Lab Session 1 Introduction to MATLAB Help Sheet

Key aims of the Session:

- Familiarise with MATLAB
- Learn some essential commands
- Work with vectors and matrices
- Learn some commands commonly used data science and machine learning

Contents

1: Introduction to MATLAB

MATLAB^{TM} is an interactive programming language that can be used in many ways, including data analysis and visualisation, simulation and engineering problem solving. It may be used as an interactive tool or as a high-level programming language.

Specialist toolboxes are available for a diverse range of applications including statistical analysis, image processing, fuzzy logic, neural networks and so on. Furthermore, real time toolboxes allow for on-line interaction with real physical systems, ideal for data logging and control.

2: Getting Started

To start MATLAB, click on the MATLAB icon on the desktop. The main MATLAB window with the command line prompt >> should appear.

2.1: Entering Commands

MATLAB is an interpreted language. This means that the instructions you give to it are processed and interpreted by the computer directly, as opposed to for instance C: in C, when you write a program, you first need to compile it before it can be executed by the machine. This means that at any moment, all the variables are in the memory of the computer (equivalent to debug mode of C/C++) and can be readily accessed. This is a tremendous help to quickly debug programs.

There are two ways to give instructions to MATLAB:

1. On the **command line**, after the prompt >> if you type an instruction, MATLAB executes it and gives you the result.
2. Through a script or a function. In both cases, code is stored in a text file with **.m** as the extension.
   Scripts contain lists of instructions, just as could be written on the command line. To execute a script, you can type the name of the .m file (without the .m extension) on the command line or open the script and click Run on the menu bar. When a script is executed, all the instructions it contains are executed successively.
   Functions are modular scripts that can take in arguments. They will not be detailed here, but you can look at the on-line help to find out about them.

2.2: Statements and Variables

Statements have the generic form: variables = expression

The equal symbol = implies the assignment of the expression to the variable. The symbol **:** = is not used for assignment in MATLAB.

A typical statement is:

```
>> x = 3
```

```
x = 3
```

Here the number 3 is assigned to the variable name x. The statement is executed immediately after the enter key is pressed. The value of x is automatically displayed after the statement is executed. If the statement is followed by a semi-colon ***;*** then the value is not displayed. Thus, now typing the statement:

```
>> x = 4;
```

would not result in any output on screen but the assignment will still have been carried out. You can confirm this for the above example by checking the current value of the variable:

```
>> x
```

```
x = 4
```

2.3: Comments

Most languages provide some functionality for entering comments in code. It is good practice to use comments to explain what your code is doing, either for your benefit later or for the benefit of others who are reading your code. In MATLAB, you can use `%` to denote a comment.

2.4: Working with Directories

In order to work with stored files, such as scripts or image files, MATLAB needs to be able to find them. You can enter the whole file path, but this will need to be changed if you want to run the code on another computer, which is not convenient. This is an example of "hardcoding." More common and transferrable practices are to either:

1. Store the file in your current working directory or the same directory in which your script is stored.
   Hint: Type `pwd` to find out where you are.
2. Use relative file paths to reference the file. This can allow you to organise your files into folders without needing to hardcode the full address.
   HINT: To reference a file in a sibling directory, you need to use "`../`" to go to the parent directory. For example: `load('../textfiles/example.txt')`
3. You can make the folder's path known to MATLAB by typing `addpath(foldername)` or adding this line to your script. Alternatively, go to the File menu and Set Path sub-menu to change the path. This will make all of the contents of the folder accessible as would be if it were in the main directory.
   HINT: You can type `path` to see the full path, and type `help path` if you need to change it.

Common Commands:

Use the `pwd` command to know where you are.

Use the `dir` command to list the files in your current directory.

Use the `cd` command to change directory.

2.5: Workspace and saving results

MATLAB stores your variables in a workspace. You can list these variables, clear them from memory and export them to files using the following commands:

`>> whos` will list the variables in the workspace:

`>> clear` will clear the variables in the workspace:

`>> save filename.mat` will output the variables in the workspace to the file filename.mat. This will be stored in the working directory.

`>> load filename.mat` will load the data from the file filename.mat into the workspace

HINTS:

- Consider how you can clear only specific variables from the workspace, or how you can clear all variables except for specific variables.
- Consider how you can save specific variables from the workspace or load only certain labels from a file.

2.6: Getting Help and Demos

You can get help on all commands in MATLAB by typing the `help` command. For example, type `help help`. Beware, MATLAB is case sensitive.

Other useful help commands:

`>> help general` – general purpose functions.

`>> help ops` – operators and special characters.

`>> help elfun` – elementary maths functions.

You can browse the range of toolboxes available in MATLAB by clicking the *fx* dialog to the left of the command prompt, or view the MATLAB documentation by just typing `doc`.

You can access a demonstration by typing `demo matlab`. This will open a window from which you can select a demo for different tools.

3: Working with Vectors and Matrices

Vectors and matrices work particularly well in MATLAB. Operations can be performed on vectors and matrices directly without the need of for loops, which is particularly useful for tasks such as image processing.

3.1: Creating Vectors and Matrices

A vector can be created using square brackets. A typical vector statement is:

```
>> x = [1 3]

x = 1 3
```

Here the numbers 1 and 3 are assigned to an array or vector (i.e. a list of numbers) with the variable name x.

Try the following commands in MATLAB:

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

>> x

>> who

>> whos

>> y = [6; 7; 8; 9; 0; 9; 8; 7; 6]

>> y'

>> z = [1 2 3; 4 5 6; 7 8 9; 0 1 2]
```

HINTS:

- When creating an array, such as a vector or matrix: use a space “ “ or comma “`,` “ to move to the next column; use a semicolon “`;`” to move to the next row.
- Use a semicolon at the end of the instruction if you want to suppress the output.
- Use an apostrophe “`'` ” to transpose a vector or matrix.
- Type “`who`” to list the variables in your workspace and “`whos`” to list the variables with more details, including the size.
- To find out the size of a matrix A, you can type `size(A)`

To create larger vectors, you can use the colon ":" operator. Try the following commands. What is the meaning of using two colons?

```
>> a = 0:10
>> b = -5:10
>> c = 0:2:10
>> d = 10:-2:-5
>> e = 0:0.01:4.2
>> f = -pi:0.01:pi
```

To initialise a new matrix, you can use the `zeros` or `ones` commands. If you want a diagonal matrix, you can use the eye command. You can also create a matrix of random values using the `rand` or `randi` commands. Try the following:

```
>> g = zeros(3,1)
>> h = zeros(2,4)
>> i = ones(5,3)
>> j = eye(5)
>> k = rand(3,2)
>> l = randi(5,3,2)
```

HINT: Type "`help rand`" and "`help randi`" to find out the difference between these commands and what the arguments mean.

3.2: Matrix elements

Individual elements of a matrix may be extracted using the column and row index. Note that in MATLAB, row and column indices start from 1, not from 0.

Create a new matrix, extract the element in the third row and second column and assign the value to a new variable m:

```
M = rand(4,3)
>> m = M(3, 2)
```

The colon character, ":" is used to extract an entire row or column:

```
>> x = M(2,:)
>> y = M(:,2)
```

The colon operator can also be used to pick out a range of rows or columns. The statement `M(a:b,c:d)` means that rows a to b and column c to d will be extracted. E.g.:

```
>> M(2:3,1:2)
```

You can also use indices to change an element or multiple elements of a matrix. E.g.:

```
>> M(3,3) = 0
>> M(2,:) = 1
>> M(2:3,1:2) = 2
```

The last row or column can be referred to with the keyword `end`:

```
>> M(:,end)
```

A row or a column of a matrix can be deleted by setting it to a null vector, `[]`.

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

```
>> M(1,:)=[]
```

It is important to note that the colon operator "`:`" stands for all columns or all rows. You can create a vector version of matrix A, stacking each column on top of the other by typing:

```
>> M(:)
```

3.3: Dimension

To determine the dimensions of a matrix or vector, you can use the command `size`. For example,

```
>> size(M)
```

```
ans =
3 2
```

means 3 rows and 2 columns.

You can assign the size values to variables:

```
>> [m,n]=size(M)
```

```
m =
3
n =
2
```

The statement `d = size(M, n)` returns the length of the dimension specified by the scalar n. For example, `size(M,1)` is the number of rows of M and `size(M,2)` is the number of columns of M.

3.4: Arithmetic Operators

The arithmetic operators perform addition (+), subtraction (-), multiplication (*), division (/), exponentiation (^), and modulus (mod) operations.

To try these in MATLAB, create a few variables:

```
>> x1 = 2;

>> x2 = 3;

>> y1 = randi(5,5,5);

>> y2 = randi(5,5,5);
```

Now you can try some operations with these, e.g.

```
>> x1+x2

>> y1-y2

>> x1+y2

>> x1*x2
>> y1*y2

>> y1/y2

>> y1^x1

>> mod(x2,x1)
```

NOTE: What happens in the third example? Try adding matrices of different sizes.

HINT: In contrast to some other languages, you cannot use `%` to calculate the modulus since this is used to identify comments in MATLAB.

The division, multiplication and exponent operators carry out matrix operations[1] but often you want to do pointwise (or element-by-element) operations. You can do this by placing a full stop before the operator, i.e. you can use "`./`", "`.*`" or "`.^`". Try the following and compare the results with the commands above:

```
>> y1.*y2

>> y1./y2

>> y1.^x1

>> y1*y1

>> y1.*y1

>> y1^2
```

3.5: Error Messages

If an expression is entered 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.
```

It is important to read the whole of the error message to try to understand the cause of the error.

To fix the error, we can, of course retype the expressions. A convenient way to recall a command so that it can be edited is to use the up-arrow key ↑. When the command is displayed at the command prompt, it can be modified if needed and executed.

[1] See https://en.wikipedia.org/wiki/Matrix_multiplication for example, if you are not familiar with matrix multiplication.

4: Visualisation:

Graphics play an important role in design and analysis. It can be important to visualise vectors and functions.

Available plot functions in MATLAB include:

`plot(x,y)`	plots the array x versus the array y
`stem(Y)`	plots the data sequence Y as stems from the x-axis terminated with circles for data value. If Y is a matrix then each column is plotted as a separate series.
`bar(X,Y)`	draws the columns of the M-by-N matrix Y as M groups of N vertical bars.

The axis scales and line types are automatically chosen. However, graphs may be customised using the following functions:

`title('text')`	puts 'text' at the top of the plot
`xlabel('text')`	labels the x-axis with 'text'
`ylabel('text')`	labels the y-axis with 'text'
`text(p,q,'text','sc')`	puts 'text' at (p,q) in screen co-ordinates
`subplot`	divides the graphic window
`grid on` and `grid off`	draws grid lines on the current plot (turns on or off)

To try this, create some vectors to plot:

```
>> t =-pi:0.01:pi;

>> c = cos(t);

>> s = sin(t);
```

HINT: The trigonometric functions assume that the argument is in radians, not degrees.

Create a figure and plot the function sin(t):

```
>> figure

>> plot(s)
```

Create another figure and plot the function sin(t) against the vector t:

```
>> figure

>> plot(t,s)
```

Add labels to the x- and y- axes:

```
>> xlabel('x axis')

>> ylabel('x axis')
```

Add a title to the plot:

```
>> title('Plot Title')
```

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 1.

Table 1: Attribute 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	x	Cross
m	Magenta			s	Square
y	Yellow			d	Diamond

Create another figure and plot both functions against the vector t, with sin(t) in green and cos(t) in red:

```
>> figure

>> plot(t,s,'g',t,c,'r')
```

HINTS:

- Consider in the above examples what each of the arguments does. E.g. how do you plot a function in green or red? How could you plot it in blue?
- A figure window is brought up automatically when the plot function is used, but if you want a new figure window, the `figure` command needs to be used.
- Multiple plots may be open at one time. For your commands to apply to a particular figure, you need to identify the figure by a number, e.g. `figure(3)`. This number is a unique reference that will appear on the figure window. You can specify this number yourself when creating the figure by typing, for example, `figure(3)`.
- Type `help plot` to see more options for this function.

5: MATLAB scripts

5.1: For loops

A for loop allows a statement, or group of statements to be repeated a fixed predetermined number of times. For examples:

```
>> for i = 1:5; x(i) = i *2; end

>> x

x = 2 4 6 8 10
```

5.2: Conditional statements

The MATLAB function “`if`” conditionally executes statements. The simple form is,

```
if expression
statements
else
statements
end
```

MATLAB supports the variants of “`if`” construct.

```
    if ... end
    if ... else ... end
    if ... elseif ... else ... end
```

HINTS:

- `elseif` has no space between else and if (one word)
- No semicolon (`;`) is needed at the end of lines containing if, else, end
- Indentation of if block is not required, but does facilitate reading.
- The end statement is required

6: External data

MATLAB can import data in a number of formats, including simple text, spreadsheet files, image files, sound files and even movies. The load command is used to import ASCII text. Typically, such a file contains collected data.

For example, use a text editor (such as *nano* or *gnome text editor* in Ubuntu) to create the following file called test.txt and save it to your working directory. Here, the first column might be the time, while the second column is a measurement, say a voltage. The data file should contain only numbers separated by commas or spaces:

```
10, 8
20, 11
30, 16
40, 15
50, 18
```

To load the file, in MATLAB type:

```
>> load test.txt

>> who

>> test
```