

2.0 MATLAB Fundamentals

2.1 INTRODUCTION

MATLAB is a computer program for computing scientific and engineering problems that can be expressed in mathematical form. The name MATLAB stands for MATrix LABoratory, because the program is designed to make matrix computations very easy. Unlike other computer programs such as FORTRAN, QBASIC, etc, MATLAB does not require dimensioning since the data element is an array.

MATLAB contains application-specific solutions called **toolboxes**. MATLAB toolboxes are collections of MATLAB functions (i.e M-files) in order to extend the MATLAB computational capability for specific classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

MATLAB can also handle logical statements, but the way they are being expressed might be different from that of other programming languages. For example, DO statement is express as WHILE(1) in MATLAB. Because MATLAB interprets the number 1 as corresponding to “true,” this statement will repeat infinitely in the same manner as the DO statement. The loop is terminated with a *break* command. This command transfers control to the statement following the *end* statement that terminates the loop.

2.2 MATLAB Environment

2.2.1 Development Environment: This is a set of tools and facilities that ensure the use MATLAB functions and files. Many of these tools are graphical user interfaces. It includes the MATLAB desktop and Command Window, command history, and browsers for viewing help, the workspace, files, and the search path. Figure 2.1 shows MATLAB development environment.

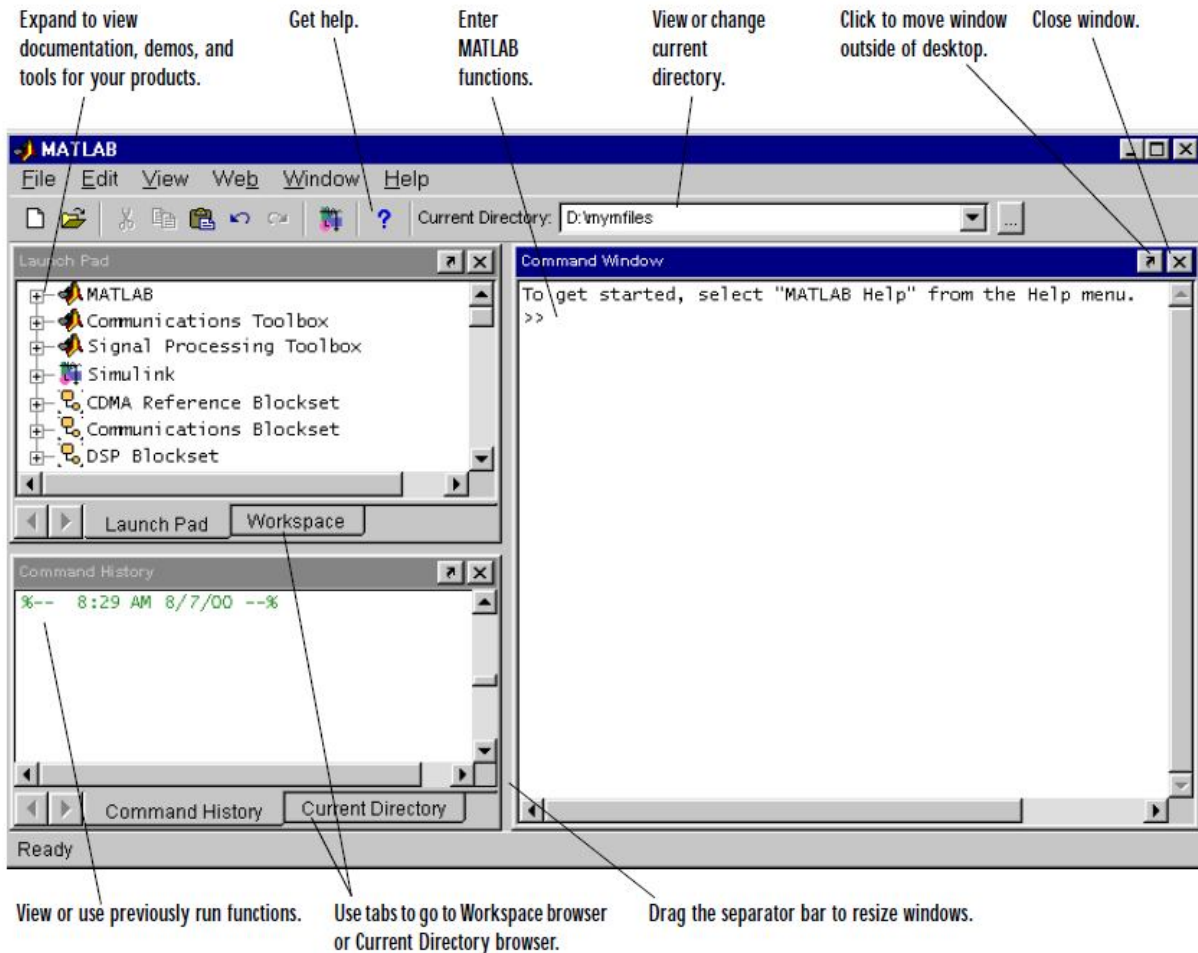


Figure 2.1: MATLAB development environment

2.2.2 MATLAB Mathematical Function Library: This is a collection of computational algorithms ranging from elementary functions like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix eigenvalues, Bessel functions, and Fourier transforms.


2.2.3 MATLAB Language: This is a matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features.

2.2.4 MATLAB Graphics: This contains functions that help to represent data or information in a graphical manner. It includes commands for two-dimensional and three-dimensional data visualization, image processing, animation, and graphics presentations. It also includes commands that allow user to fully customize the appearance of graphics as well as to build complete graphical user interfaces on MATLAB applications.

2.2.5 MATLAB Application Program Interface (API): This is a library that allows one to write C and FORTRAN programs that can interact with MATLAB. It include facilities for calling routines from MATLAB (dynamic linking), and calling MATLAB as a computational engine.

2.3 Starting and Quitting MATLAB

2.3.1 Starting MATLAB

To stat MATLAB, double-click the MATLAB shortcut icon  on your Windows desktop. After starting, the MATLAB opens shown in Fig 3.1.

2.3.2 Quitting MATLAB

To end MATLAB session, select **Exit MATLAB** from the **File** menu in the desktop, or type quit in the Command Window.

2.4 Desktop Tools

1. Command Window

Command Window is use to enter variables and run functions and M-files as well as displaying output (Fig 3.2).

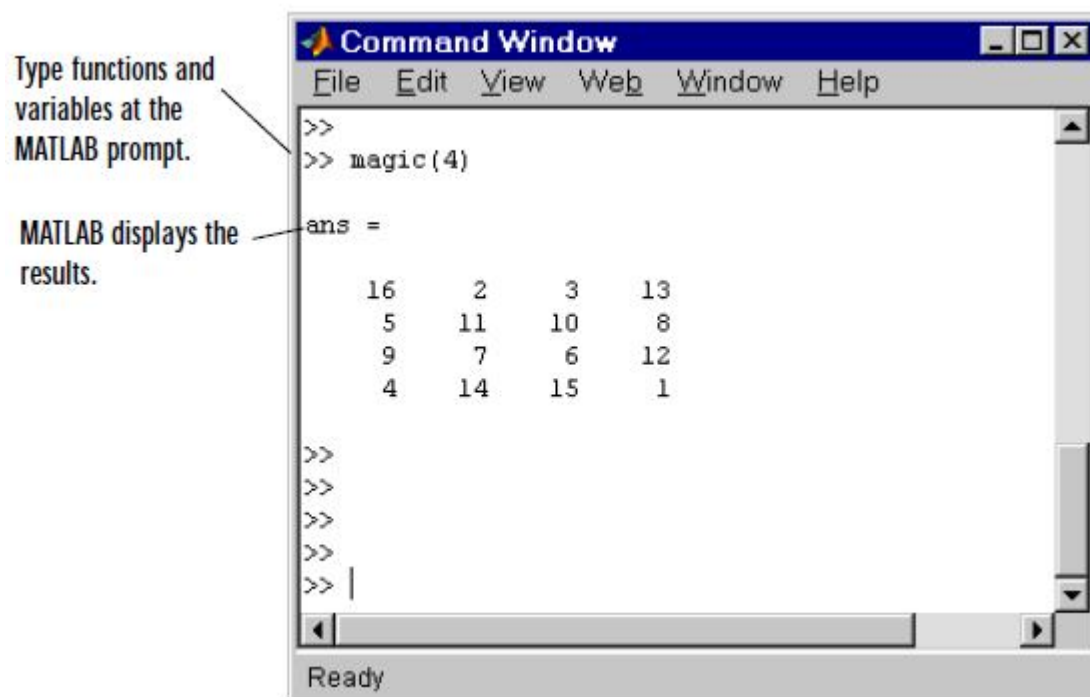


Fig 3.2 Command window

2. Command History

Input data enter in the command window are logged in the Command History window. In the Command History, the previously used functions can be view and copy to command window for execution if there is need to do so (Fig 3.3).

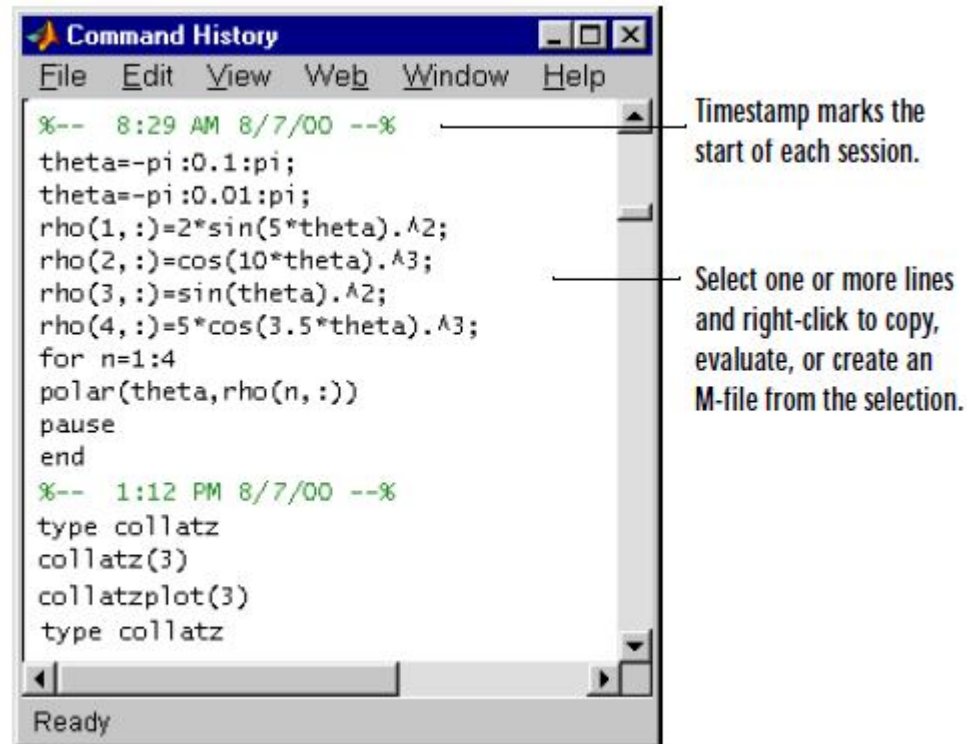


Fig 3.3 Command History

3. Launch Pad

MATLAB's **Launch Pad** provides easy access to tools, demos, and documentation (Fig 3.4).

Sample of listings in Launch Pad – you'll see listings for all products installed on your system.

Help - double-click to go directly to documentation for the product.

Demos - double-click to display the demo launcher for the product.

Tools - double-click to open the tool.

Click + to show the listing for a product.

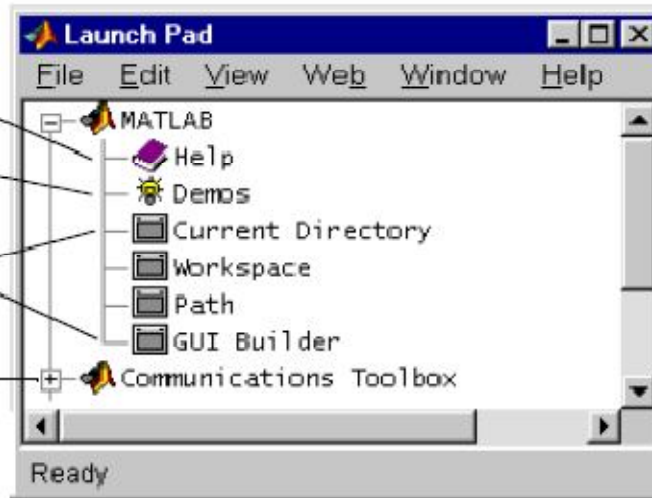


Fig.3.4 Launch Pad

Use the close box to hide the pane.

Drag the separator bar to adjust the width of the panes.

View documentation in the display pane.

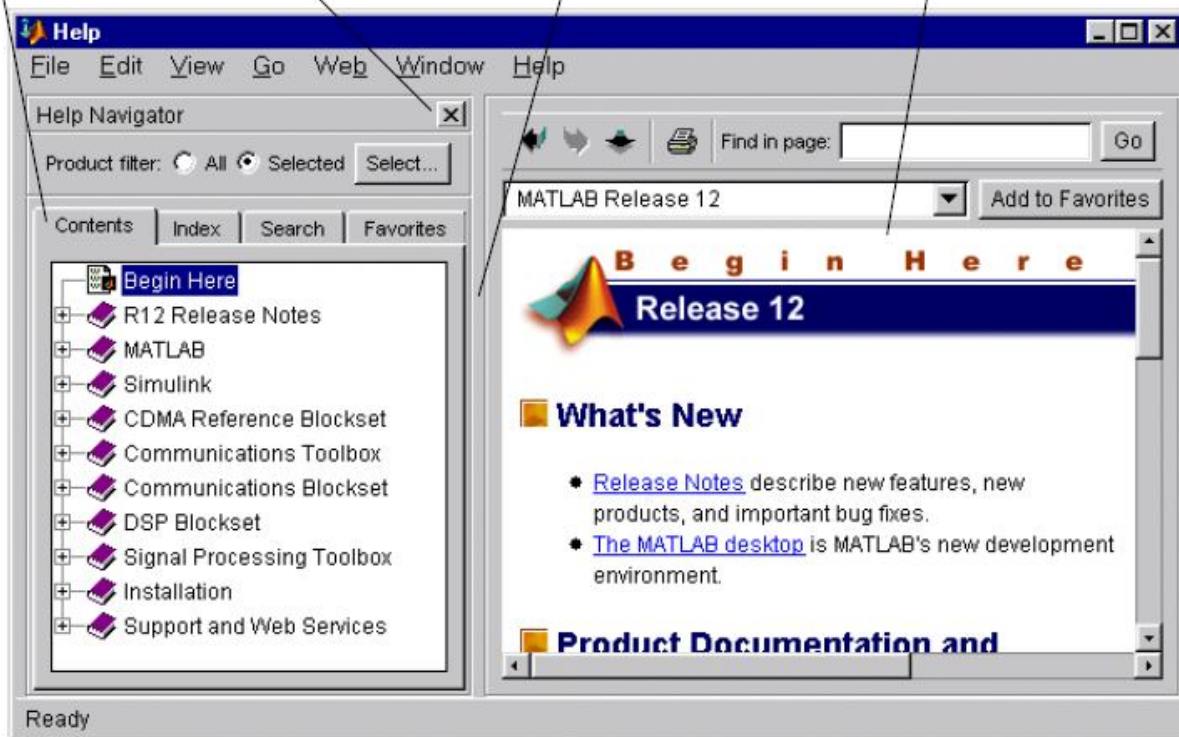


Fig.3.5 Help

4. Help Browser

The help browser is used to search and view documentation for all Math Works products. It is a Web browser integrated into the MATLAB desktop that displays HTML documents. To open the Help browser, click the help button in the toolbar, or type help browser in the Command Window (Fig.3.5).

5. Current Directory Browser

MATLAB file operations use the current directory and the search path as reference points. Any file that needs to be run must either be in the current directory or on the search path.

6. Workspace Browser

MATLAB workspace consists of a set of variables (named arrays) built up during MATLAB session and stored in memory. Variables can be added to the workspace by using functions, running M-files, and loading saved workspaces (Fig 3.6).

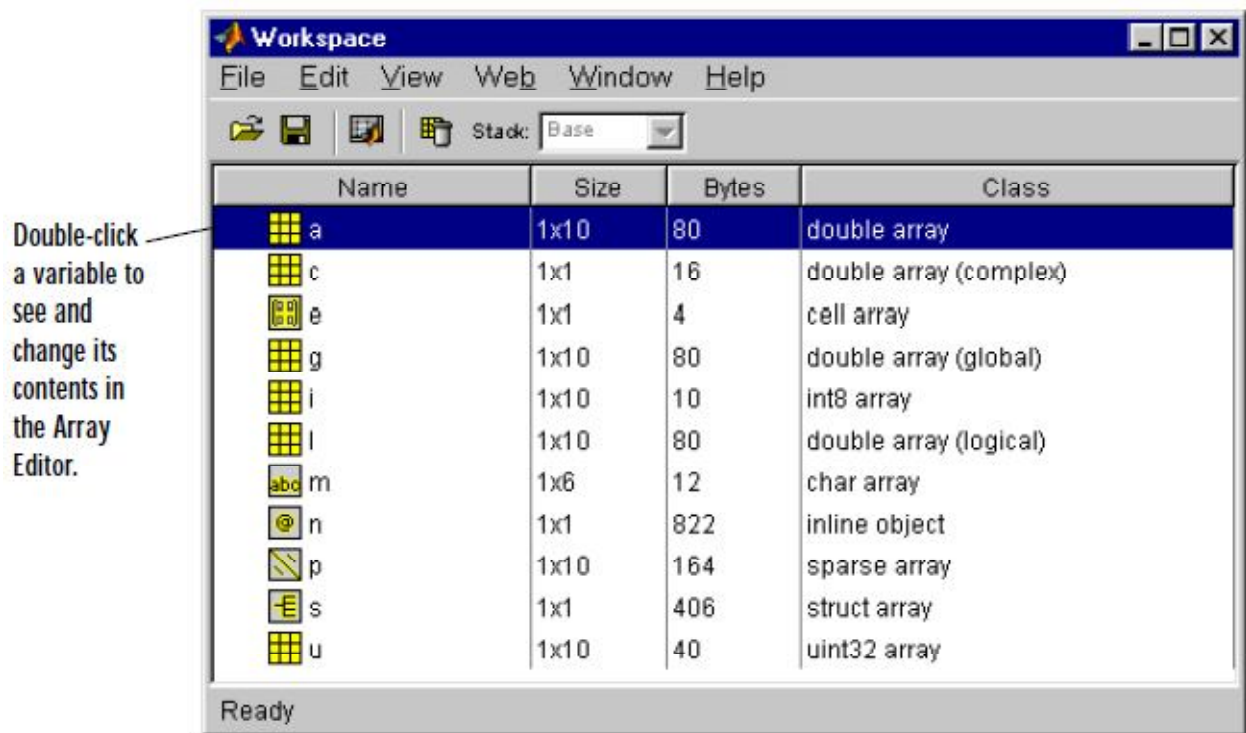


Fig.3.6 Workspace

7. Array Editor

Double-click on a variable in the Workspace browser to see it in the Array Editor. Use the Array Editor to view and edit visual representation of one- or two-dimensional numeric arrays, strings, and cell arrays of strings that are in the workspace (Fig.3.7).

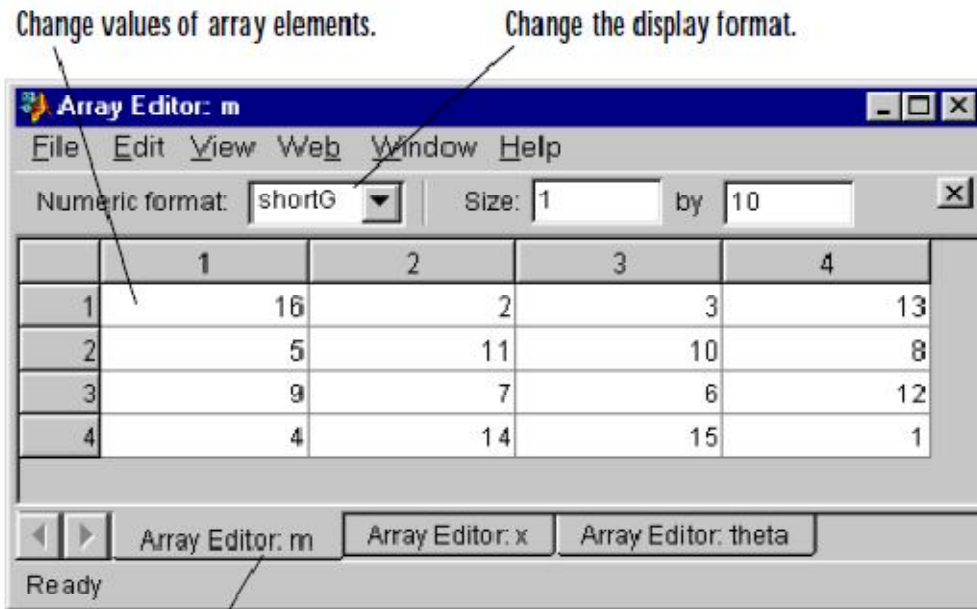


Fig.3.7 Array Editor

8. Editor/Debugger

The Editor/Debugger is used to create and debug M-files, which are programs written to run MATLAB functions. The Editor/Debugger also provides a graphical user interface for basic text editing, as well as for M-file debugging (Fig.3.8).

Comment selected lines and specify indenting style using the Text menu. Find and replace strings.

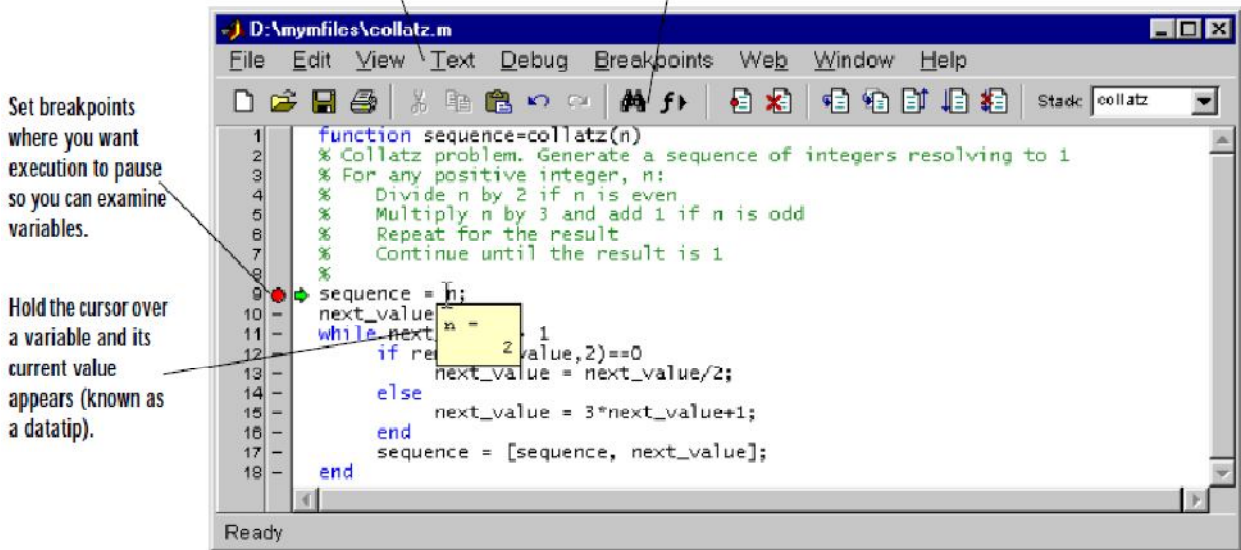


Fig 3.8 Editor/Debugger

2.5 Assignment

Assignment refers to assigning values to variable names. This results in the storage of the values in the memory location corresponding to the variable name. A variable name must comply with the following two rules:

1. It may consist only letters a–z, or letter and digits 0–9 or letter, digit and underscore (_).
2. It must start with a letter.

Examples of valid variable names: r2d2 pay_day

Examples of invalid names: pay-day 2a name\$_2a

Note that MATLAB is case sensitive, meaning that it differentiate between upper- and lowercase letters. For example: fuse, FUSE and Fuse are three different variables. Also note that Command and function names are also case sensitive.

2.5.1 Scalar

Assignment of values to scalar is as follow:

Type

```
>> a = 4
```

The press enter, you have

```
a =
```


It can be suppressed by terminating the command line with the semicolon (;) character,

e.g type

```
>> A = 6;
```

It is possible to type several commands on the same line by separating them with commas or semicolons. If they are separate with commas, they will be displayed, and if the separation is done carried out with semicolon, they will not display. For example,

```
>> a = 4,A = 6;x = 1;
```

Press enter, we have

```
a =
```

```
4
```

Since MATLAB handles complex arithmetic automatically, complex values can be assigned to variables. The unit imaginary number $\sqrt{-1}$ is preassigned to the variable *i*. Consequently, a complex value can be assigned simply as in

```
>> x = 2+i*4
```

```
x =
```

```
2.0000 + 4.0000i
```

It should be noted that MATLAB allows the symbol *j* to be used to represent the unit imaginary number for input. However, it always uses an *i* for display. For example,

```
>> x = 2+j*4
```

```
x =
```

```
2.0000 + 4.0000i
```

There are several predefined variables, for example, *pi*.

```
>> pi
```

```
ans =
```

```
3.1416
```

By default, MATLAB display output to 4 decimal places. If the user desire additional precision, enter the following:

```
>> format long
```

Now when *pi* is entered the result is displayed to 15 significant figures:

```
>> pi
```

```
ans =
```

3.14159265358979

To return to the four decimal default, type

```
>> format short
```

2.5.2 Arrays, vectors and matrices

An array is a collection of data or values that are represented by a single variable name. One dimensional array is called **vector** and two-dimensional arrays are called **matrices**. The scalars used in preceding section are actually matrices with one row and one column.

Brackets are used to enter arrays in MATLAB. For example, a row vector can be assigned as follows:

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

```
a =
```

```
1 2 3 4 5
```

A column vector can be entered into MATLAB in any of the following:

```
>> b = [2;4;6;8;10]
```

or

```
>> b = [24
```

```
6
```

```
8
```

```
10]
```

or, by transposing a row vector with the ' operator,

```
>> b = [2 4 6 8 10]'
```

The result in all three cases will be

```
b =
```

```
2
```

```
4
```

```
6
```

```
8
```

```
10
```

A matrix of values can be assigned as follows:

```
>> A = [1 2 3; 4 5 6; 7 8 9]
```

```
A =  1  2  3
     4  5  6
     7  8  9
```

In addition, the Enter key can be used to separate the rows. For example, in the following case, the Enter key would be struck after the 3, the 6 and the] to assign the matrix:

```
A =  [1 2 3
      4 5 6
      7 8 9]
```

Matrix can also be constructed by *concatenating* (i.e., joining) the vectors representing each column:

```
>> A = [[1 4 7]' [2 5 8]' [3 6 9]']
```

Colon operator

This is used for creating and manipulating arrays. If a colon is used to separate two numbers, MATLAB generates the numbers between them using an increment of one:

Example:

```
>> t = 1:5
t =
    1    2    3    4    5
```

If colons are used to separate three numbers, MATLAB generates the numbers between the first and third numbers using an increment equal to the second number:

Example:

```
>> t = 1:0.5:3
t =
    1.0000    1.5000    2.0000    2.5000    3.0000
```

Note that negative increments can also be used

```
>> t = 10:-1:5
t =
    10    9    8    7    6    5
```

In addition the colon can also be used to select the individual rows and columns of a matrix. For example, the second row of the matrix A can be selected as:

```
>> A(2,:)
```

```
ans = 4 5 6
```

We can also use the colon notation to selectively extract a series of elements from within an array. For example, based on the previous definition of the vector `t`:

```
>> t(2:4)
```

```
ans =
```

```
    9    8    7
```

Thus, the second through the fourth elements are returned.

A matrix can also be constructed from column vectors of the same length.

Example

```
x = 0:15:180;
```

```
table = [x' sin(x*pi/180)']
```

```
table =
```

```
    0    0
 15.0000  0.2588
 30.0000  0.5000
 45.0000  0.7071
 60.0000  0.8660
 75.0000  0.9659
 90.0000  1.0000
105.0000  0.9659
120.0000  0.8660
135.0000  0.7071
150.0000  0.5000
165.0000  0.2588
180.0000  0.0000
```

Linespace and logspace

The `linspace` and `logspace` functions are used to generate vectors of spaced points. The `linspace` function generates a row vector of equally spaced points. It has the form

```
linspace(x1, x2, n)
```

which generates n points between $x1$ and $x2$.

For example

```
>> linspace(0,1,6)
```

```
ans =
```

```
0 0.2000 0.4000 0.6000 0.8000 1.0000
```

If the n is omitted, the function automatically generates 100 points.

The logspace function generates a row vector that is logarithmically equally spaced. It has the form `logspace(x1, x2, n)` which generates n logarithmically equally spaced points between decades 10^{x1} and 10^{x2} .

For example,

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

```
ans =
```

```
0.1000 1.0000 10.0000 100.0000
```

If n is omitted, it automatically generates 50 points.

2.6 Mathematical operations

Mathematical operations carried out are:

Operation	Algebraic form	MATLAB
Addition	$a+b$	<code>a + b</code>
Subtraction	$a-b$	<code>a - b</code>
Multiplication	$a \times b$	<code>a * b</code>
Right division	a/b	<code>a / b</code>
Left division	$b \backslash a$	<code>a \ b</code>
Power	a^b	<code>a ^ b</code>

2.6.1 Precedence of operators

Precedence	Operator
1	Parentheses (round brackets)
2	Power, left to right
3	Multiplication and division, left to right
4	Addition and subtraction, left to right

2.6.2 Arithmetic operators of element by element on arrays.

+	Addition
-	Subtraction

<code>.*</code>	Element-by-element multiplication
<code>./</code>	Element-by-element division
<code>.\</code>	Element-by-element left division
<code>.^</code>	Element-by-element power
<code>.'</code>	Unconjugated array transpose

Examples:

```
>> 2*pi
```

```
ans =
```

```
6.2832
```

```
>> y = pi/4;
```

```
>> y ^ 2.45
```

```
ans =
```

```
0.5533
```

```
>> y = -4 ^ 2
```

```
y =
```

```
-16
```

```
>> y = (-4) ^ 2
```

```
y =
```

```
16
```

```
>> x = 2+4i
```

```
>> 3 * x
```

```
ans =
```

```
6.0000 + 12.0000i
```

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

```
>> b = [4 5 6]';
```

```
>> a * A
```

```
ans =
```

```
30 36 42
```

```
>> A * b
ans =
    32
    77
   122
```

Note that matrices cannot be multiplied if the inner dimensions are unequal. Here is what happens when the dimensions are not those required by the operations.

```
>> A * a
?? Error using ==> mtimes
Inner matrix dimensions must agree.
```

```
>> A^2
ans =
    30 36 42
    66 81 96
   102 126 150
```

```
>> A.^2
ans =
    1 4 9
   16 25 36
   49 64 81
```

2.7 Use of Built-In functions

MATLAB program has so many built-in function which can be used by typing the correct function name. Examples include: plotting function, polynomial function, etc.

Example: Calculate the velocity of a free-falling body given as

$$v = \sqrt{\frac{gm}{cd}} \tanh\left(\sqrt{\frac{gcd}{m}} t\right)$$

where v is velocity (m/s), g is the acceleration due to gravity (9.81 m/s^2), m is mass (kg), cd is the drag coefficient (kg/m), and t is time (s).

```
>> t = [0:2:20]';
>> g = 9.81; m = 68.1; cd = 0.25;
>> v = sqrt(g*m/cd)*tanh(sqrt(g*cd/m)*t)
```

v =

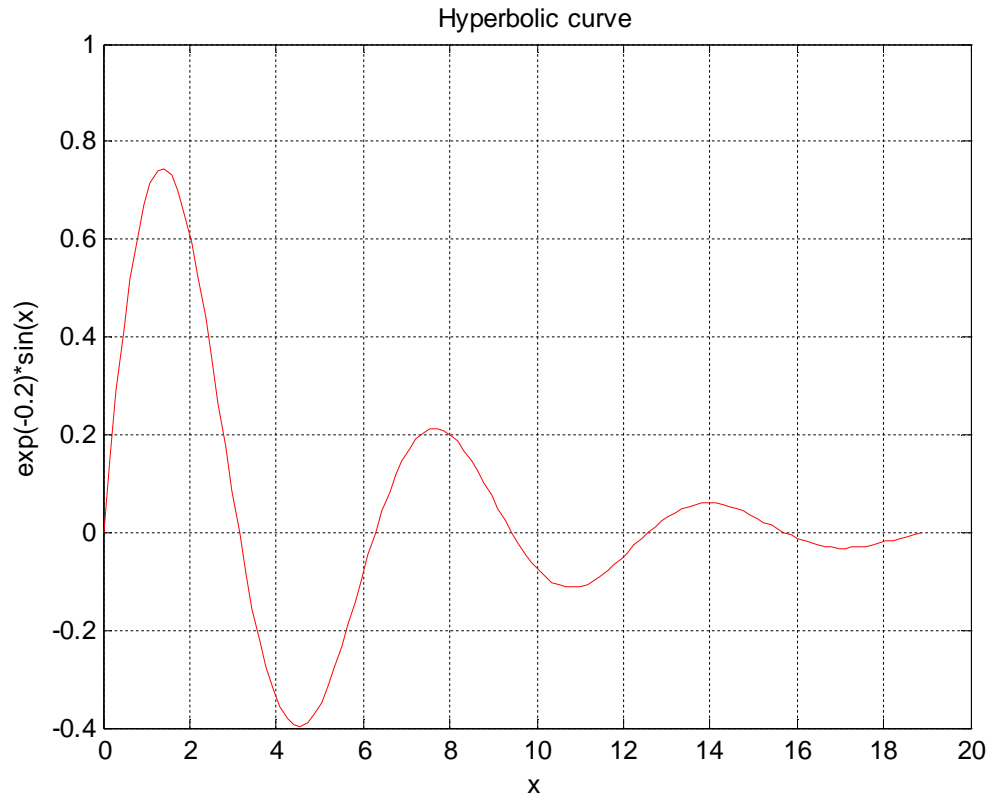
0
18.7292
33.1118
42.0762
46.9575
49.4214
50.6175
51.1871
51.4560
51.5823
51.6416

2.8 Graphics

MATLAB can be use for graphical presentation of data.

Examples:

1. `x = 0 : pi/20 : 6 * pi;`
`y = exp(-0.2*x).*sin(x);`
`plot(x, y, 'r'), title('Hyperbolic curve'),xlabel('x'),ylabel('exp(-0.2)*sin(x)'),grid`



2. `% Vertical motion under gravity`
`g = 9.8; % acceleration due to gravity`
`u = 60; % initial velocity (meters/sec)`
`t = 0 : 0.1 : 12.3; % time in seconds`
`s = u * t - g / 2 * t .^ 2; % vertical displacement in meters`
`plot(t, s), title('Vertical motion under gravity'), xlabel('time'), ... ylabel('vertical displacement'), grid`
`disp([t' s']) % display a table`

Vertical motion under gravity

