Appendix A

Basic MATLAB Tutorial

Extracted from:
http://www1.gantep.edu.tr/~bingul/ep375
http://www.mathworks.com/products/matlab

A.1 Introduction

This is a basic tutorial for the MATLAB program which is a high-performance language for technical computing for platforms:

- Linux
- Windows
- Mac OS

MATLAB integrates computation, visualization, and programming in an easy to use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include

- Math and calculations
- Algorithm development
- Symbolic and numeric computations
- Scientific and engineering graphics
- Simulations
- Data modeling
- Data acquisition
- Data analysis
A.2 Basic Commands

- `help command` get help for a command. i.e. `help fzero`.
- `clear` clears all variables in the memory (workspace).
- `whos` lists all the variables (and details) on the workspace.
- Semicolon (;) at the end will suppress the output
- Command history: upper and lower arrow keys on keyboard calls the previous commands.

A.3 Predefined Constants

The number $\pi$

```
>> pi
ans = 3.1416
```

The number $e$

```
>> e
ans = 2.7183
```

The virtual number $i$

```
>> i
ans = 0 + 1i
```

A.4 Formatting Output

There are three ways:

```
>> format short
>> pi
ans = 3.1416

>> format long
>> pi
ans = 3.141592653589793

>> format rat
>> pi
ans = 355/113
```
A.5 Variables

A variable is a symbolic name indicating a location in your computer’s memory where you can store a value which can later be retrieved.

In MATLAB, if you don’t use a variable name, your calculation is labeled and assigned by ans variable.

\[
\begin{align*}
\text{>> } & 3 + 4.5 \\ 
& \text{ans} = 7.5
\end{align*}
\]

You can store the result another variable.

\[
\begin{align*}
\text{>> } & a = 3 + 4.5 \\ 
& a = 7.5
\end{align*}
\]

Another example

\[
\begin{align*}
\text{>> } & g = 9.8; \\ 
\text{>> } & t = 1.2; \\ 
\text{>> } & h = 0.5 \times g \times t^2 \\ 
& h = 7.0560
\end{align*}
\]

MATLAB allows you to define complex numbers. For example, the following commands will evaluate the magnitude of a complex number:

\[
\begin{align*}
\text{>> } & z = 3 + 4i \\ 
& z = 3 + 4i \\ 
\text{>> } & m = \text{abs}(z) \\ 
& m = 5
\end{align*}
\]

Many built-in functions, such as sqrt(), log(), can be used for real and complex calculations:

\[
\begin{align*}
\text{>> } & c = \text{sqrt}(-5) \\ 
& c = 0.00000 + 2.23607i \\ 
\text{>> } & d = \text{sqrt}(4+4i) \\ 
& d = 2.1974 + 0.9102i \\ 
\text{>> } & k = \text{log}(i) \\ 
& k = 0.00000 + 1.57080i \\ 
\text{>> } & q = \text{log}(-7) \\ 
& q = 1.9459 + 3.1416i
\end{align*}
\]
A.6 Some Built-in Functions and Operators

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>sin(x)</td>
<td>sine of x (x is in radian)</td>
<td>sin(0.1) = 0.0998</td>
</tr>
<tr>
<td>cos(x)</td>
<td>cosine of x (x is in radian)</td>
<td>cos(0.1) = 0.9950</td>
</tr>
<tr>
<td>tan(x)</td>
<td>tangent of x (x is in radian)</td>
<td>tan(0.1) = 0.1003</td>
</tr>
<tr>
<td>sind(x)</td>
<td>sine of x (x is in degrees)</td>
<td>sind(30) = 0.5000</td>
</tr>
<tr>
<td>cosd(x)</td>
<td>cosine of x (x is in degrees)</td>
<td>cosd(30) = 0.8660</td>
</tr>
<tr>
<td>tand(x)</td>
<td>tangent of x (x is in degrees)</td>
<td>tand(30) = 0.5774</td>
</tr>
<tr>
<td>asin(x)</td>
<td>angle in radian from sin(x)</td>
<td>asin(0.8)= 0.9273</td>
</tr>
<tr>
<td>acos(x)</td>
<td>angle in radian from cos(x)</td>
<td>acos(0.8)= 0.6435</td>
</tr>
<tr>
<td>atan(x)</td>
<td>angle in radian from tan(x)</td>
<td>atan(0.8)= 0.6747</td>
</tr>
<tr>
<td>asind(x)</td>
<td>angle in deg. from sind(x)</td>
<td>asind(0.8)= 53.13</td>
</tr>
<tr>
<td>acosd(x)</td>
<td>angle in deg. from cosd(x)</td>
<td>acosd(0.8)= 36.87</td>
</tr>
<tr>
<td>atand(x)</td>
<td>angle in deg. from tand(x)</td>
<td>atand(0.8)= 38.66</td>
</tr>
<tr>
<td>abs(x)</td>
<td>$</td>
<td>x</td>
</tr>
<tr>
<td>log(x)</td>
<td>$\ln(x)$</td>
<td>log(2) = 0.6932</td>
</tr>
<tr>
<td>log10(x)</td>
<td>$\log_{10}(x)$</td>
<td>log10(2) = 0.3010</td>
</tr>
<tr>
<td>exp(x)</td>
<td>$e^x$</td>
<td>exp(2) = 7.3891</td>
</tr>
<tr>
<td>mod(x, y)</td>
<td>x modulo y</td>
<td>mod(12, 5) = 2</td>
</tr>
<tr>
<td>x^y</td>
<td>$x^y$</td>
<td>$10^3 = 1000$</td>
</tr>
<tr>
<td>1e+3</td>
<td>$10^3$</td>
<td>$1e+3 = 1000$</td>
</tr>
<tr>
<td>1e-3</td>
<td>$10^{-3}$</td>
<td>$1e-3 = 0.1000$</td>
</tr>
</tbody>
</table>

A.7 Vectors

An array or a vector is a group of objects, all of the same name and data type. The objects are called the elements of array and are indexed consecutively as 1, 2, 3, ···. In MATLAB, there are several ways to define arrays. One can use the column operator:

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

>> a = 1:2:5
a = 1 3 5

>> a = 1:0.5:5
a = 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

Or use linspace() function. For example linspace(1,5,4) generates a row vector from 1 to 5 with 4 equal steps, that is:
 Alternately you can directly input each element.

\[
\begin{bmatrix}
1 & 2 & 3 & 4 & 5 \\
\end{bmatrix}
\]

All commands above generate the row vectors. One can convert a row vector to a column vector as follows:

\[
\begin{bmatrix}
3.1 & 4.5 & -7.2 \\
\end{bmatrix}
\]

Hence if \( v \) is a row vector, then \( v' \) is its transpose. Here is another example:

\[
\begin{bmatrix}
1 & 2 & 3 \\
\end{bmatrix}
\]

One can join two column vectors to form a matrix as follows:

\[
\begin{bmatrix}
2 & 1 \\
4 & 3 \\
6 & 5 \\
\end{bmatrix}
\]
On the other hand, we may need product (multiplication) of vectors. The following operation is not valid for two row vectors:

```matlab
>> a = [2 4 6];
>> b = [1 3 5];
>> a*b
Error: inner matrix dimensions must agree.
```

However, one can perform a scalar (dot) product of two vectors as follows:

```matlab
>> a * b'
ans = 44
```

In MATLAB, there is an alternative vector product known as *element-wise multiplication*.

```matlab
>> a = [2 4 6];
>> b = [1 3 5];
>> a .* b
ans = 2 12 30
```

Similarly, *element-wise division* and *element-wise power* are also available.

```matlab
>> a ./ b
ans = 2.0000 1.3333 1.2000
>> a .^ b
ans = 2 64 7776
```
A.8 Basic Statistics With Vectors

```matlab
>> x = [1 2 2.5 3.1 3];

>> length(x)
ans = 5

>> sum(x)
ans = 11.6000

>> mean(x)
ans = 2.3200

>> std(x)
ans = 0.8585

>> min(x)
ans = 1

>> max(x)
ans = 3.1000

>> [value index] = max(x)
value = 3.1000
index = 4
```

A.9 Matrices

In MATLAB, any variable can be considered as a matrix. A scalar is $1 \times 1$ matrix, while a row vector of size $n$ is $n \times 1$ matrix and etc. A $2 \times 3$ matrix can be defined simply as follows:

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

and a $3 \times 3$ matrix may be:

```matlab
>> B = [1 2 3; 0 5 6; 7 8 9]
B =
    1 2 3
    0 5 6
    7 8 9
```
There are some basic functions to work with matrices:

```
>> B = [1 2 3; 0 5 6; 7 8 9];

>> det(B)
ans = -24

>> inv(B)
Ans =
     0.1250   -0.2500    0.1250
     -1.7500    0.5000    0.2500
      1.4583   -0.2500   -0.2083
```

Consider two matrices to multiply:

```
\begin{bmatrix}
  1 & 2 \\
  3 & 4
\end{bmatrix}
\times
\begin{bmatrix}
  0 & 1 \\
  8 & 5
\end{bmatrix}
```

```
>> M = [1 2; 3 4];
>> N = [0 1; 8 5];
>> M * N
ans =
     16    11
     32    23
```

### A.10 Special Matrices

```
>> P = zeros(2,2)
P =
   0   0
   0   0

>> P = zeros(2,1)
P =
   0
   0

>> Q = ones(2,2)
Q =
   1   1
   1   1
```
A.11 Basic Plotting

MATLAB has extensive plotting capabilities. Here we illustrate some basic commands for 2D and 3D plots.

You can obtain the histogram of a collection of data via `histogram` function. Consider you have 10,000 random numbers taken uniformly from the range (0, 1). The distribution of this data can be plotted as follows:

```matlab
>> r = rand(10000,1);
>> histogram(r,20);
```

Figure A.1: Histogram of 10000 uniform random numbers.

A simple plot of bi-variate data is performed by the `plot` function.
There are many options for a plot. Here is a slightly advanced code which sets axis titles.

```matlab
>> x = -3*pi:0.1:3*pi;
>> y = sin(x)./x;
>> plot(x, y)
>> xlabel('time (s)')
>> ylabel('speed (m/s)')
>> title('Speed vs time')
>> text(-9,0.8,'Non-uniform motion')
>> grid on
```
A.12 M-files

You can save all MATLAB commands that you have invoked in command window in an *m-file*. The m-file can be generated by *edit* command in MATLAB’s command window (or from the File Menu).

```
>> edit plot1.m
```

This will open a file called plot1.m in the editor. Then, type the following in the editor and save it.

```
x = -3*pi:0.1:3*pi;
y = sin(x)./x;
plot(x, y)
xlabel('time (s)')
ylabel('speed (m/s)')
title('Speed vs time')
text(-9,0.8,'Non-uniform motion')
grid on
```

In command window, you can run the command as follows:

```
>> plot1
```

Of course, this will create the same plot as in Figure A.2.

You can write a program to input a data from keyboard as well. To do that, you can use *input* function. The following script (saved in avr.m) reads two numbers from keyboard and outputs their averages.
APPENDIX A. BASIC MATLAB TUTORIAL

```matlab
a = input('Enter an integer: ');
b = input('Enter an integer: ');
c = (a+b) / 2;
printf('Mean of %d and %d is %f
',a,b,c);
```

Output is:

```matlab
>> avr
Enter an integer: 3
Enter an integer: 4
Mean of 3 and 4 is 7.500000
```

### A.13 Functions

A function is an object that accepts some inputs and then outputs a result depending on the inputs. In MATLAB general declaration of a function is:

```matlab
function [output_arguments] = fname(input_arguments)
function block
end
```

Here, `fname` is the name of the function. `input_arguments` and `output_arguments` of the function must be separated by commas.

To make the function accessible to other programs units, it must be saved under the filename `fname.m`.

The following function takes two sides of a right angle triangle and outputs the hypotenuse of the triangle.

```matlab
>> edit hipo.m

function z = hipo(x,y)
    z = sqrt(x^2 + y^2);
end
```

Now we can use the function in command window as follows:

```matlab
>> hipo(3,4)
ans = 5
```

A function may return more than one value. The following example converts a coordinate \((x,y)\) specified in cartesian coordinate into polar coordinate, \((r,\theta)\)

```matlab
function [r,t] = car2pol(x,y)
    r = sqrt(x^2+y^2);
    t = atan(y/x);
end
```
>> car2pol(3,4)
ans = 5

>> [radius, angle] = car2pol(3,4)
radius = 5
angle = 0.9273

You can send/get a vector of values in a function. The following function takes a vector, and returns another vector.

```
function r = fun(v)
    r = sqrt(abs(v) + 1.5));
end
```

If your function block is short, then you can define it as an `inline` function.

```
>> f = inline('x.^2');
>> x = 1:5;
>> plot(x,f(x))
```

### A.14 Selection

Control statements allow us to make decisions. In MATLAB, basic control (selection) statements are well known `if`, `if ... else` and `elseif` statements. As listed below, there are several operators used in control structures.

- `<` less than
- `<=` less than or equal to
- `>` greater than
- `>=` greater than or equal to
- `==` equal to
- `~=` not equal to
- `&` logical and
- `|` logical or
- `~` logical not

The following script (saved in evenodd.m) reads a number from keyboard and outputs a suitable message whether it is an even or odd number.

```
x = input('Enter an integer: ');
if mod(x,2) == 0
    disp('This is an even number.')
else
    disp('This is an odd number.')
end
```
In command window, an example run is given below:

```
>> evenodd
Enter an integer: 29
This is an odd number.
```

## A.15 Loops

Loops are used to repeat a block of statements in a program. In MATLAB, the famous loop statements are `while` and `for`. Here there are two examples.

```matlab
while 1
    disp('Hello Mars');
end
```

```matlab
x = rand(100,1);
N = 0;

for i = 1:length(x)
    if x(i) > 0.5
        N = N + 1;
    end
end
p = N / length(x)
```

## A.16 Reading Data Files

In most of the applications, experimental data are acquired by electronic devices and stored in a file. Consider you have data having two columns. First column is wavelength in nanometer and second is the intensity of light from an infrared source. First a few lines are given below. (You can download the file from course web page).

```
1.975717010e+002 1.283807331e-003
1.97773154e+002 5.922408891e-004
1.979749451e+002 1.192245516e-003
1.981765900e+002 7.430875557e-004
1.983782501e+002 1.193400705e-003
1.985799408e+002 6.572923739e-004
...```

To read and plot this data is very simple in MATLAB. If the data file is stored as infrared.txt then:
% read data and copy to vectors
[w, I] = textread('infrared.txt');

% plot data
plot(w,I,'color','red','linewidth',2)
xlabel('Wavelength')
ylabel('Intensity')
grid on

Figure A.4: The light distribution of an IR source.
A.17 Exercises

1. The average distance to the Sun is about 93 million miles. If the speed of light is about \( c = 3 \times 10^8 \text{ m/s} \) then how many minutes and seconds does it take Sun light to reach the earth? Solve this problem using MATLAB script. (Hint:1 mile is about 1.609 kilometers)

2. Write MATLAB script to evaluate the sum
\[
1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots - \frac{1}{1000001}
\]
with a for loop and time the computation using tic and toc.

3. Write a program to find the closest value of the number \( \pi = 3.141592653589793 \cdots \) obtained from a rational number of the form \( r = A/B \) where \( A \) and \( B \) are 3 digit integers.

4. Write a function named isPrime(n) that returns 1 if \( n \) is a prime number and returns 0 otherwise. Example function call in MATLAB command line:

\[
\text{>> } x = \text{isPrime}(3) \\
x = 1
\]

5. Write a function of the form rmaxval(x) to remove the largest element(s) from a vector \( x \). Example call in command window:

\[
\text{>> } a = [1 2 5 0 0 5]; \\
\text{>> } b = \text{rmaxval}(a) \\
b = 1 2 0 0
\]

6. Modify the program given in A.16 to get maximum intensity value and corresponding wavelength. See Figure A.4.