Introduction To MatLab for LMH Coding Society

Gorwarah, Tajmeet

April 18, 2020

Chapter 1

Introduction to MatLab

Hey guys! The majority of the material in this guide is taken from math students' Computational Mathematics course material written by Dr Vidit Nanda. I have modified the things that I thought needed to be simplified but for the most part it will be quite similar in structure.

1.1 What is MatLab?

The following is an excerpt taken from the notes for Computational Mathematics Students' Guide by Dr Vidit Nanda:

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 Symbolic Math Toolbox

This is the crux of the main usefulness of MatLab. Everyone knows that computers work with numbers only (think binary), but with MatLab we can use something called the *Symbolic Math Toolbox*, which is used for doing **algebraic expressions/manipulation** that we are all used to in doing any kind of maths. This can be used to solve algebraic equations, factorisation of polynomials and simplifying complicated expressions (although sometimes humans do it better). In school, if you haven't done any maths, then topics such as *integrations* and *differentiation* or just *calculus* in general is very daunting for people with little background in these topics. This language helps with that issue, as it can actually evaluate *limits, sums, derivatives* and *integrals* with just a few commands.

The origins of *Symbolic Math Toolbox* is well documented on internet. For those interested, lookup the computer algebra system called *MuPAD* which is the main powerhouse of *Symbolic Math Toolbox*.

1.1.2 Open-source Alternative

As mentioned in my Facebook post, MatLab is not an open source software, and that means that people need to acquire a license (which is quite expensive). Fortunately for us, we can use the University's resources to get a license for free. However, that also means that after we graduate, we won't have access to it unless we can shell out the licensing fee. But fear not! as there is an amazing open source alternative for MatLab called *Octave*. This is a software that is almost the same as Matlab in its syntax and works almost exactly like MatLab (not to say there are no differences but for the most part it's kind of the same software just free). This includes the *Symbolic Math Toolbox* which is called *OctSymPy* which is actually being developed here in Oxford!

Note that the differences are well documented on the internet and if you would like to use *Octave*, feel free to do so, however this guide will primarily focus on MatLab (in the classes you can bring the issues to me so that I can see what went wrong with MatLab and *Octave* code.)

1.2 The command prompt (or Terminal) and getting help

When you open the MatLab program, the symbol "$\gg$" is called the prompt. This is where you can enter commands and it gives a response similar to how unix/linux terminals and windows cmd/powershell works.

To illustrate its usefulness, lets try and use it as a calculator. Try the following commands:

```
>> 5/10
>> 1+2
>> 2*50
>> factorial(5)
```

Now you might be wondering what this factorial thing is? This is a nice segue to our next useful topic.

1.2.1 The *help* command

There are many ways to get help for MatLab commands (like the internet guides) but the main one is to use the in-built command called *help*. This is a command that I fall back on when I don't know what is happening (trust me you're not alone). To see how it works, try the following command in the terminal:

```
>> help factorial
```

Your screen would now be showing how to use this command/function. Throughout this tutorial, you should refer to help command to see how everything makes sense (because in the beginning it doesn't). Using help on commands that you know how to use is the best way to learn because that way you understand how their documentation works and then you can apply your knowledge later to other commands that you don't know.

1.2.2 Variables

Now, as we know how to use MatLab as a calculator, we also need to be able to store values (i.e. numbers) to some variables to make our life easier. There are quite a few uses of using variables (for those of you who have only worked with functional programming this will be a difficult pill to swallow - jokes, I love functional programming too)

Anyways, try writing this in the prompt:

```
>> a = 1
>> b = 5
>> c = a + b
```

After you do this, you can later call these values by just typing in the prompt there variable name like such:

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

The assignment is done from right to left, I will explain more about this in the class.

The "=" is the assignment operator which basics takes the value from the right and stores it in the left. For example, $a = 1$ is basically we telling the computer to take "1" and store it in "a". Note that this only works for variables, if you try something like $1 = 2$ you will most likely get a logic error (I think). Now keep in mind that there are some times that we want to compare values on the left and the right. I'm sure everyone is familiar with the "$>$" and "$<$" symbol. These are comparative operators, but sometimes we want to see whether something on the left is equal to the the thing in the right, for this reason we use the *logic equal to* operator "==". We will come back to the *logic operators* later when we are working on control flow (conditional statements). For now, just know that "=" and "==" are different operators.

Now that you have seen that you can store value, you might be wondering how certain irrational numbers like π might be stored. Matlab comes with quite a few pre determined words which we are not allowed to use as variable names these include words like *while, for,* and *function.* In the case of π, we can assign a value (I think) but just because its allowed doesn't mean its a good idea. For example, if you type:

```
>> pi
```

You will see that you get a value of pi for quite a few decimal places but if you type:

```
>> pi = 3.2
```

You wouldn't come across any errors (I think), but this causes the program to lose precision if the program uses the value of π.

clear **command**

In order to return everything back to normal, type the command:

```
>> clear
```

This will remove all the stored values and return to its initial state where nothing was stored.

Note that when things don't work as you'd expect them to and you are sure that the code you've written is correct then it's worth it to try to clear everything and rerun because sometimes we store things that we have forgotten about.

save **command**

This is the opposite of the *clear* command, in terms of its function. It saves all the current variables into a file that you can name in the current folder. Try understanding this command from the *help* function. I will give you an example of this in the tutorial. (This is mostly useful if you have experiments and don't want to lose what you've got so far).

```
<++ insert code here ++>
```