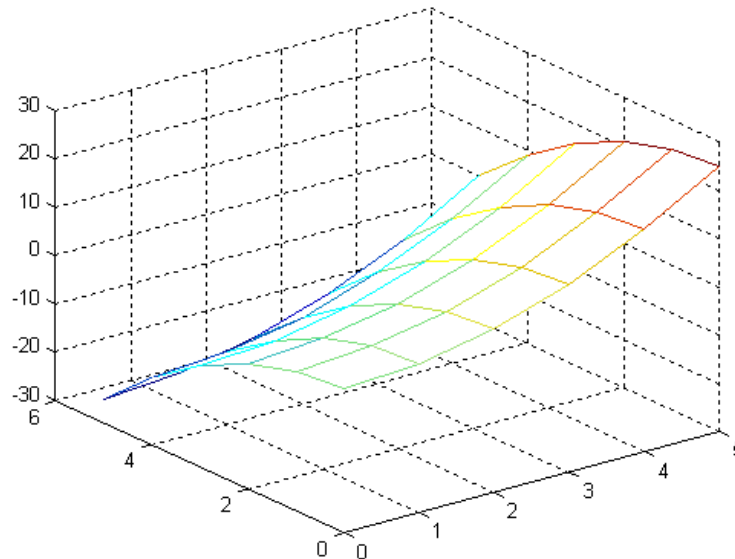


Illustration 1:

### 3 Special Elementar Function Matlab

Matlab has a large number of special elementary function which may be useful for numerical calculation. In this chapter, we will discuss some of them.

#### *feval()*

Function `feval()` is used to evaluate a function. For example, if we have a function  $f(x) = x^2 + 2x + 1$  and we evaluate the function at  $x=3$

```
>> f=inline('x^2+2*x+1','x');
```

```
>> f(3)
```

```
ans =
```

```
16
```

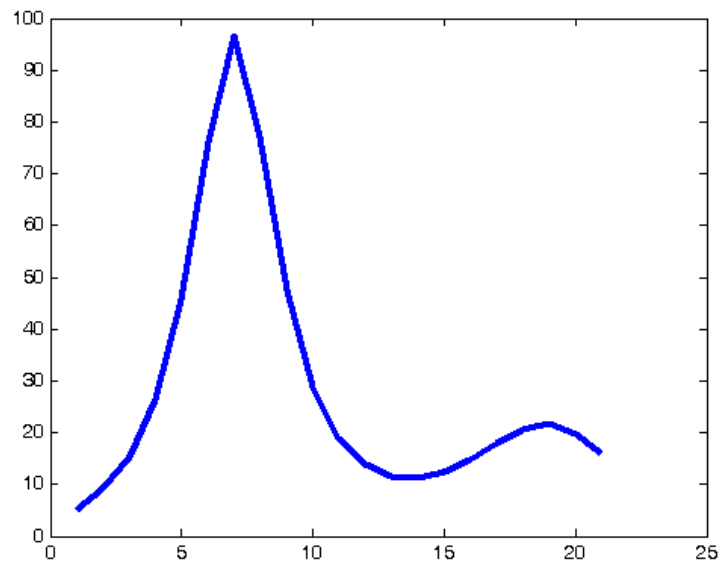
If we use the function provided by Matlab called *humps*. To evaluate the function, we must create a handle function by using @ sign.

```
>> fhandle=@humps;
```

```
>> feval(fhandle,1)
```

```
ans =
```

```
16
```



Gambar 2.1 Fungsi humps

## Polyval

Function polyval is used to specify the value of the polynomial of the form

$$p(x) = a_0 + a_1x^1 + a_2x^2 + a_3x^3 + a_4x^4 + \dots + a_{n-1}x^{n-1} + a_nx^n$$

Matlab has a simply way to express the form of the polynomial by

$$p = [ a_n \ a_{n-1} \ \dots \ a_3 \ a_2 \ a_1 \ a_0 ]$$

### Example

Given a polynomial  $p(x) = x^4 + 3x^2 + 4x + 5$ . It will be evaluated at  $x = 2, -3$  and 4.

### Solution

- First, we type the polynomial by simply way  $p = [1 \ 0 \ 3 \ 4 \ 5]$ .
- Second, we type the point we are going to evaluate  $x = [2, -3, 4]$
- Third, Evaluate the polynomial at point x by command  $\text{polyval}(p, x)$

If we write in the command window

```
>> p=[1 0 3 4 5];
```

```
>> x=[2,-3,4];
```

```
>> polyval(p,x)
```

```
ans =
```

```
41 101 325
```

*Polyfit*

*Fungsi polyder*

*Fungsi poly*

*Fungsi conv*

Fungsi deconv

## CHAPTER 3

### VECTOR AND MATRICES

Vector is one-dimensional array of numbers. Vector can be a coulomn or row. Matlab can create a coulomn vector by enclosing a set of numbers sparated by semicolon. For example, to create a coulomn vector with three elements we write:

```
>> a=[1;2;3]
```

```
a =
```

```
1
```

```
2
```

```
3
```

To create a row vector, we enclose a set of numbers in square brackets, but this time we use coma or blank space to delimitate the elements.

```
>> a=[4,5,6]
```

```
a =
```

```
4 5 6
```

```
>> a=[4 5 6]
```

```
a =
```

```
4 5 6
```

A column vector can be turned into a row vector by three ways. The first, we can turn using the transpose operation.

```
>> v=[3,2,1];
```

```
>> v'
```

```
ans =
```

```
3
```

```
2
```

```
1
```

Secondly, we can create a column vector by enclosing a set of numbers with semicolon notation to delimit the numbers.

```
>> w=[3;2;1]
```

```
w =
```

```
3
```

```
2
```

```
1
```

It is also possible to add or subtract two vectors to produce a new vector. In order to perform this operation, the two vectors must both be the same type and the same length. So, we can add two column vectors together to produce a new column vector or we can add two row vectors to produce a new row vector. Let's add two column vectors together:

```
>> p=[-1;2;3;1];
```

```
>> q=[2;1;-3;-2];
```

```
>> p+q
```

```
ans =
```

```
1
```

```
3
```

```
0
```

```
-1
```

Now, let's subtract one row vector from another:

```
>> p=[-1,2,3,1];
```

```
>> q=[2,1,-3,-2];
```

```
>> p-q
```

```
ans =
```

```
1 3 0 -1
```

## **1 CREATING LARGER VECTOR FROM EXISTING VARIABLES**

Matlab allow us to append vectors together to create a new one. Let u and v are two column vectors with m and n elements respectively that we have created in Matlab. We can create third vector w whose first m elements of vector u and whose second n elements of vector v. In order to create the vector w, it is done by writing



[u;v].

```
>> u=[1;2;3];
```

```
>> v=[5;6];
```

```
>> w=[u;v]
```

```
w =
```

```
1
```

```
2
```

```
3
```

```
5
```

```
6
```

This can also be done with row vector. If we have two row vectors r and s with m and n elements respectively. So, a new row vector can be created by writing [r,s].

```
>> r=[1,2,3];
```

```
>> s=[5,6,7,8];
```

```
>> [r,s]
```

```
ans =
```

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

## 2 **CREATING A VECTOR WITH UNIFORMLY SPACED ELEMENTS**

Matlab allow us to create a vector with elements that are uniformly spaced

by increment  $q$ . To create a vector  $x$  that have uniformly spaced elements with first element  $a$ , final element  $b$  and stepsize  $q$  is

$$x = [ a : q : b ];$$

We can create a list of even number from 0 to 10 with uniformly spaced elements:

```
>> x=[0:2:10]
```

```
x =
```

```
0 2 4 6 8 10
```

Again, let's we create a list of small number starts from 0 to 1 with uniformly space 0.1;

```
>> x=0:0.1:1
```

```
x =
```

```
Columns 1 through 7
```

```
0 0.1000 0.2000 0.3000 0.4000 0.5000 0.6000
```

```
Columns 8 through 11
```

```
0.7000 0.8000 0.9000 1.0000
```

The set of  $x$  values can be used to create a list of points representing the values of some given function. For example, suppose that whave  $y=e^x$  , then we have

```
>> y=exp(x)
```

```
y =
```

```
Columns 1 through 7
```

```
1.0000  1.1052  1.2214  1.3499  1.4918  1.6487  1.8221
```

```
Columns 8 through 11
```

```
2.0138  2.2255  2.4596  2.7183
```

But, if we have a function  $y=x^2$ , we can not obtain a list of values representing the function without adding dot notation.

```
>> x^2
```

```
??? Error using ==> ^
```

```
Matrix must be square.
```

If we use dot notation behind the variable x;

```
>> x.^2
```

```
ans =
```

```
Columns 1 through 7
```

```
0  0.0100  0.0400  0.0900  0.1600  0.2500  0.3600
```

```
Columns 8 through 11
```

```
0.4900  0.6400  0.8100  1.0000
```

Matlab also allow us to create a row vector with m uniformly spaced element by typing

```
linspace(a,b,m)
```

```
>> linspace(0,2,5)
```

```
ans =
```

```
0 0.5000 1.0000 1.5000 2.0000
```

Matlab also allow us to create a a row vector with n logaritmicly spaced elements by typing

```
logspace(a,b,n)
```

```
>> logspace(1,3,5)
```

```
ans =
```

```
1.0e+003 *
```

```
0.0100 0.0316 0.1000 0.3162 1.0000
```

### 3 *Characterizing Vector*

The *length* command returns a number of a vector elements, for example:

```
>> a=[0:5];
```

```
>> length(a)
```

```
ans =
```

```
6
```

We can find the largest and smallest of a vector by max and min command, ffor example:

```
>> a=[0:5];
```

```
>> max(a)
```

```
ans =
```

```
5
>> min(a)
ans =
0
```

### Accessing the array elements

- If we have array  $a=[1,2,3,4;5,6,7,8;6,7,8,9]$ . To display all of array elements at the 3th row, we can type

```
>> a(3,:)
ans =
    6    7    8    9
```

The colon notation means all of array elements.

- For example, if we want to display all of array elements at the first and second column then we can type

```
>> a(:,[1 2])
ans =
    1    2
    5    6
    6    7
```

- To access the array elements  $a$  at the 1<sup>st</sup> dan 2<sup>nd</sup> row, and 3<sup>rd</sup> and 4<sup>th</sup> column, we can type

```
>> a(1:2,3:4)
```

```
ans =
```

```
3 4
```

```
7 8
```

- If we wish to replace all of the array elements at 2<sup>nd</sup> and 3<sup>rd</sup> row and 1<sup>st</sup> and 2<sup>nd</sup> column with the value equal to 1

```
>> a(2:3,1:2)=ones(2)
```

```
a =
```

```
1 2 3 4
```

```
1 1 7 8
```

```
1 1 8 9
```

- We are able to create the table using the colon operator. For example, if we wish to create the sinus table starting from a given angle and step size 30°

```
>> x=[0:30:180]';
```

```
>> trig(:,1)=x;
```

```
>> trig(:,2)=sin(pi/180*x);
```

```
>> trig(:,3)=cos(pi/180*x);
```

```
>> trig
```

```
trig =
```

```
0 0 1.0000
```

```

30.0000  0.5000  0.8660
60.0000  0.8660  0.5000
90.0000  1.0000  0.0000
120.0000 0.8660 -0.5000
150.0000 0.5000 -0.8660
180.0000 0.0000 -1.0000

```

- The Colon operator can be used to perform operation at Gauss elimination.

For example

```
>> a=[-1,1,2,2;8,2,5,3;10,-4,5,3;7,4,1,-5];
```

```
>> a(2,:)=a(2,:)-a(2,1)/a(1,1)*a(1,;)
```

a =

```

-1.00    1.00    2.00    2.00
     0   10.00   21.00   19.00
10.00   -4.00    5.00    3.00
 7.00    4.00    1.00   -5.00

```

```
>> a(3,:)=a(3,:)-a(3,1)/a(1,1)*a(1,;)
```

a =

```

-1.00    1.00    2.00    2.00
     0   10.00   21.00   19.00

```

```
0    6.00    25.00    23.00
7.00    4.00    1.00    -5.00
```

```
>> a(4,:)=a(4,:)-a(4,1)/a(1,1)*a(1,:)
```

```
a =
```

```
-1.00    1.00    2.00    2.00
0    10.00    21.00    19.00
0    6.00    25.00    23.00
0    11.00    15.00    9.00
```

```
>> a(3,:)=a(3,:)-a(3,2)/a(2,2)*a(2,:)
```

```
a =
```

```
-1.00    1.00    2.00    2.00
0    10.00    21.00    19.00
0    0    12.40    11.60
0    11.00    15.00    9.00
```

```
>> a(4,:)=a(4,:)-a(4,2)/a(2,2)*a(2,:)
```

```
a =
```

```
-1.00    1.00    2.00    2.00
0    10.00    21.00    19.00
0    0    12.40    11.60
```



```

0      0      -8.10   -11.90

>> a(4,:)=a(4,:)-a(4,3)/a(3,3)*a(3,:)

a =

-1.00    1.00    2.00    2.00
0     10.00   21.00   19.00
0      0     12.40   11.60
0      0      0     -4.32

```

- The keyword *end* states the final element of the array elements. For example, if we have a vector

```

>> a=[1:6];

>> a(end)

ans =

6

>> sum(a(2:end))

ans =

20

```

- The colon operator can play as a single subscript. For a special case, the colon operator can be used to replace all of the array elements

```

>> a=[1:4;5:8]

a =

```

```
1  2  3  4
5  6  7  8

>> a(:)=-1

a =

-1 -1 -1 -1
-1 -1 -1 -1
```

### Replication of row and column

Sometimes, we need to generate the array elements at a given row and column to any other row and column. For this purpose, we need Matlab command *repmat* to replicate the row/column elements.

```
>> a=[1;2;3];

>> b=repmat(a,[1 3])

b =

1  1  1
2  2  2
3  3  3
```

The command `repmat(a,[1 3])` means that “replicate vector a into one row and three columns.

```
>> c=repmat(a,[2 1])

c =
```

```
1
```

```
2
```

```
3
```

```
1
```

```
2
```

```
3
```

The command `repmat(a,[2 1])` means that “replicate vector a into two rows and 1 column.

The alternative command `repmat (a,[1 3])` is `repmat(a,1,3)` and `repmat (a,[2 1])` is `repmat(a,2,1)`.

### Deleting rows and columns

We can use the colon operator and the blank array to delete the array elements.

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

```
b =
```

```
1  2  3
```

```
3  4  5
```

```
6  7  8
```

```
>> b(:,2)=[]
```

```
b =
```

```
1 3
```

```
3 5
```

```
6 8
```

If we want to delete the array elements at 2<sup>nd</sup> and 3<sup>rd</sup> column, we just type

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

```
>> b(:,[2 3])=[]
```

```
b =
```

```
1
```

```
3
```

```
6
```

Similarly, if we want to delete the array elements at 2<sup>nd</sup> and 3<sup>rd</sup> rows

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

```
>> b([2 3],:)=[]
```

```
b =
```

```
1 2 3
```

## Array manipulation

Below are some functions applied to manipulate an array,

- *diag*, the function is used to create an diagonal array. If we have a vector *v* then *diag(v)* will create a diagonal array with diagonal elements of *v*.

```
>> v=[1:4];  
  
>> diag(v)  
  
ans =  
  
    1    0    0    0  
    0    2    0    0  
    0    0    3    0  
    0    0    0    4
```

To move right or move down the diagonal elements, we just type *diag(v,b)*

```
>> diag(v,2)  
  
ans =  
  
    0    0    1    0    0    0  
    0    0    0    2    0    0  
    0    0    0    0    3    0  
    0    0    0    0    0    4  
    0    0    0    0    0    0  
    0    0    0    0    0    0
```

```
>> diag(v,-2)  
  
ans =  
  
    0    0    0    0    0    0  
    0    0    0    0    0    0
```

```
1 0 0 0 0 0
0 2 0 0 0 0
0 0 3 0 0 0
0 0 0 4 0 0
```

- *fliplr*, perintah ini digunakan untuk mempertukarkan elemen-elemen matriks yang berada di sebelah kanan dengan elemen-elemen yang berada di sisi kiri.

```
>> a=[1,2,3;4,5,6]'
```

```
a =
```

```
1 4
```

```
2 5
```

```
3 6
```

```
>> fliplr(a)
```

```
ans =
```

```
4 1
```

```
5 2
```

```
6 3
```

Kalau a berupa vektor, maka seperti terlihat di bawah ini

```
>> a=[1,2,3,4,5,6]
```

```
a =
```

```
1 2 3 4 5 6
```

```
>> fliplr(a)

ans =

     6     5     4     3     2     1
```

- *flipud*, fungsi ini digunakan untuk mempertukarkan elemen matriks yang berada di atas dengan yang ada di bawah.

```
>> a=[1,2,3;4,5,6]

a =

     1     2     3
     4     5     6

>> flipud(a)

ans =

     4     5     6
     1     2     3
```

- *rot90*, fungsi ini digunakan untuk merotasikan matriks A sebesar  $90^\circ$  menentang arah dengan arah jarum jam.

```
>> a=[1,2,3;4,5,6]

a =

     1     2     3
     4     5     6

>> rot90(a)
```

```
ans =  
  
    3    6  
  
    2    5  
  
    1    4
```

- *tril*, fungsi ini digunakan untuk menentukan elemen segitiga bawah dari matriks tertentu. Sedangkan *triu*, untuk menentukan elemen segitiga atas matriks.

```
>> a=[1,2,3;4,5,6]
```

```
a =  
  
    1    2    3  
  
    4    5    6
```

```
>> tril(a)
```

```
ans =  
  
    1    0    0  
  
    4    5    0
```

```
>> triu(a)
```

```
ans =  
  
    1    2    3  
  
    0    5    6
```



## Fungsi matriks lainnya

Masih ada banyak fungsi yang dapat digunakan untuk manipulasi matriks. Beberapa diantaranya

- *det*, fungsi ini digunakan untuk menentukan determinan matriks. Ingat, bahwa matriks yang memiliki determinan hanyalah matriks bujur sangkar.

```
>> a=[1,2,3;4,3,-2;-1,5,2];
```

```
>> det(a)
```

```
ans =
```

```
73
```

- *eig*, ini digunakan untuk menentukan nilai eigen.

```
>> a=[1,2,3;4,3,-2;-1,5,2];
```

```
>> eig(a)
```

```
ans =
```

```
5.5031
```

```
0.2485 + 3.6337i
```

```
0.2485 - 3.6337i
```

- *inv*, fungsi ini digunakan untuk melakukan invers matriks seperti telah dijelaskan di atas.
- *lu*, adalah fungsi untuk melakukan dekomposisi matriks menjadi matriks segitiga bawah dan matriks segitiga atas.

```
>> a=[1,2,3;4,3,-2;-1,5,2];
```

```
>> [L,U]=lu(a)
```

L =

0.25	0.22	1.00
1.00	0	0
-0.25	1.00	0

U =

4.00	3.00	-2.00
0	5.75	1.50
0	0	3.17

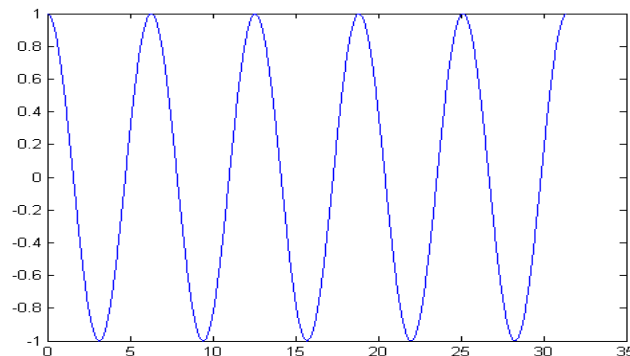
## CHAPTER 4

## INTRODUCTION TO GRAPHICS

**Basic 2-D graphs**

Graphs in 2-d are drawn with the *plot* statement. Axes are automatically created and scaled to include the minimum and maximum data points. The most common form of plot is *plot(x,y)* where x and y are vectors with the same length. For example

```
x=0:pi/200:10*pi;  
y=cos(x);  
plot(x,y)
```



Picture 4.2 Graph of x vs y

We may also use the *linspace* command to specify the function domain, so the previous scripts can be written as

```
x=linspace(0,10*pi,200);
```

```
y=cos(x);  
plot(x,y)
```

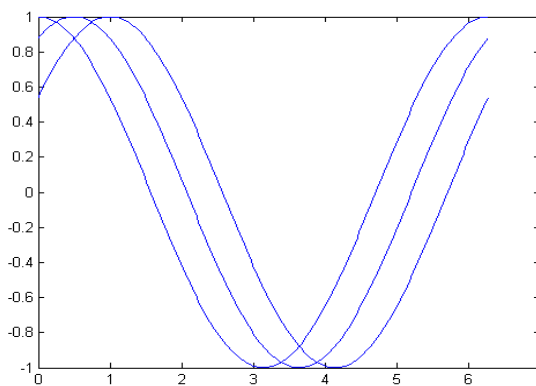
### Multiple plot on the same axes

There are at least two ways of drawing multiple plots on the same axes. The ways are followings:

1. The simplest way is to use *hold* to keep the current plot on the axes. All subsequent plots are added to the axes until *hold* is released. For example,

```
x=linspace(0,2*pi,200);
```

```
y1=cos(x);  
plot(x,y1);  
hold;  
y2=cos(x-0.5);  
plot(x,y2);  
y3=cos(x-1.0);  
plot(x,y3)
```



Picture 4.2 Creating multiple graphs on the same axes with *hold*

2. The second way is to use plot with multiple arguments, e.g

```
plot(x1,y1,x2,y2,x3,y3,...)
```

plot the vector pairs  $(x_i, y_i)$ ,  $(x_2, y_2)$ ,  $(x_3, y_3)$ , etc. For example,

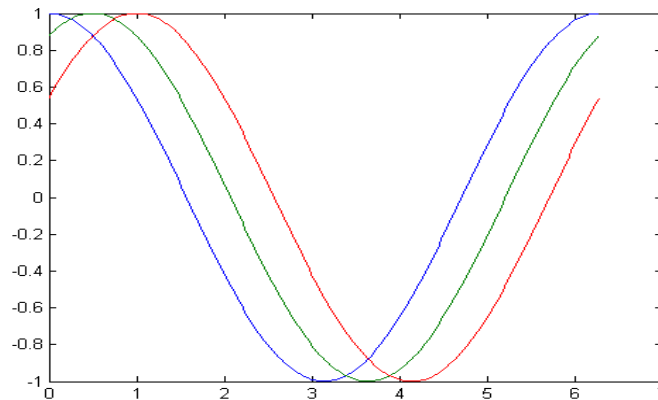
```
x=linspace(0,2*pi,200);
```

```
y1=cos(x);
```

```
y2=cos(x-0.5);
```

```
y3=cos(x-1.0);
```

```
plot(x,y1,x,y2,x,y3)
```



Picture 4.3 Three graphs on the one axes

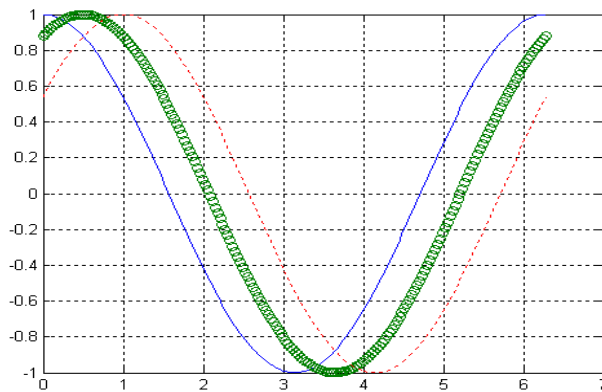
### Line style, markers and color

Line style, markers and color may be selected for a graph with a string argument to plot. The general form of using line style, markers and color for a graph is

```
plot(x,y,'LineStyle_Marker_Color')
```

For example

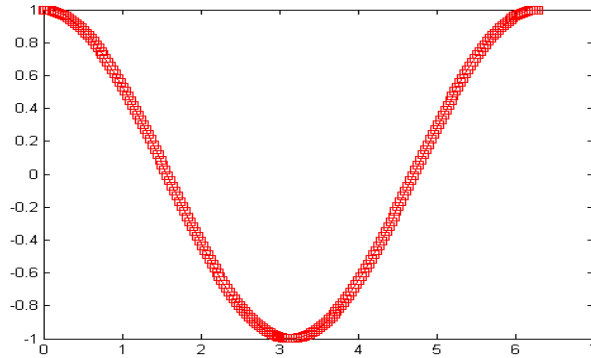
```
x=linspace(0,2*pi,200);  
y1=cos(x);  
y2=cos(x-0.5);  
y3=cos(x-1.0);  
plot(x,y1,'-',x,y2,'o',x,y3,':')  
grid
```



Picture 4.4. Displaying three graphs with different line style

Notice that picture 4.4 is displayed with grids. The grids can be accompanied on the graph with *grid command*.

```
x=linspace(0,2*pi,200);  
y=cos(x);  
plot(x,y,'-squares')
```



Picture 4.5. The graph is displayed with line style, marker and color

We can specify color and line width on the graph with commands:

- LineWidth: specify line width of the graph
- MarkerEdgeColor: specify the marker color and edge color of the graph.
- MarkerFaceColor: specify the face color of the graph.
- MarkerSize: specify the size of the graph

```
x = -pi:pi/10:pi;
```

```
y = tan(sin(x)) - sin(tan(x));
```

```
plot(x,y,'--rs','LineWidth',3,...
```

```
    'MarkerEdgeColor','k',...
```

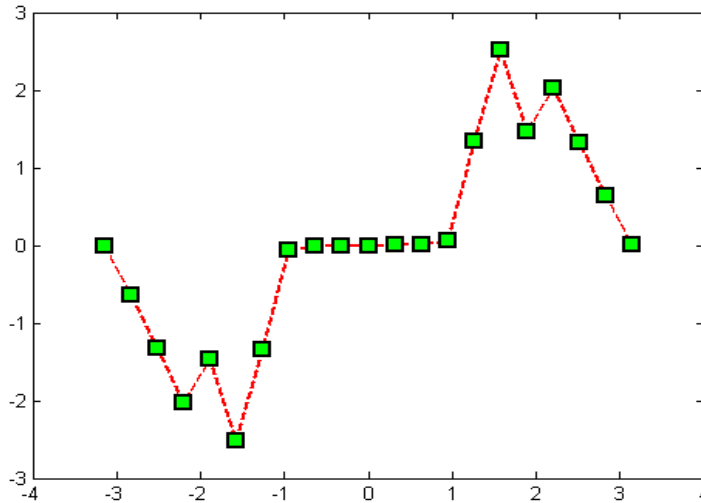
```
    'MarkerFaceColor','g',...
```

```
    'MarkerSize',5)
```

The previous script will display a graph y vs x with

- Line style is dash with red color and the marker is square('--rs'),
- Line width is 3
- Color of the edge marker is black(k),

- The face marker is green (g),
- The size of marker is 5



Picture 4.6. The graph is displayed with line style dash, line width 3, followed by square marker with fill color is green and the edge color is black with size of square marker is 5.

### Adding Label, Legend and Title of the Graph

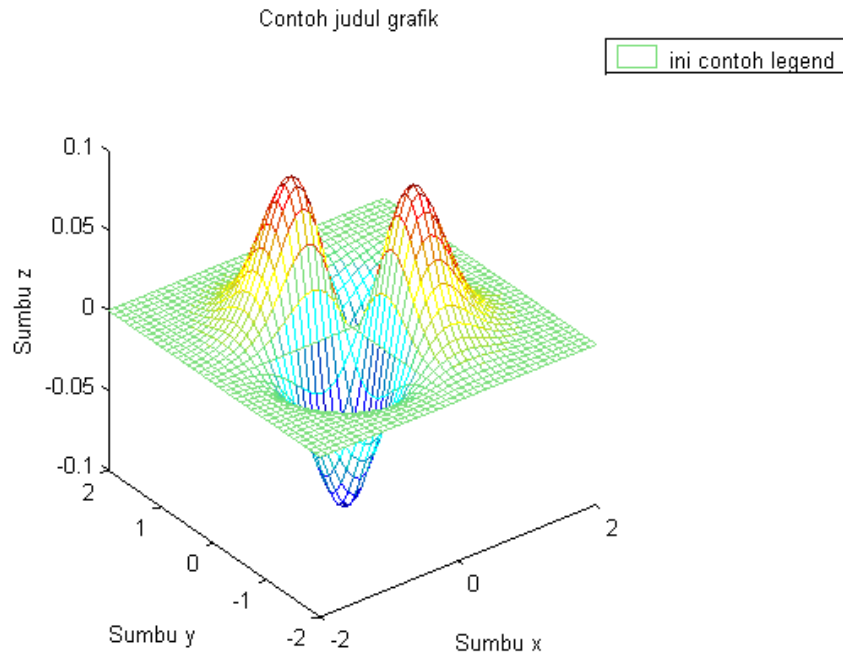
It is very important to add label on the axes of the graph. As it can be used to make easy understanding the meaning of the graph. The commands that are common :

- xlabel : to add label on the absis (x axes)
- ylabel : to add label on the ordinat (y axes)
- zlabel : to add label on the absis (z axes)
- title : to add title of the graph
- legend : to add legend of the graph



For example, notice the script below

```
clear; close all;
x=-2:0.1:2;
y=-2:0.1:2;
[X,Y]=meshgrid(x,y);
f=-X.*Y.*exp(-2*(X.^2+Y.^2));
mesh(X,Y,f);
xlabel('Sumbu x');
ylabel('Sumbu y');
zlabel('Sumbu z');
title('Contoh judul grafik');
legend('ini contoh legend')
```



Picture 4.12

### Adding Text on the Graph

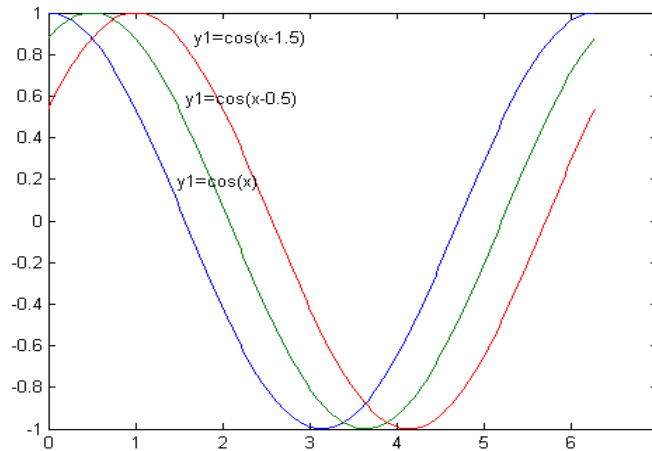
Sometime, we need to add any text to make clearly if there are more than one graph on the one axes. To add some texts one the graph, we can use `gtext()`. For example

```
clear; close all;  
x=linspace(0,2*pi,200);  
y1=cos(x);  
y2=cos(x-0.5);  
y3=cos(x-1.0);  
plot(x,y1,x,y2,x,y3);
```

```

gtext('y1=cos(x)');gtext('y1=cos(x-0.5)');
gtext('y1=cos(x-1.5)');

```



Gambar 4.12 The graph with adding text

We can also add the adding text on the graph with the way as below

```

clear; close all;
x=linspace(0,2*pi,200);
y1=cos(x);
y2=cos(x-0.5);
y3=cos(x-1.0);
plot(x,y1,x,y2,x,y3)
text(pi/2,cos(pi/2),' \leftarrow y1=cos(x)');
text(pi/3,cos(pi/3-0.5),' \leftarrow y1=cos(x-0.5)');
text(2*pi/3,cos(2*pi/3-1.0),' \leftarrow y1=cos(x-1.0)')

```

## CHAPTER 6

### FLOW CONTROL

There are eight flow control statements in MATLAB:

1. `if`, together with `else` and `elseif`, executes a group of statements based on some logical condition.
2. `switch`, together with `case` and `otherwise`, executes different groups of statements depending on the value of some logical condition.
3. `while` executes a group of statements an indefinite number of times, based on some logical condition.
4. `for` executes a group of statements a fixed number of times.
5. `continue` passes control to the next iteration of a `for` or `while` loop, skipping any remaining statements in the body of the loop.
6. `break` terminates execution of a `for` or `while` loop.
7. `try...catch` changes flow control if an error is detected during execution.
8. `return` causes execution to return to the invoking function. All flow constructs use `end` to indicate the end of the flow control block.

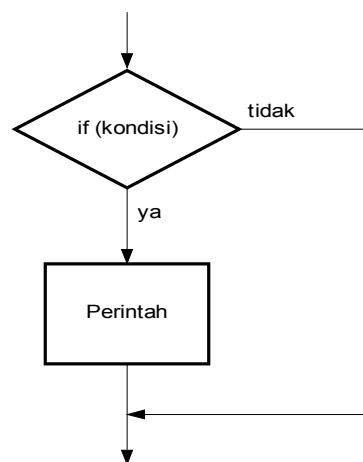
`if`, `else`, and `elseif` evaluates a logical expression and executes a group of statements based on the value of the expression. In its simplest form, its syntax is

```
if (logical_expression)
    statements
end
```

If the logical expression is true (1), MATLAB executes all the statements between the if and end lines. It resumes execution at the line following the end statement. If the condition is false (0), MATLAB skips all the statements between the if and end lines, and resumes execution at the line following the end statement. For example,

```
if mod(a,2) == 0
    disp('a is even')
    b = a/2;
end
```

The if statement can be described in the diagram as picture 3 below



Gambar 6.1 Diagram alir percabangan if

Generally, the condition is logical expression, that is, the expression consisting of relational operator that is true (1) or false (0). The followings are some relational operators:

<b>Relational Operator</b>	<b>Meaning</b>
<	Less than
<=	Less than or equal
>	Greater than
>=	Greater than or equal
==	Equal to
~=	Not equal

The followings are some examples of logical expression using relational operator and its meaning

$$b^2 - 4ac < 0 \quad (b^2 - 4ac) < 0$$

$$b^2 > 4ac \quad b^2 > 4ac$$

$$b^2 - 4ac == 0 \quad (b^2 - 4ac) = 0$$

$$b \neq 4 \quad (b \neq 4)$$

You can nest any number of if statements.

The else and elseif statements further conditionalize the if statement: The else statement has no logical condition. The statements associated with it execute if the preceding if (and possibly elseif condition) is false (0). The elseif statement has a logical condition that it evaluates if the preceding if (and possibly elseif condition) is false (0). The statements associated with it execute if its logical condition is true (1).

You can have multiple elseifs within an if block. if n < 0                      % If n negative, display error message.

```
disp('Input must be positive');  
elseif rem(n,2) == 0 % If n positive and even, divide by 2.  
    A = n/2;  
else  
    A = (n+1)/2; % If n positive and odd, increment and divide.  
end  
if Statements and Empty Arrays  
An if condition that reduces to an empty array  
represents a false condition. That is, if A  
    S1  
else  
    S0  
end  
will execute statement S0 when A is an empty array.
```