LECTURE 2 – INTRODUCTION TO MATLAB

In this chapter you will learn about:

- Accessing MATLAB;
- using MATLAB as a calculator;
- MATLAB identifiers, constants and functions;
- using the MATLAB help system;
- making `.m` file scripts;
- plotting simple line graphs.

1.1 What is Matlab?

MATLAB is often described as a problem solving environment. It is a programming language and set of tools for solving mathematical problems.

The name MATLAB was originally a contraction of "Matrix Laboratory" and indeed MATLAB's core strength is numerical computing involving matrices, vectors and linear algebra. It also has extensive plotting and graphics routines.

1.1.1 What is the Symbolic Math Toolbox?

The Symbolic Math Toolbox is an add-on for MATLAB which adds symbolic computing abilities. The strength of such a system is the ability to manipulate *algebraic* expressions. The Symbolic Math Toolbox can be used to solve algebraic equations, factorize polynomials and simplify rational expressions. It can also solve some differential equations. It can perform many of the standard operations of analysis such as evaluating sums, limits, derivatives and integrals. Under the hood, the Symbolic Math Toolbox is powered by a computer algebra system called MuPAD.

The combination of MATLAB and the Symbolic Math Toolbox allows the user to mix symbolic and numerical computing within the MATLAB environment.

1.1.2 What is Octave?

Some people are concerned about the ethical issue surrounding computing. Should software be freely available? If these issues are important to you, you might want to consider Octave (`http://www.octave.org`) as an alternative to MATLAB. It is Free Software (also known as

Open Source Software) which means you can use it without restriction, study its source code, improve it, and share it with others as you wish. None of these things are true of MATLAB.

OctSymPy (see http://github.com/cbm755/octsympy) is an add-on package that adds symbolic computing features to Octave (it is analogous to the Symbolic Math Toolbox for MATLAB). OctSymPy is new software developed right here at Oxford! But beware: it is still "beta" quality.

The syntax of commands in Octave is almost the same as MATLAB and the differences are well-documented. But having said that, this manual will concentrate on MATLAB and this is likely the tool your demonstrators will know best. Try Octave if you wish, but do so "at your own risk."

1.1.3 Learning Matlab

In order to become technically adept at using MATLAB it is important that you attempt all the exercises contained in this manual; as with most tools, MATLAB is best learned by actually *using* it to *do* mathematics, and this should be practised as often as possible. Try to incorporate MATLAB into your weekly problem sheets, by using it to check some of your hand-written answers. Examples include substituting your solution back into the equation and plotting an answer to make sure it makes sense.

In the rest of this chapter, we will give a brief introduction to MATLAB, outlining some of its basic features and illustrating them with some short examples. The output of each command line is not printed in this manual; after executing each line you should check that the output is as you would expect. If it is not, then you should think about why this might be, and if necessary consult your demonstrator.

1.2 The command prompt, variables, getting help

Note that in MATLAB the prompt is the symbol ">>". Commands are entered to the right of the prompt, followed by the return/enter key. To illustrate this try entering the following:

```
>> 12/15
```

which should display `0.8`. Try some more commands

```
>> 3 + 18
>> 3+18
>> 3   + 18

>> 40*21
>> factorial(3)
```

1.2.1 Using MATLAB's built-in help system

There are various ways of getting help on MATLAB commands but one of them is built into the command prompt and is a good "first stop" for help. To use it, type `help` followed by the name of a command:

```
>> help factorial
```

You can also access help in various ways using the graphical user interface. (Don't worry if much of the information in the help text doesn't make sense yet.) Note that this help tends to refer to functions in CAPITALS but MATLAB is case sensitive and usually the actual

function is lowercase. Presumably this is for historical reasons, left over from a bygone era before copy-and-paste. Throughout this manual you should use the help facility to learn more about the commands you are introduced to.

1.2.2 Assigning to variables

Numbers can be assigned to variables

```
>> a = 4
>> b = factorial(4)
>> B = 5
```

Later you can recall the value of a variable

```
>> a
>> b
```

and variables can be used in further calculations

```
>> my_var = a^2 + b/10 + 2*B
```

It is important to think about what the symbol `=` means here: MATLAB uses it as the *assignment operator*. `B = 5` means "assign the value 5 to B". You could pronounce this as "B gets 5". You will sometimes see people writing the assignment operator as "$B \leftarrow 5$" and some other programming languages use "`B := 5`" to avoid confusion. For better or for worse, MATLAB uses `=`. Later on, we will meet the equality operator `==` which tests for equality and expresses symbolic equality: it deserves to be pronounced "equals".

There are a few words that are reserved for MATLAB operations and so cannot be used for variable names. Examples include `while`, `for`, and `function`: we will encounter many of these later the course and if you try to use one as variable name you'll usually get an error. But just because something is allowed doesn't mean it's a good idea:

```
>> pi = 3.2
>> sin(pi)
```

This seems like a good time to show you how to clear all variables.

```
>> clear
>> pi     % back to its usual value
>> a      % gives an error
```

Indeed, it is recommended that you start all exercises with `clear`. Let's say that again in bold:

Usually, you should start exercises with `clear`.
When something doesn't work as expected, try `clear`.

A semicolon suppresses the output (this is useful, for example, if the output is extensive and not specifically required). It can also make your code and output more readable: this will be more important as you start writing longer scripts in MATLAB.

```
>> a = 6;
>> b = 3;
>> c = a*b;
>> c = 2*(c + 6)
```

1.2.3 Floating-point numbers

By default MATLAB produces numerical answers which are often approximations (albeit highly accurate approximations)

```
>> 5/3
>> pi
>> sqrt(2)
```

These numerical solutions are typically accurate to about 15 decimal places:

```
>> sin(0)
>> sin(pi)
```

Note the latter gives `1.2246e-16`, that is 1.222×10^{-16}. You can increase the number of displayed digits with

```
>> format long
>> sin(pi)
```

but note this has no effect on the actual accuracy of the expression, just its display.

Internally, MATLAB by default stores numbers in "IEEE Double-Precision Floating Point" or just `double` for short. Each `double` takes 8 bytes (that is 64 bits, "binary digits") in the memory of the computer. The 64 bits is used to store numbers in scientific notation with a certain number of bits for the exponent (the "-16" in our example above) and the rest for the coefficient (1.222 in our example).

Computers are very fast at doing arithmetic on doubles and this forms the basis for much of computation. For example, encoding/decoding your voice, music, and video on your phone, performing large-scale climate simulations, finding a location from GPS satellites, or finding eigenvalues of large matrices all involve billions of operators on doubles. A surprising number of things we do (e.g., searching the internet) are just special cases of "finding eigenvalues of large matrices".

However, sometimes it's nice when 6/9 becomes 2/3 not "0.666666666666667". Particularly when learning or studying mathematics, we are often less interested in the numerical solution of something but rather in the how and why of it. Think about the area under a curve (a number, rather easily approximated) versus the indefinite integral (a general expression). We want the computer to do both these tasks.

1.2.4 Symbolic computing

Symbolic manipulation software or computer algebra systems do work with symbols rather than numbers. They try to work out derivatives and integrals and solve equations much the same way you do: by manipulating the symbols according to certain rules ("6/9: 6 is 2*3 and 9 is 3*3 so cancel the 3's to get 2/3").

The Symbolic Math Toolbox adds symbolic computing to MATLAB. We can check if we have the toolbox with the command:

```
>> ver
```

which might produce a lot of output but the important bit for us is

```
Symbolic Math Toolbox                Version 5.11      (R2013b)
```

We will use features in this manual that were introduced in Release 2012a so you'll need at least that version. A few commands may require Release 2013b.

With the toolbox, you use enter symbolic expressions at the MATLAB prompt:

```
>> sym('6/9')
>> sym('1')
>> a = sym('pi')
>> sin(a)
>> b = sym('sqrt(2)')
>> b^2
```

Those are single quotes which surround a string. Notice that you can also assign symbolic expressions to variables.

In fact, you can leave out the quotes when entering small integers or simple fractions.

```
>> sym(1)
>> sym(1000)
>> sym(6/9)
```

It is recommended to use this shortcut only for integers and very simply fractions, and to revert to the quote form if there is any doubt. The following exercise should convince you there can indeed be doubt.

Exercise 1.1 Observe the output of `sym(13/10)`, `sym(133/100)`, `sym(1333/1000)` and follow the pattern a few more steps. What happens? Try the experiment again with quotes (`sym('13/10')`, etc). (If you're curious, `help sym` explains how the quoteless mechanism works). □

In general, you probably don't want to use decimal places inside the `sym` command.

```
>> sym('6/9')    % yes
>> sym(6/9)      % sure, good too
>> sym('6.0/9')  % probably not what you wanted
```

The latter will use "variable precision arithmetic" which we won't make much use of in this course: "`help vpa`" if you want to know more, or see Appendix B.2.

1.2.5 Saving your work in `.m` file scripts

Instead of just typing commands into the command prompt, you can create a script. Within the graphical user interface, one way to do this is to select "file, new, script" from the menu bar. Add some commands to the file, for example:

```
% my solution to Exercise 1.1
sym(13/10)
sym(133/100)
```

Save the file as "`ex1_1.m`". From within the editor, you can run your script by clicking on the icon with the green triangle and the white square. You can also switch to the command prompt and type the name of your script (without the `.m`)

```
>> ex1_1
```

In either case, the contents of the script should execute in the command window. You can alternate between editing your script and running it.

It is recommended to save each exercise as a script and you may want to use additional scripts as you work through this manual. Find what works for you but remember you may want to access this material later (for example, next term when you work through the projects).