Lab 0: Introduction to MATLAB

The objective of this lab is to introduce you to the basic operations of MATLAB. Read through the handout sitting in front of a computer that has a MATLAB software. Practice each new command by completing the examples and exercises. Turn-in the answers for all the exercise problems as your lab report. When answering the problems, indicate the output displayed. It may be easier to open a word document, and after each command you may copy and paste the commands and outputs to the word document. Similarly, plots can also be pasted to your word document.

The time allotted for this lab: 2 hours
Requirements: a lab report to demonstrate you have finished all the exercises.

1. Introduction

In this lab, we will learn how to

- perform basic mathematical operations on simple variables, vectors, matrices, and complex numbers.
- generate 2-D plots.

2. Starting MATLAB

Start MATLAB by clicking on it in the **Start** menu.

Once the program is running, you will see a screen similar to Figure 1

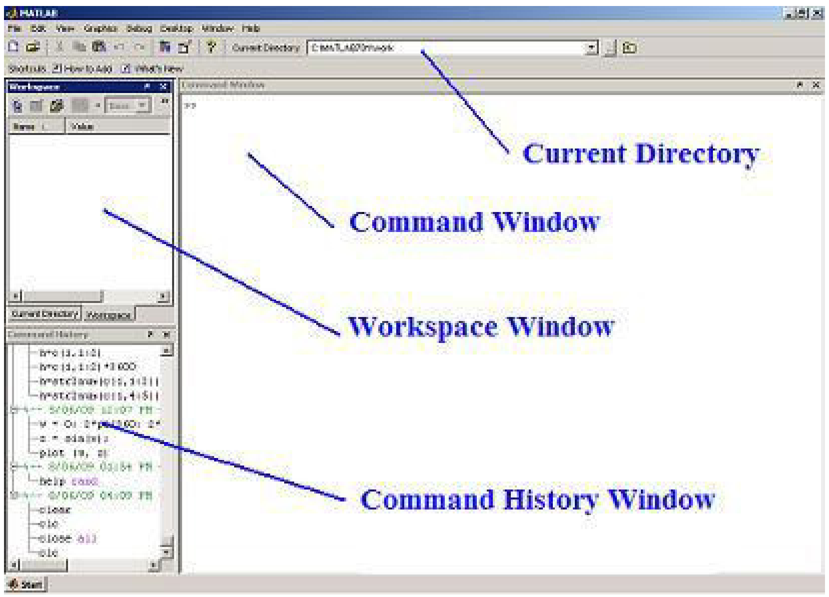


Figure 1: The MATLAB Graphical User Interface (GUI) (Image taken from http://www.matrixlab-examples.com/using-MATLAB.html)

The Command Window: is where you type commands. Hit Enter to run the command you just typed.

The Current Directory: shows the directory that you are working in. This directory is where MATLAB expects to find your files (M-files, audio, images, etc). If you encounter a ‘file not found’ error, it means the file is not in your Current Directory. You can change the working directory by typing into it or clicking through the file browser.

The Workspace Window: displays information about all the variables that are currently active. In particular, it will tell you the type (int, double, etc) of variable, as well as the dimensions of the data structure (such as a 2x2 or 8000x1 matrix). This information can be extremely useful for debugging!

The Command History Window: keeps track of the operations that you’ve performed recently. This is handy for keeping track of what you have or haven’t already tried.

3. Elementary Math in Matlab

MATLAB has a number of built-in functions, such as square root, sine, cosine, exponential, etc, functions. The following tables give a list of some commonly used functions and basic arithmetic operators. Later you will also write your own functions.

Table 1.1: Basic arithmetic operators

SYMBOL	OPERATION	EXAMPLE
$+$	Addition	$2+3$
$-$	Subtraction	$2-3$
$*$	Multiplication	$2*3$
$/$	Division	$2/3$

Table 1.2: Elementary functions

`cos(x)`	Cosine	`abs(x)`	Absolute value
`sin(x)`	Sine	`sign(x)`	Signum function
`tan(x)`	Tangent	`max(x)`	Maximum value
`acos(x)`	Arc cosine	`min(x)`	Minimum value
`asin(x)`	Arc sine	`ceil(x)`	Round towards $+\infty$
`atan(x)`	Arc tangent	`floor(x)`	Round towards $-\infty$
`exp(x)`	Exponential	`round(x)`	Round to nearest integer
`sqrt(x)`	Square root	`rem(x)`	Remainder after division
`log(x)`	Natural logarithm	`angle(x)`	Phase angle
`log10(x)`	Common logarithm	`conj(x)`	Complex conjugate

Table 1.3: Predefined constant values

`pi`	The π number, $\pi = 3.14159\ldots$
`i,j`	The imaginary unit i, $\sqrt{-1}$
`Inf`	The infinity, ∞
`NaN`	Not a number

Table 1.4 Elementary matrices

`zeros(m,n)`	Returns an m-by-n matrix of zeros
`ones(m,n)`	Returns an m-by-n matrix of ones

Try the following MATLAB function to see what it does:

```
>> zeros(2,3)
```

4 MATLAB Commands

4.1 Help

MATLAB has two important help commands to find the right syntax and a description of a command with its all options.

Typing ***help*** by itself on the command line gives you a list of help topics.

If you want to find more about the syntax of a certain command, you type in

```
>> help function_name
```

where the word function_name is in the above entry is substituted by the actual name of a function for which description is sought. For example, we can type in

```
>> help plot
```

and MATLAB will display the description, syntax, and options of the plot function.

The other helpful command is ***lookfor***. If we are not sure the exact name of the function, we can make a search for a string which is related to the function, and MATLAB displays all m-files (commands) that have a matching string. MATLAB searches for the string on the first comment line of the m-file. For example:

```
>>lookfor  inverse
```

will display all m-files which contain the string ‘inverse’ in the comment line.

Two other useful commands are

```
>>whos
```

which lists all your active variables (this info also appears in your Workspace Window), and

```
>> clear
```
which clears and deletes all variables (for when you want to start over).

MATLAB has **tab-completion** for when you're typing long function names repeatedly. This means that you can type part of a command and then press <Tab> and it will fill in the rest of the command automatically. Moreover, MATLAB **stores command history**, and previous commands can be searched by pressing the up or down arrow keys.

Try to use the up arrow key to recall a previous command you've used (e.g. `lookfor inverse` or `clear`).

4.2 Matrix Operations

MATLAB is designed to operate efficiently on matrices (hence the name MATLAB = "Matrix Laboratory). Consequently, MATLAB treats all variables as matrices, even scalars!

Like many other programming languages, variables are defined in MATLAB by typing:

<VariableName> = <Assignment>

MATLAB will then acknowledge the assignment by repeating it back to you. The following are what you would see if you defined a **scalar x**, a **vector y**, and a **matrix z**:

```
>> x=21
x =
         21
>> y=[1, 2, 3]
y =
         1         2      3
>> z=[1, 2, 3; 6, 7, 8]
 z =
       1    2       3
       6    7       8
```

You can see from the above examples that scalar variables require no special syntax; you just type the value after the equals sign. Matrices are denoted by square brackets [ ]. Commas separate the elements within a row, and semicolons separate different rows. A row array, such as y, is just a special case of a matrix that has only one row.

<u>Element-Wise Operations</u>:

You often may want to perform an operation on each element of a vector while doing a computation. For example, you may want to add two vectors by adding all of the corresponding elements. The addition (+) and subtraction (-) operators are defined to work on matrices as well as scalars. For example, if x = [1 2 3] and y = [4 6 2], then

```
>>  x=[ 1 2 3];y=[4 6 2];
>>  w  =  x+y
w =
   5     8     5
```

Multiplying two matrices element by element is a little different. The * symbol is defined as matrix multiplication when used on two matrices. Use .* to specify element-wise multiplication. So, using the x and y from above,

```
>>  w  =  x .* y
w =
   4    12     6
```

You can perform exponentiation on a vector similarly. Typing x .^ 2 squares each element of x.

```
>>  w  =  x .^ 2
w =
   1     4     9
```

Exercise 1:

Define the following column arrays in MATLAB: $a = \begin{pmatrix} 2 \\ 4 \end{pmatrix}$ and b = $\begin{pmatrix} 3 \\ 1 \end{pmatrix}$

Then issue the following commands:

```
a’
a * b’
a’ * b
a .* b
3 .* b
```

What do each of these three operators do?

’ * .*

Exercise 2:

Perform the following operation:

a * b

What is the error message?

What does the message mean?

What is the correct fix?

Exercise 3:

Perform the following:
c = a + b
d = a + b;
What does the ; do?

4.3 size command

The size command is extremely useful. This command tells you the dimensions of the matrix that MATLAB is using to represent the variable. To determine the dimension of a matrix x, for example, you type in

```
>> size(x)
```

Exercise 4:

Define e = 17 in MATLAB. Use size to find the dimensions of a, b', and e?
There is an alternate syntax for size to determine the length of a vector.

Use the alternate syntax (length) for size to determine the height of b.

You can also just use whitespace to separate elements within a row. The following two ways to define the variable are equivalent:

```
>> y = [1, 2, 3]
y =
       1       2    3
 >> y2 = [1 2 3]
 y2 =
       1      2     3
```

You now know how to define matrices. The ( ) operator allows you to access the contents of a matrix. MATLAB is 1-indexed, meaning that the first element of each array has index 1 (indexing starts from 1 not from 0).

```
>> y = [1, 2, 3];
>> y(1)
ans =
      1
```

To access a single element in a multidimensional matrix, use (i,j). The syntax is matrix(row,column):

```
>> y= [1, 2; 3, 4];
>> y(2, 1)
ans =
```

3

There is a quick way to define arrays that are a range of numbers, by using the **: operator**.

Frequently you want to create vectors whose elements are defined by a simple expression or to access a sequence of elements in a vector. The colon operator : provides a very convenient way of doing this.

In its simplest form, a:b starts at a, then increases in steps of 1, until b is reached.
Create a row vector of the integers from 1 to 10.

```
>>  x = 1:10
```

In this case, the square brackets are not required to create a row vector.

A slightly more general form of the colon operator is a:step:b, which starts at a, then adds step repeatedly, until b is reached (or exceeded).

If step > 0, then a should be less than or equal to b, and the final number is less than or equal to b.
Create a row vector y of the integers between 5 and 11 with a step of 1.

```
>>  y = 6:2:13
```

This produces 6, 8, 10, 12.

Exercise 5:

Define the following:
g = 0:25;
h = -10:10;
How big are g and h? Write the command to find their sizes.
What are the first, second, third, and last elements of each?
What exactly do g and h contain?

Create the following vector k as follows:
k = -10:0.1:10;
How big is k?
What are the first, second, third, and last elements?
What exactly does k contain?

4.4 Plot

The MATLAB command **plot** allows you to graphically display vector data in the form of (surprise!) a plot. Most of the time, you'll want to graph two signals and compare them. If you had a variable t for time, and x for a signal, then typing the command

The basic MATLAB graphing procedure, for example in 2D, is to take a vector of x-coordinates, x = (x_1; : : : ; x_N), and a vector of y-coordinates, y = (y_1; : : : ; y_N), locate the points (x_i; y_i), 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.

```
>> x = [1 2 3 4 5 6];
>> y = [3 -1 2 4 5 1];
>> plot(x,y)
```

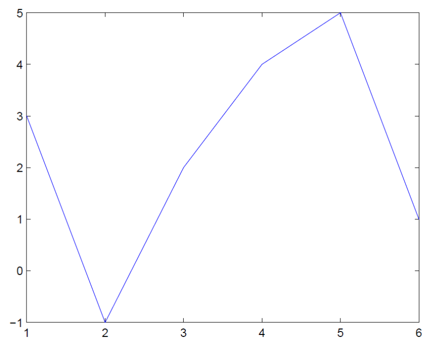


Figure 2

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

Table 1.1 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

Note: The plot functions have deferent 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.

```
>> y=[ 1 -2 3 -3 4 -6];

>> t=0: (length(y)-1);  t=t*0.1;  % To create a time vector of [0 .1 .2 .3 .4 .5];

>> plot(t,y);
```

will display a plot of the signal y against time (every 0.1 sec). However, the time vector `t` must have the same length as your signal `y.` See help plot if you haven't done so already.

You MUST label your plots in order to communicate clearly. Your graphs must be able to tell a story without you being present! Here are a few useful annotation commands:

```
>> title('Here is a title'); %– Adds the text "Here is a title" to the top of the plot.
>> xlabel('Distance traveled (m)');  %– Adds text to the X axis.
>> ylabel('Distance from Origin (m)'); % – Adds text to the Y axis.

>> >> hold off;
>> grid on;  %– Adds a grid to the plot.

>> grid off;  %– Removes the grid (sometimes the plot is too cluttered with it on).
>> hold on;  %– Draws the next plot on top of the current one (on the same axes). Useful for
comparing plots.
                    %– Erases the current plot before drawing the next (this is the default).
```

In order to display multiple plots in one window, you must use the subplot command. This command takes three arguments as follows: subplot(m,n,p). The first two arguments break the window into an m by n grid of smaller graphs, and the third argument p selects which of those smaller graphs is being drawn right now.

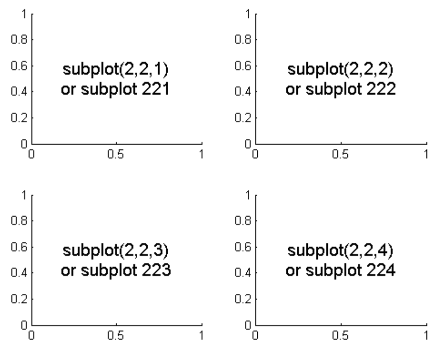
For example, if you had three signals x, y, and z, and you wanted to plot each against time t, then we could use the subplot command to produce a single window with three graphs stacked vertically:

```
>> subplot(3,1,1);
>> plot(t,x);
>> subplot(3,1,2);
>> plot(t,y);
>> subplot(3,1,3);
>> plot(t,z);
```

See help subplot or look online for more examples.

subplot Create axes in tiled positions.

subplot(m,n,p), or subplot(mnp), breaks the Figure window into an m-by-n matrix of small axes, selects the p-th axes for the current plot. The axes are counted along the top row of the Figure window, then the second row, etc. For example:



MATLAB only handles discrete representations of signals. As such, you need a way to represent time for continuous signals in MATLAB. All functions of time in MATLAB must be defined over a particular range of time (and with a particular granularity or increment). Note that the granularity (step size) of your time vector will impact the resolution of your plots.

<u>Exercise 6:</u>

Create and plot a signal $x_0(t) = te^{-|t|}$ using the following commands:

```
>> t = -10:0.1:10;
>> xo = t .* exp(-abs(t));
>> plot(t,xo)
```

Create the related signals $x_e(t) = |t|e^{-|t|}$ and $x(t) = 0.5 * [x_o(t) + x_e(t)]$.
What are $x_o(t)$ and $x_e(t)$, relative to $x(t)$?
Plot all three signals together in one window.

<u>Multiple data sets in one plot</u>

Multiple (*x; y*) *pairs* arguments create *multiple* graphs with a single call to *plot*. For example, these statements plot three related functions of *x*: $y_1 = 2\cos(x)$, $y_2 = \cos(x)$, 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)')
```

Please try to plot lines with different colors.

NOTES:

- `0:pi/100:2*pi` yields a vector that
  - starts at 0,
  - takes steps (or increments) of $\pi/100$,
  - stops when 2π is reached.