Introduction to Matlab:

Application to Electrical Engineering

Houssem Rafik El Hana Bouchekara

Umm El Qura University

(version 1, Februray 2011)

1 Chapter 1

1.1 Tutorial lessons 1

1.1.1 Introduction

The primarily objective is to help you learn *quickly* the first steps. The emphasis here is “learning by doing”. Therefore, the best way to learn is by trying it yourself. Working through the examples will give you a feel for the way that MATLAB operates. In this introduction we will describe how MATLAB handles simple numerical expressions and mathematical formulas.

The name MATLAB stands for MATrix LABoratory. MATLAB was written originally to provide easy access to matrix software developed by the LINPACK (linear system package) and EISPACK (Eigen system package) projects.

The basic building block in MATLAB is the **matrix**. The fundamental data type is the **array**. Vectors, scalars, real and complex matrices are all automatically handled as special cases of basic arrays. The built-in functions are optimized for **vector operations**. Thus, vectorized commands or codes run much faster in MATLAB (**vectorization** is a way of computing in which an operation is performed simultaneously on a list of numbers rather than sequentially on each member of the list).

A nice thing to realize is that MATLAB is primarily a numerical computation package, although with the '**Symbolic**' Toolbox it can do also symbolic algebra. Mathematica, Maple, and Macsyma are primarily symbolic algebra packages. MATLAB's ease of use is its best feature since you can have more learning with less effort, while the computer algebra systems have a steeper learning curve.

In mathematical computations, especially those that utilize vectors and matrices, MATLAB is better in terms of ease of use, availability of built-in functions, ease of programming, and speed. MATLAB's popularity today has forced such packages as Macsyma and Mathematica to provide extensions for files in MATLAB's format.

There are numerous prepared commands for **2D** and **3D graphics** as well as for animation. The user is not limited to the built-in functions; he can write his own functions in MATLAB language. Once written, these functions work just like the internal functions. MATLAB's language is designed to be **easy to learn and use**.

The many built-in functions provide excellent tools for linear algebra, signal processing, data analysis, optimization, solution of ordinary differential equations (ODEs), and many other types of scientific operations.

There are also several optional 'toolboxes' available which are collections of functions written for special applications such as '**Image Processing**', 'Statistics', 'Neural Networks', etc.

The software package has been commercially available since 1984 and is now considered as a standard tool at most universities and industries worldwide.

1.2 Starting and Quitting MATLAB

1.2.1 Starting MATLAB

On a Microsoft Windows platform, to start MATLAB, double-click the MATLAB shortcut icon on your Windows desktop.

On Linux, to start MATLAB, type matlab at the operating system prompt.

After starting MATLAB, the MATLAB desktop opens – see “MATLAB Desktop”.

You can change the directory in which MATLAB starts, define startup options including running a script upon startup, and reduce startup time in some situations.

1.2.2 Quitting MATLAB

To end your MATLAB session, select **Exit MATLAB** from the **File** menu in the desktop, or type quit in the Command Window. To execute specified functions each time MATLAB quits, such as saving the workspace, you can create and run a finish.m script.

1.3 MATLAB Desktop

When you start MATLAB, the MATLAB desktop appears, containing tools (graphical user interfaces) for managing files, variables, and applications associated with MATLAB.

The first time MATLAB starts, the desktop appears as shown in the following illustration, although your Launch Pad may contain different entries.

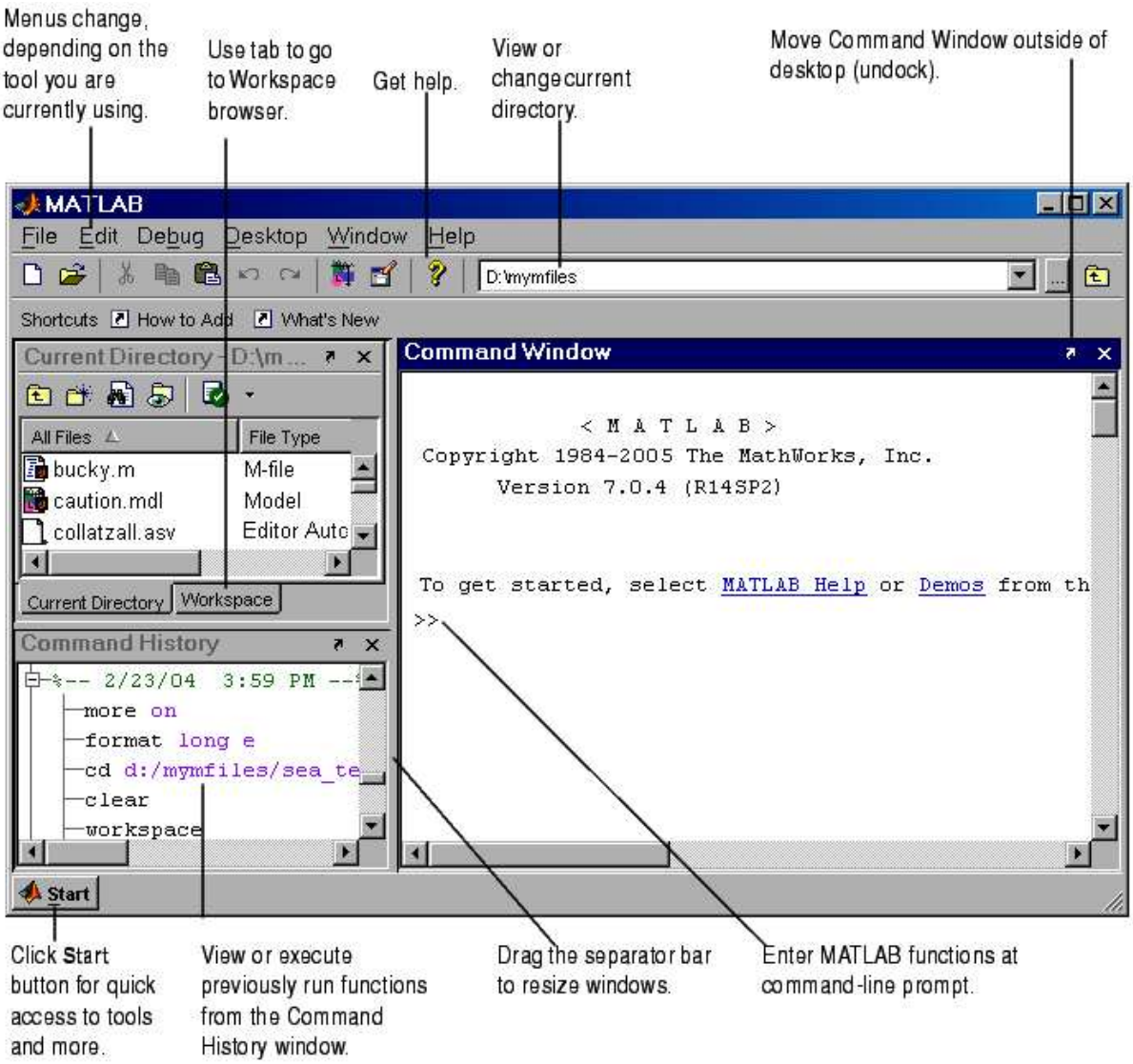

Figure 1: The graphical interface to the MATLAB workspace.

You can change the way your desktop looks by opening, closing, moving, and resizing the tools in it. You can also move tools outside of the desktop or return them back inside the desktop (docking). All the desktop tools provide common features such as context menus and keyboard shortcuts.

You can specify certain characteristics for the desktop tools by selecting **Preferences** from the **File** menu. For example, you can specify the font characteristics for Command Window text. For more information, click the **Help** button in the **Preferences** dialog box as shown in Figure 2.

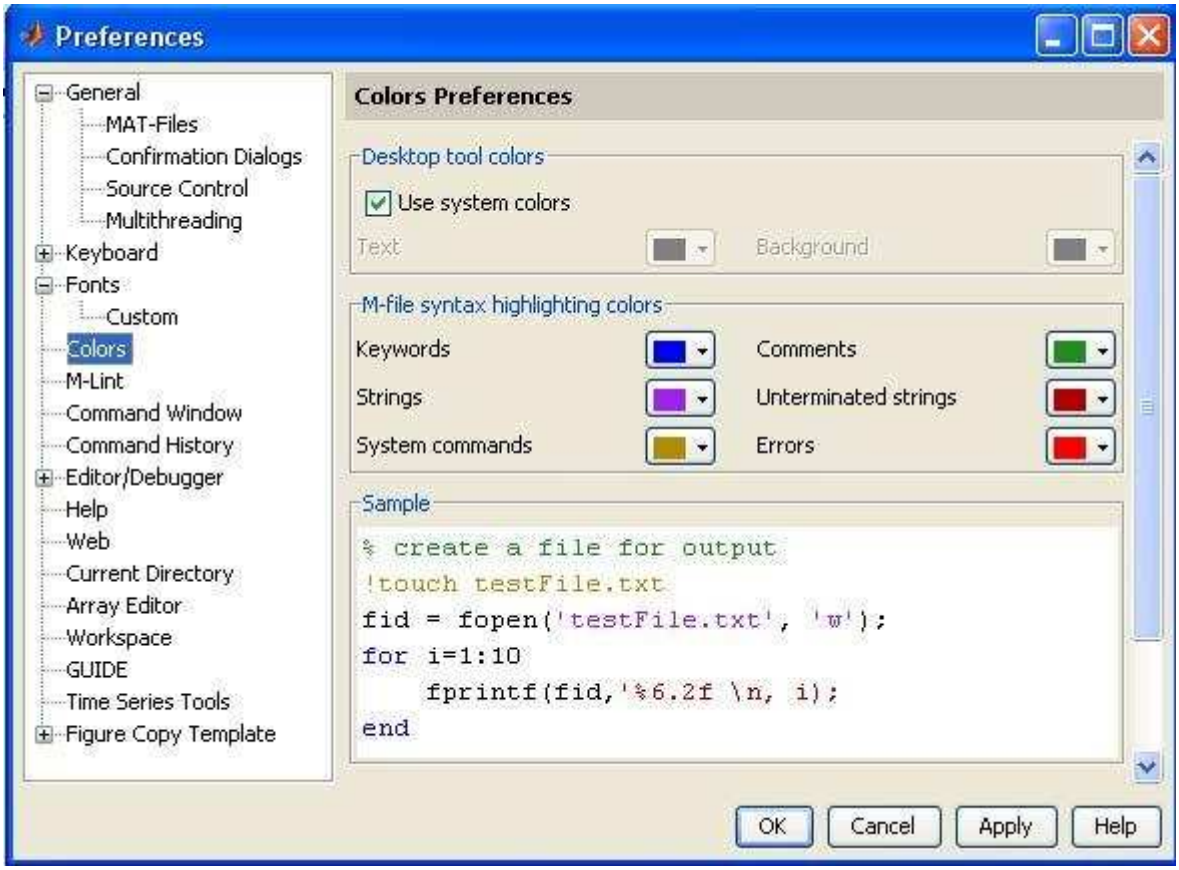

Figure 2: Customization

1.4 Desktop Tools

This section provides an introduction to MATLAB's desktop tools. You can also use MATLAB functions to perform most of the features found in the desktop tools. The tools are:

- Command Window".
- Command History.
- Launch Pad
- Help Browser.
- Current Directory Browser.
- Workspace Browser.
- Array Editor.
- Editor/Debugger.

1.4.1 Command Window

Use the Command Window to enter variables and run functions and M-files.

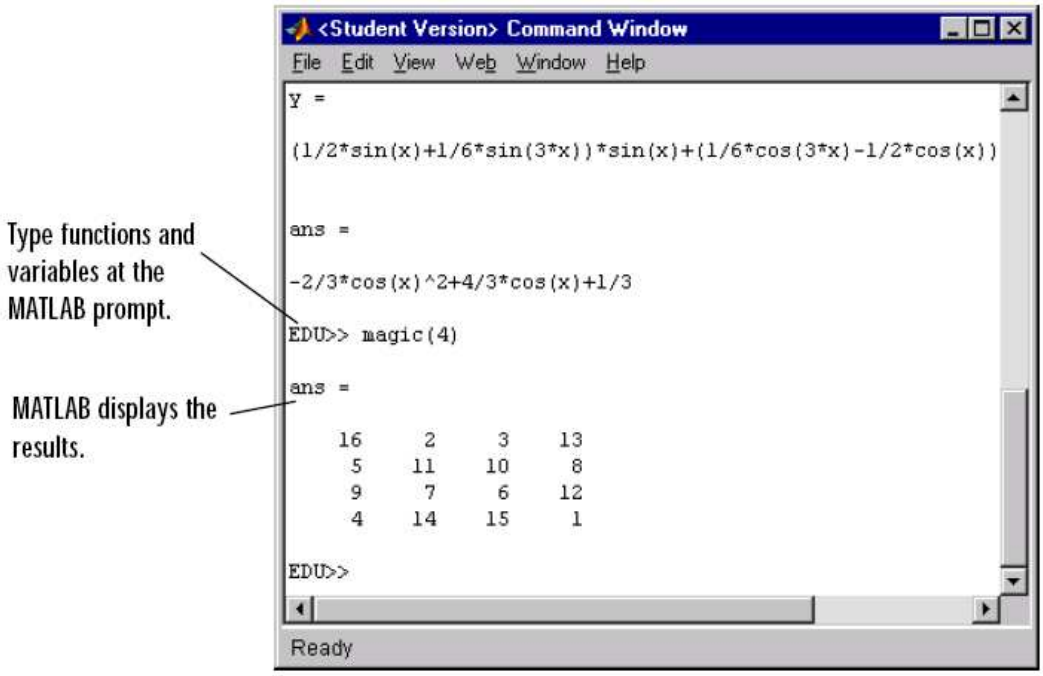

Figure 3: Command Window.

1.4.2 Command History

Lines you enter in the Command Window are logged in the Command History window. In the Command History, you can view previously used functions, and copy and execute selected lines.

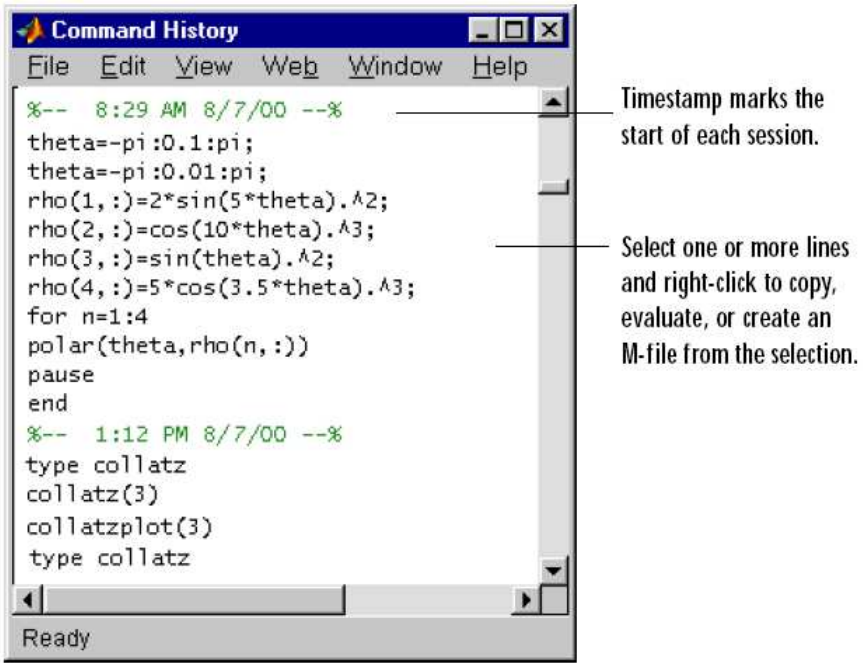

Figure 4: Command History.

To save the input and output from a MATLAB session to a file, use the diary function.

Running External Programs

You can run external programs from the MATLAB Command Window. The exclamation point character ! is a shell escape and indicates that the rest of the input line is a command to the operating system. This is useful for invoking Timestamp marks the start of each session.

Select one or more lines and right-click to copy, evaluate, or create an M-file from the selection. utilities or running other programs without quitting MATLAB. On Linux, for example,

!emacs magik.m

invokes an editor called emacs for a file named magik.m. When you quit the external program, the operating system returns control to MATLAB.

1.4.3 Launch Pad

MATLAB's Launch Pad provides easy access to tools, demos, and documentation.

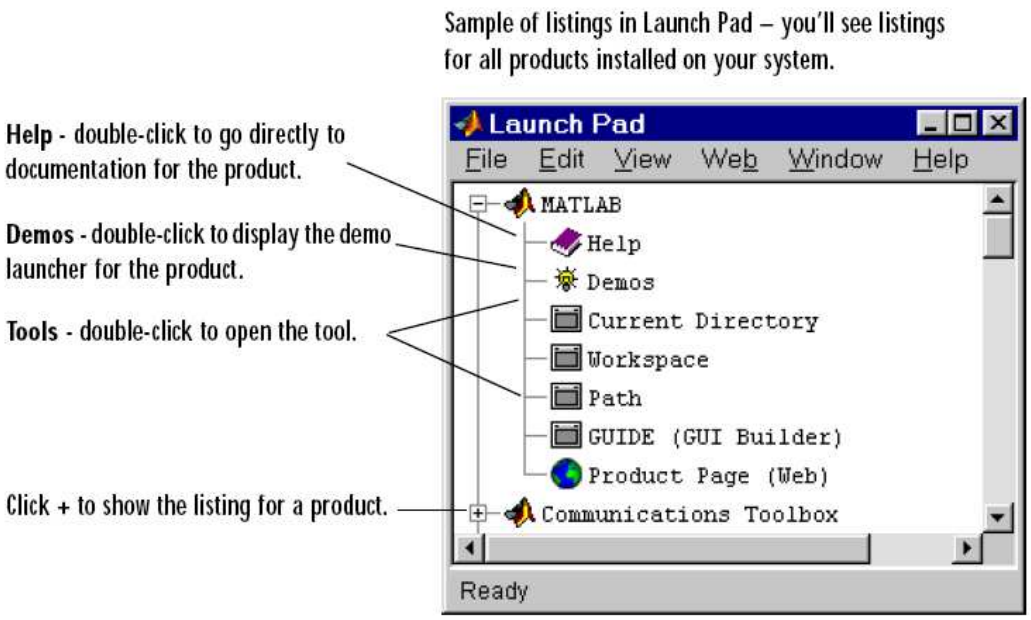

Figure 5: Launch Pad.

1.4.4 Help Browser

Use the Help browser to search and view documentation for all MathWorks products. The Help browser 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 helpbrowser in the Command Window.

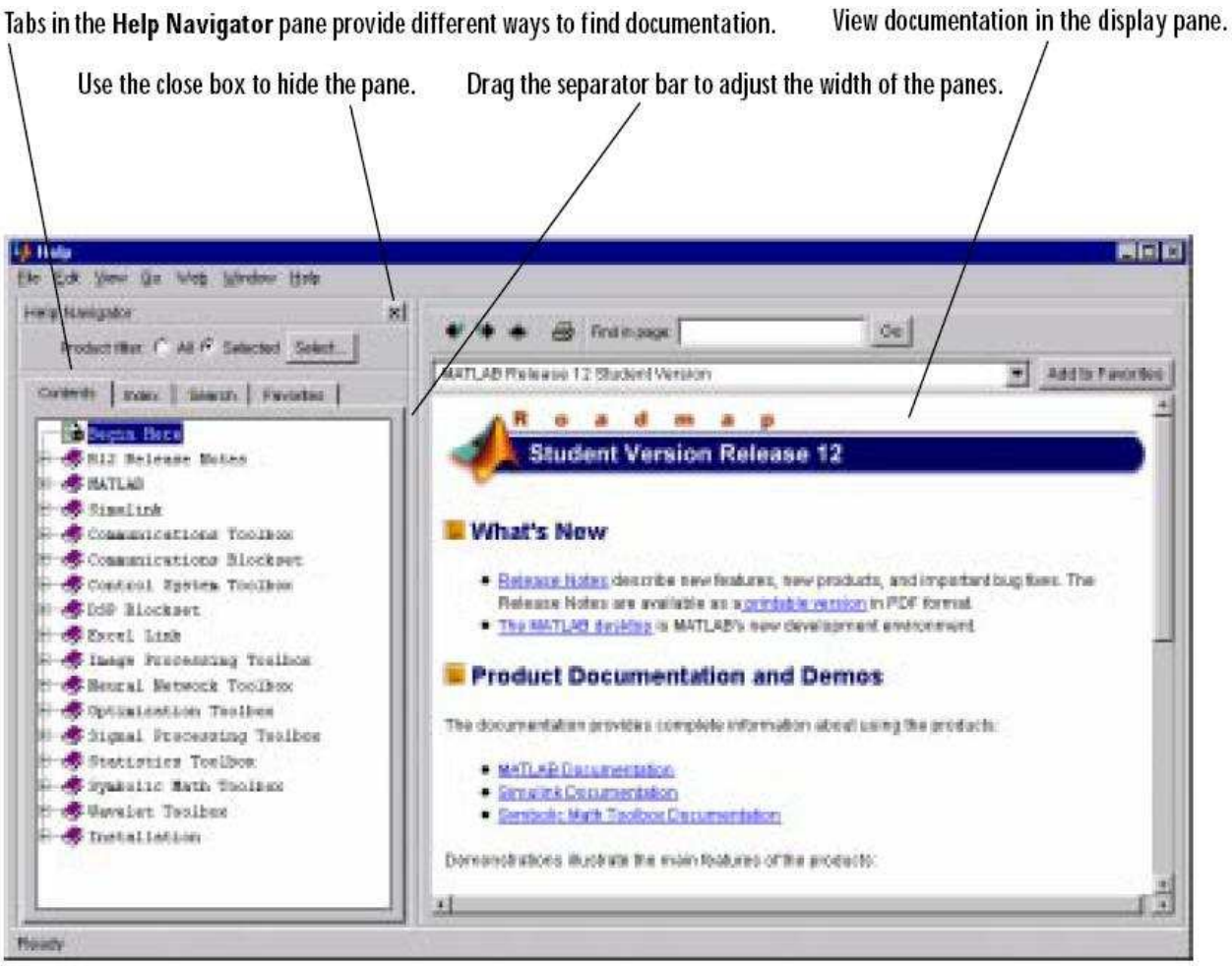

Figure 6: Help Browser

1.4.5 Current Directory Browser

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

A quick way to view or change the current directory is by using the Current Directory field in the desktop toolbar as shown below.

Figure 7: Current Directory Browser.

To search for, view, open, and make changes to MATLAB-related directories and files, use the MATLAB Current Directory browser. Alternatively, you can use the functions dir, cd, and delete.

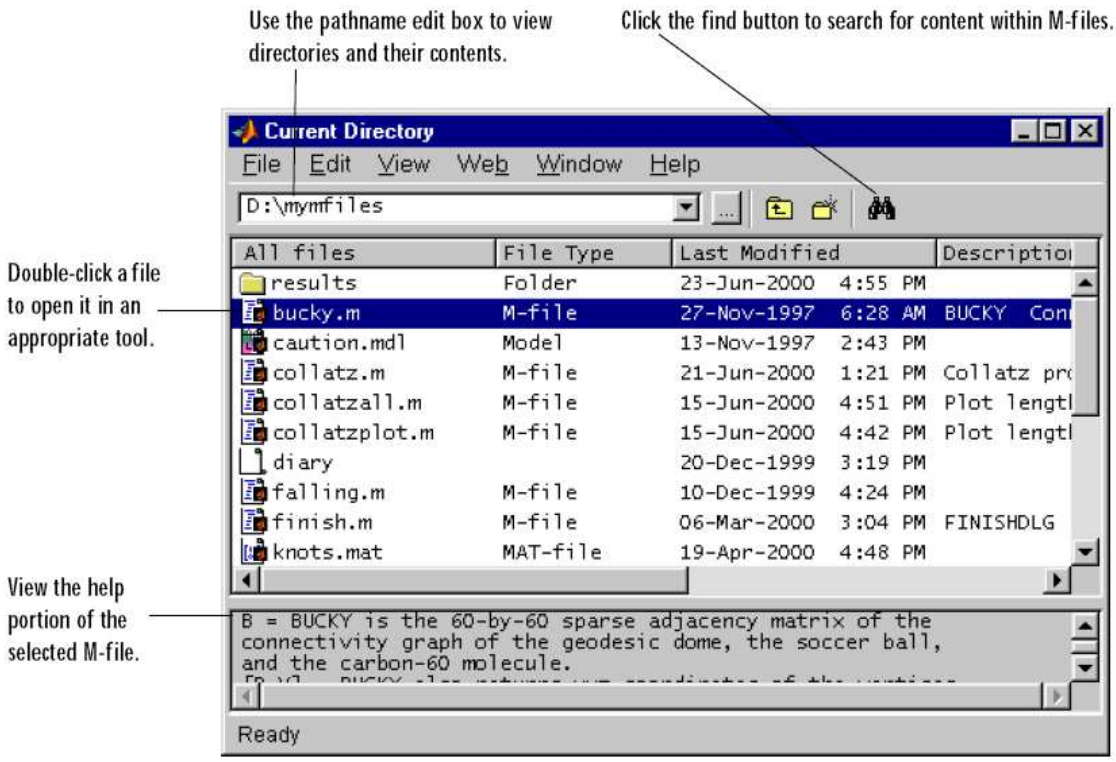

Figure 8: Current Directory Browser.

Search Path

To determine how to execute functions you call, MATLAB uses a *search path* to find M-files and other MATLAB-related files, which are organized in directories on your file system. Any file you want to run in MATLAB must reside in the current directory or in a directory that is on the search path. By default, the files supplied with MATLAB and MathWorks toolboxes are included in the search path.

To see which directories are on the search path or to change the search path, select Set Path from the File menu in the desktop, and use the Set Path dialog box. Alternatively, you can use the path function to view the search path, addpath to add directories to the path, and rmpath to remove directories from the path.

1.4.6 Workspace Browser

The MATLAB workspace consists of the set of variables (named arrays) built up during a MATLAB session and stored in memory. You add variables to the workspace by using functions, running M-files, and loading saved workspaces.

To view the workspace and information about each variable, use the Workspace browser, or use the functions who and whos.

To delete variables from the workspace, select the variable and select **Delete** from the **Edit** menu. Alternatively, use the clear function.

The workspace is not maintained after you end the MATLAB session. To save the workspace to a file that can be read during a later MATLAB session, select **Save Workspace As** from the **File** menu, or use the save function. This saves the workspace to a binary file called a MAT-file, which has a .mat extension. There are options for saving to different formats. To read in a MAT-file, select **Import Data** from the **File** menu, or use the load function.

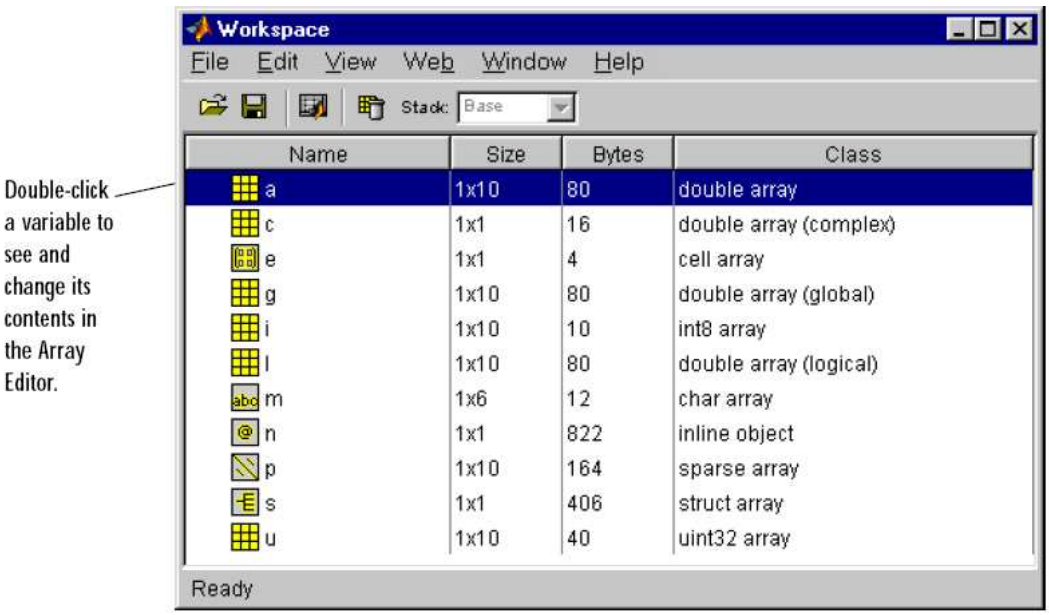

Figure 9: Workspace Browser.

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 a visual representation of one- or two-dimensional numeric arrays, strings, and cell arrays of strings that are in the workspace.

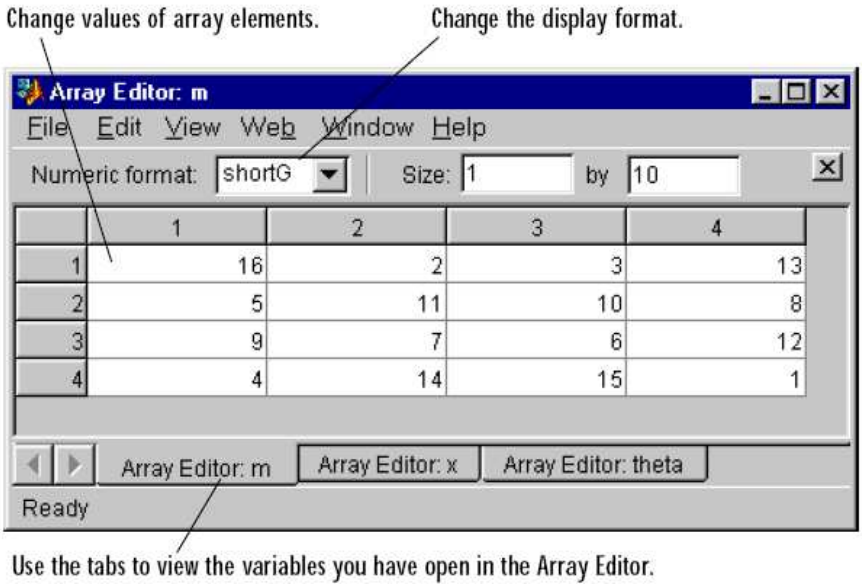

Figure 10: Array Editor.

1.4.7 Editor/Debugger

Use the Editor/Debugger to create and debug M-files, which are programs you write to run MATLAB functions. The Editor/Debugger provides a graphical user interface for basic text editing, as well as for M-file debugging.

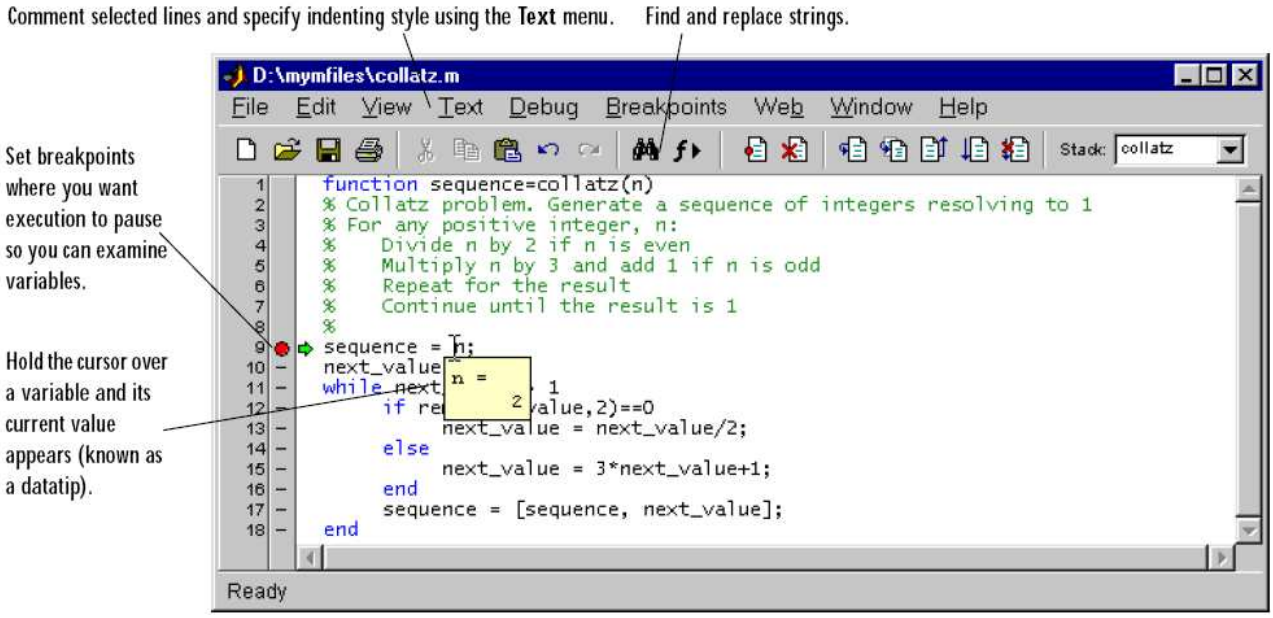

Figure 11: Editor/Debugger.

You can use any text editor to create M-files, such as Emacs, and can use preferences (accessible from the desktop **File** menu) to specify that editor as the default. If you use another editor, you can still use the MATLAB Editor/ Debugger for debugging, or you can use debugging functions, such as dbstop, which sets a breakpoint.

If you just need to view the contents of an M-file, you can display it in the Command Window by using the type function.

1.5 Getting started

Now, we are interested in doing some simple calculations. We will assume that you have sufficient understanding of your computer under which MATLAB is being run.

You are now faced with the MATLAB desktop on your computer, which contains the prompt (>>) in the Command Window. Usually, there are 2 types of prompt:

```
>>      for full version

EDU>    for educational version
```

Note: To simplify the notation, we will use this prompt, >>, as a standard prompt sign, though our MATLAB version is for educational purpose.

1.5.1 Using MATLAB as a calculator

As an example of a simple interactive calculation, just type the expression you want to evaluate. Let's start at the very beginning. For example, let's suppose you want to calculate the expression, $1 + 2 \times 3$. You type it at the prompt command (>>) as follows,

```
>> 1+2*3

ans =

7
```

You will have noticed that if you do not specify an output variable, MATLAB uses a default variable 'ans', short for answer, to store the results of the current calculation. Note that the variable 'ans' is created (or overwritten, if it is already existed). To avoid this, you may assign a value to a variable or output argument name. For example,

```
>> x = 1+2*3

x       =

7
```

will result in x being given the value $1 + 2 \times 3 = 7$. This variable name can always be used to refer to the results of the previous computations. Therefore, computing 4x will result in

```
>> 4*x

ans     =

28.0000
```

Before we conclude this minimum session, Table 1.1 gives the partial list of commonly used MATLAB operators and special characters used to solve many engineering and science problems.

Table 1: Operators and special Characteristics

+	plus
-	minus
*	matrix multiplication
.*	array multiplication
^	matrix power
.^	array power
kron	Kronecker tensor product
\	backslash (left division)
/	slash (right division)
./ and .\	right and left array division
:	colon
( )	parentheses
[ ]	brackets
{ }	curly braces
.	decimal point
...	continuation
,	comma
;	semicolon
%	comment
!	exclamation point
'	transpose and quote
.'	nonconjugated transpose
=	assignment
==	equality
< >	relational operators
&	logical AND
|	logical OR
~	logical NOT
xor	logical exclusive OR

After learning the minimum MATLAB session, we will now learn to use some additional operations.

1.5.2 Creating MATLAB variables

MATLAB variables are created with an assignment statement. The syntax of variable assignment is variable name = a value (or an expression)

For example,

```
>> x = expression
```

where expression is a combination of numerical values, mathematical operators, variables, and function calls. On other words, expression can involve:

- manual entry
- built-in functions
- user-defined functions

1.5.3 Overwriting variable

Once a variable has been created, it can be reassigned. In addition, if you do not wish to see the intermediate results, you can suppress the numerical output by putting a semicolon (;) at the end of the line. Then the sequence of commands looks like this:

```
>> t = 5;

>> t = t+1

t          =

6
```

1.5.4 Error messages

If we enter an expression incorrectly, MATLAB will return an error message. For example, in the following, we left out the multiplication sign, *, in the following expression

```
>> x = 10;

>> 5x

??? 5x

|

Error: Unexpected MATLAB expression.
```

1.5.5 Making corrections

To make corrections, we can, of course retype the expressions. But if the expression is lengthy, we make more mistakes by typing a second time. A previously typed command can be recalled with the up-arrow key ↑. When the command is displayed at the command prompt, it can be modified if needed and executed.

1.5.6 Controlling the hierarchy of operations or precedence

Let's consider the previous arithmetic operation, but now we will include *parentheses*. For example, $1 + 2 \times 3$ will become $(1 + 2) \times 3$

```
>> (1+2)*3

ans  =

9
```

and, from previous example

```
>> 1+2*3

ans      =

7
```

By adding parentheses, these two expressions give different results: 9 and 7.

The order in which MATLAB performs arithmetic operations is exactly that taught in high school algebra courses. *Exponentiations* are done *first*, followed by *multiplications* and *divisions*, and finally by *additions* and *subtractions*. However, the standard order of precedence of arithmetic operations can be changed by inserting *parentheses*. For example, the result of $1 + 2 \times 3$ is quite different than the similar expression with parentheses $(1 + 2) \times 3$. The results are 7 and 9 respectively. Parentheses can always be used to overrule *priority*, and their use is recommended in some complex expressions to avoid ambiguity.

Therefore, to make the evaluation of expressions unambiguous, MATLAB has estab- lished a series of rules. The order in which the arithmetic operations are evaluated is given in Table 2Table 2. MATLAB arithmetic operators obey the same *precedence* rules as those in

Table 2: Hierarchy of arithmetic operations.

Precedence	Mathematical operations
First	The contents of all parentheses are evaluated first, starting from the innermost parentheses and working outward.
Second	All exponentials are evaluated, working from left to right
Third	All multiplications and divisions are evaluated, working from left to right
Fourth	All additions and subtractions are evaluated, starting from left to right

most computer programs. For operators of *equal* precedence, evaluation is from *left* to *right*. Now, consider another example:

$$\frac{1}{2+3^2}+\frac{4}{5}\times\frac{6}{7}$$

In MATLAB, it becomes

```
>> 1/(2+3^2)+4/5*6/7

ans =

0.7766
```

or, if parentheses are missing,

```
>> 1/2+3^2+4/5*6/7

ans =

10.1857
```

So here what we get: two different results. Therefore, we want to emphasize the importance of precedence rule in order to avoid ambiguity.

1.5.7 Controlling the appearance of floating point number

MATLAB by default displays only 4 decimals in the result of the calculations, for example −163.6667, as shown in above examples. However, MATLAB does numerical calculations in *double* precision, which is 15 digits. The command format controls how the results of computations are displayed. Here are some examples of the different formats together with the resulting outputs.

```
>> format short

>> x=-163.6667
```

If we want to see all 15 digits, we use the command format long

```
>> format long

>> x= -1.636666666666667e+002
```

To return to the standard format, enter format short, or simply format.

There are several other formats. For more details, see the MATLAB documentation, or type help format.

Note - Up to now, we have let MATLAB repeat everything that we enter at the prompt (>>). Sometimes this is not quite useful, in particular when the output is pages en length. To prevent MATLAB from echoing what we type, simply enter a semicolon (;) at the end of the command. For example,

```
>> x=-163.6667;
```

and then ask about the value of x by typing,

```
>> x

x          =

-163.6667
```

1.5.8 Managing the workspace

The contents of the workspace persist between the executions of separate commands. Therefore, it is possible for the results of one problem to have an effect on the next one. To avoid this possibility, it is a good idea to issue a clear command at the start of each new independent calculation.

```
>> clear
```

The command clear or clear all removes all variables from the workspace. This frees up system memory. In order to display a list of the variables currently in the memory, type

```
>> who
```

while, whos will give more details which include size, space allocation, and class of the variables.

1.5.9 Keeping track of your work session

It is possible to keep track of everything done during a MATLAB session with the diary command.

```
>> diary
```

or give a name to a created file,

```
>> diary FileName
```

where FileName could be any arbitrary name you choose.

The function diary is useful if you want to save a complete MATLAB session. They save all input and output as they appear in the MATLAB window. When you want to stop the recording, enter diary off. If you want to start recording again, enter diary on. The file that is created is a simple text file. It can be opened by an editor or a word processing program and edited to remove extraneous material, or to add your comments. You can use the function type to view the diary file or you can edit in a text editor or print. This command is useful, for example in the process of preparing a homework or lab submission.

1.5.10 Entering multiple statements per line

It is possible to enter multiple statements per line. Use commas (,) or semicolons (;) to enter more than one statement at once. Commas (,) allow multiple statements per line without suppressing output.

```
>> a=7; b=cos(a), c=cosh(a)

b        =

0.6570

c        =

548.3170
```

1.5.11 Miscellaneous commands

Here are few additional useful commands:

- To clear the Command Window, type clc
- To abort a MATLAB computation, type ctrl-c
- To continue a line, type . . .

1.5.12 Getting help

To view the online documentation, select MATLAB Help from Help menu or MATLAB Help directly in the Command Window. The preferred method is to use the *Help Browser*. The Help Browser can be started by selecting the ? icon from the desktop toolbar. On the other hand, information about any command is available by typing

```
>> help Command
```

Another way to get help is to use the lookfor command. The lookfor command differs from the help command. The help command searches for an exact function name match, while the lookfor command searches the quick summary information in each function for a match. For example, suppose that we were looking for a function to take *the inverse of a matrix*. Since MATLAB does not have a function named inverse, the command help inverse will produce nothing. On the other hand, the command lookfor inverse will produce detailed information, which includes the function of interest, inv.

```
>> lookfor inverse
```

Note - At this particular time of our study, it is important to emphasize one main point. Because MATLAB is a huge program; it is impossible to cover *all the details* of each function one by one. However, we will give you information how to get help. Here are some examples:

- Use on-line help to request info on a specific function

```
>> help sqrt
```

- In the current version (MATLAB version 7), the doc function opens the on-line version of the help manual. This is very helpful for more complex commands.

```
>> doc plot
```

- Use lookfor to find functions by keywords. The general form is

```
>> lookfor FunctionName
```

1.6 Exercises

2 Chapter 2

2.1 Mathematical functions

MATLAB offers many predefined mathematical functions for technical computing which contains a large set of mathematical functions.

Typing help elfun and help specfun calls up full lists of *elementary* and *special*functions respectively.

There is a long list of mathematical functions that are *built* into MATLAB. These functions are called *built-ins*. Many standard mathematical functions, such as sin(*x*), cos(*x*), tan(*x*), *ex*, *ln*(*x*), are evaluated by the functions sin, cos, tan, exp, and log respectively in MATLAB.

Table 3 lists some commonly used functions, where variables x and y can be numbers, vectors, or matrices.

Table 3: Elementary functions.

`abs`	Absolute value and complex magnitude
`acos, acosh`	Inverse cosine and inverse hyperbolic cosine
`acot, acoth`	Inverse cotangent and inverse hyperbolic cotangent
`acsc, acsch`	Inverse cosecant and inverse hyperbolic cosecant
`angle`	Phase angle
`asec, asech`	Inverse secant and inverse hyperbolic secant
`asin, asinh`	Inverse sine and inverse hyperbolic sine
`atan, atanh`	Inverse tangent and inverse hyperbolic tangent
`atan2`	Four-quadrant inverse tangent
`ceil`	Round toward infinity
`complex`	Construct complex data from real and imaginary components
`conj`	Complex conjugate
`cos, cosh`	Cosine and hyperbolic cosine
`cot, coth`	Cotangent and hyperbolic cotangent
`csc, csch`	Cosecant and hyperbolic cosecant
`exp`	Exponential function
`fix`	Round toward zero
`floor`	Round toward minus infinity
`gcd`	Greatest common divisor
`imag`	Imaginary part of a complex number
`lcm`	Least common multiple
`log`	Natural logarithm
`log2`	Base 2 logarithm and dissect floating-point numbers into exponent and mantissa
`log10`	Common (base 10) logarithm
`mod`	Modulus (signed remainder after division)
`nchoosek`	Binomial coefficient or all combinations
`real`	Real part of complex number
`rem`	Remainder after division
`round`	Round to nearest integer
`sec, sech`	Secant and hyperbolic secant
`sign`	Signum function
`sin, sinh`	Sine and hyperbolic sine
`sqrt`	Square root
`tan, tanh`	Tangent and hyperbolic tangent

In addition to the elementary functions, MATLAB includes a number of predefined constant values. A list of the most common values is given in Table 4.

Table 4: Predefined constant values

`pi`	3.14159265...
`i`	Imaginary unit, $\sqrt{-1}$
`j`	Same as `i`
`eps`	Floating-point relative precision, 2^{-52}
`realmin`	Smallest floating-point number, 2^{-1022}
`realmax`	Largest floating-point number, $(2-\varepsilon)2^{1023}$
`Inf`	Infinity
`NaN`	Not-a-number

2.1.1 Examples

We illustrate here some typical examples which related to the elementary functions previously defined.

As a first example, the value of the expression $y = e^{-a}\sin(x) + 10\sqrt{y}$, for $a = 5$, $x = 2$, and $y = 8$ is computed by

```
>> a = 5; x = 2; y = 8;
>> y = exp(-a)*sin(x)+10*sqrt(y)
y        =
28.2904
```

The subsequent examples are

```
>> log(142)
ans      =
4.9558
>> log10(142)
ans      =
2.1523
```

Note the difference between the natural logarithm log(x) and the decimal logarithm (base 10) log10(x).

To calculate sin(π/4) and e10 , we enter the following commands in MATLAB,

```
>> sin(pi/4)

ans      =

0.7071

>> exp(10)

ans      =

2.2026e+004
```

Notes:

- Only use built-in functions on the right hand side of an expression. Reassigning the value to a built-in function can create problems.
- There are some exceptions. For example, i and j are pre-assigned to $\sqrt{-1}$. However one or both of i or j are often used as loop indices.
- To avoid any possible confusion, it is suggested to use instead ii or jj as loop indices.

2.2 Basic plotting

2.2.1 Overview

MATLAB has an excellent set of graphic tools. Plotting a given data set or the results of computation is possible with very few commands. You are highly encouraged to plot mathematical functions and results of analysis as often as possible. Trying to understand mathematical equations with graphics is an enjoyable and very efficient way of learning math- ematics. Being able to plot mathematical functions and data freely is the most important step, and this section is written to assist you to do just that.

2.2.2 Creating simple plots

The basic MATLAB graphing procedure, for example in 2D, is to take a vector of x- coordinates, x = ($x1, \ldots, xN$), and a vector of y-coordinates, y = ($y1, \ldots, yN$), locate the points (xi, yi), with i = 1, 2, . . . , n and then join them by straight lines. You need to prepare x and y in an identical array form; namely, x and y are both row arrays or column arrays of the *same* length.

The MATLAB command to plot a graph is plot(x,y). The vectors x = (1, 2, 3, 4, 5, 6) and y = (3, −1, 2, 4, 5, 1) produce the picture shown in Figure 2.1.

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

>> y = [3 -1 2 4 5 1];

>> plot(x,y)
```

Note: The plot functions has different forms depending on the input arguments. If y is a vector plot(y)produces a piecewise linear graph of the elements of y versus the index of the elements of y. If we specify two vectors, as mentioned above, plot(x,y) produces a graph of y versus x.

For example, to plot the function sin (*x*) on the interval [0, 2π], we first create a vector of *x* values ranging from 0 to 2π, then compute the *sine* of these values, and finally plot the result:

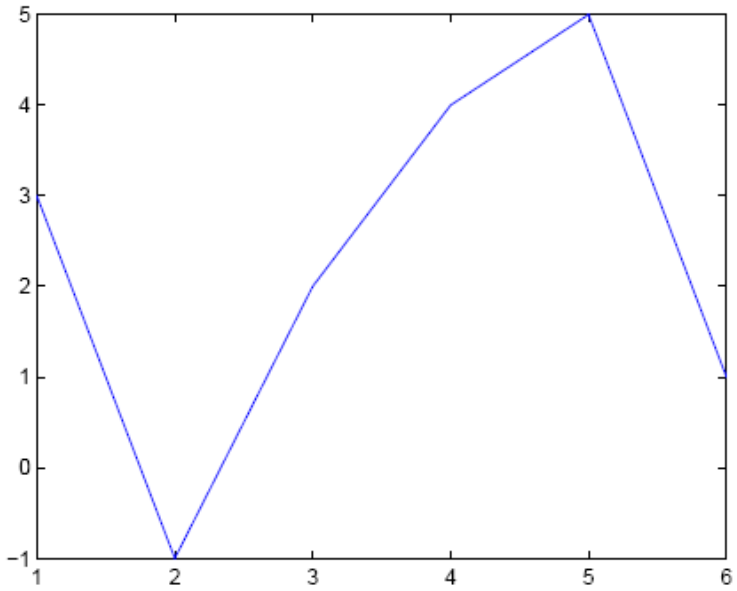

Figure 12: Plot for the vectors x and y

```
>> x = 0:pi/100:2*pi;

>> y = sin(x);

>> plot(x,y)
```

Notes:

- 0:pi/100:2*pi yields a vector that
  - starts at 0,
  - takes steps (or increments) of π/100,
  - stops when 2π is reached.
- If you omit the increment, MATLAB automatically increments by 1.

2.2.3 Adding titles, axis labels, and annotations

MATLAB enables you to add axis labels and titles. For example, using the graph from the previous example, add an *x*- and *y*-axis labels.

Now label the axes and add a title. The character \pi creates the symbol π. An example of 2D plot is shown in Figure 2.2.

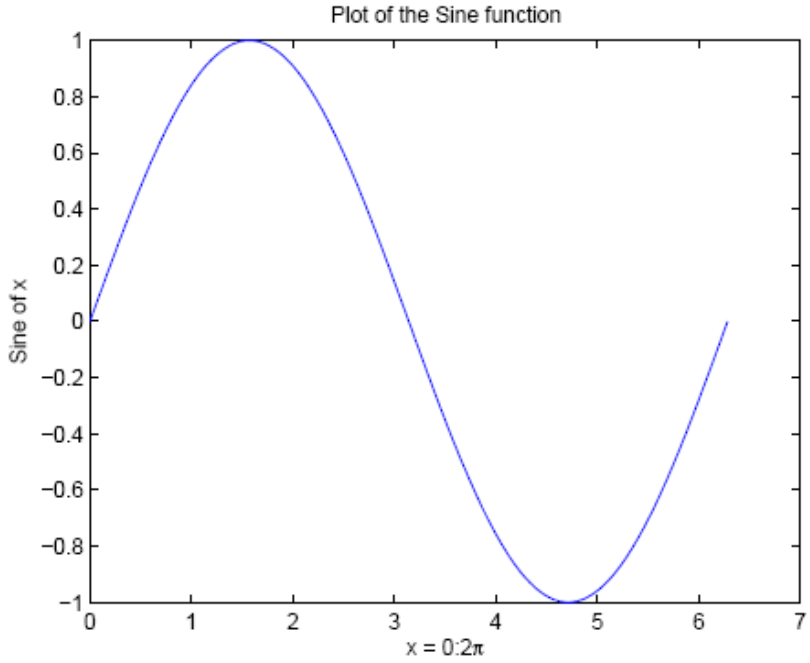

Figure 2.2: Plot of the Sine function

```
>> xlabel('x = 0:2\pi')

>> ylabel('Sine of x')

>> title('Plot of the Sine function')
```

The color of a single curve is, by default, blue, but other colors are possible. The desired color is indicated by a third argument. For example, red is selected by plot(x,y,'r'). Note the single quotes, ' ', around r.

2.2.4 Multiple data sets in one plot

Multiple (*x, y*) *pairs* arguments create *multiple* graphs with a single call to *plot*. For example, these statements plot three related functions of *x*: *y*1 = 2cos(*x*), *y*2 = cos(*x*), and *y*3 = 0.5*cos(*x*), in the interval $0 \leq x \leq 2\pi$.

```
>>  x = 0:pi/100:2*pi;

>> y1 = 2*cos(x);

>> y2 = cos(x);

>> y3 = 0.5*cos(x);

>> plot(x,y1,'--',x,y2,'-',x,y3,':')

>> xlabel('0 \leq x \leq 2\pi')

>> ylabel('Cosine functions')
```

```
>> legend('2*cos(x)','cos(x)','0.5*cos(x)')

>> title('Typical example of multiple plots')

>> axis([0 2*pi -3 3])
```

The result of multiple data sets in one graph plot is shown in Figure 13.

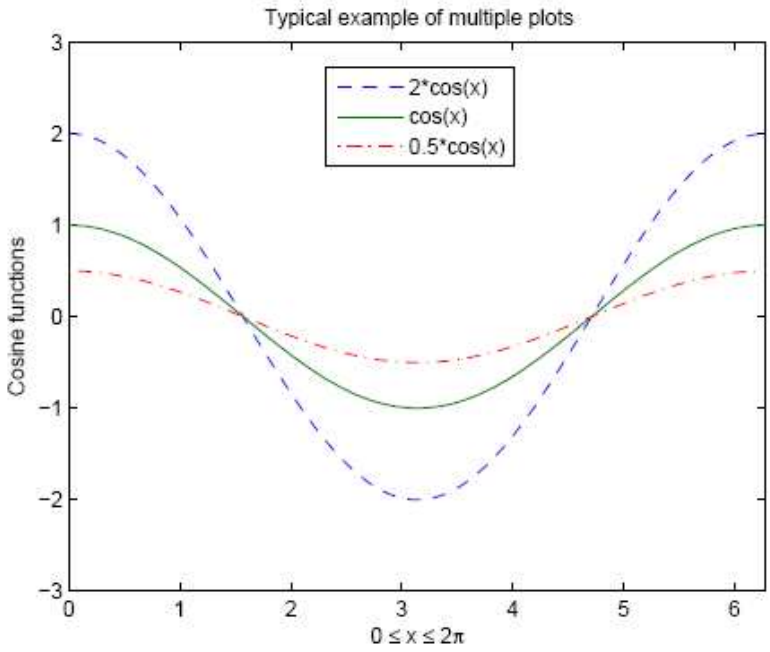

Figure 13: Typical example of multiple plots.

By default, MATLAB uses *line style* and *color* to distinguish the data sets plotted in the graph. However, you can change the appearance of these graphic components or add annotations to the graph to help explain your data for presentation.

2.2.5 Specifying line styles and colors

It is possible to specify *line styles*, *colors*, and *markers* (e.g., circles, plus signs, . . .) using the plot command: plot(x,y,'style_color_marker'), where style_color_marker is a *triplet* of values from Table 5.

To find additional information, type help plot or doc plot.

Table 5: Attributes for plot

Symbol Color	Symbol Line Style	Symbol Marker

k	Black	–	Solid	+	Plus sign
r	Red	––	Dashed	o	Circle
b	Blue	:	Dotted	*	Asterisk
g	Green	–.	Dash-dot	.	Point
c	Cyan	none	No line	×	Cross
m	Magenta			s	Square
y	Yellow			d	Diamond

Specifying the Color and Size of Markers You can also specify other line characteristics using graphics properties (see line for a description of these properties):

LineWidth —Specifies the width (in points) of the line.

MarkerEdgeColor —Specifies the color of the marker or the edge color for filled markers (circle,square, diamond, pentagram, hexagram, and the four triangles).

MarkerFaceColor —Specifies the color of the face of filled markers.

MarkerSize —Specifies the size of the marker in units of points.

For example, these statements, produce the graph of

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

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

plot(x,y,'--rs','LineWidth',2,...

        'MarkerEdgeColor','k',...

        'MarkerFaceColor','g',...

        'MarkerSize',10)
```