MATLAB - An Introduction for Mathematicians

Scott Morgan

July 19, 2016

Introduction - The Matlab Interface

- Matlab is a computational software package originally written by numerical analyst Clive Moler in the 1970s.

- Unlike other programming languages used in numerical computations like Fortran and C++, Matlab makes use of highly developed external libraries and functions which allows you to relieve yourself of many of the mundane tasks associated with programming. For example, Matlab has built in functions that can calculate matrix inverses, products and eigenvalues. All of these things would have to be hard-coded into a Fortran program, eating up time that could be spent on more important issues.

- With a very readable and easy to navigate interface, Matlab is a great starting tool for anyone interested in programming.

- Although it has many qualities, nothing is perfect and what it gains in usability it loses on computational time. A program written in Fortran or C++ will run a lot faster in general than the same program written in Matlab, something which may be of concern for very large programs. However, this is most certainly not of concern here, and all of the programs written in this course take only a fraction of a second to run on a workstation.

- It should be noted that these notes are designed only as a companion to the classes which they accompany - a student should not expect to gain any insight from these notes without attending class.

- E-mail corrections and/or comments are always welcome and can be directed to `MorganSN@cardiff.ac.uk`

Installing Matlab

- It would be superfluous to repeat the already thorough installation guidelines present on the MathWorks website, found here:
 `http://uk.mathworks.com/help/install/ug/install-mathworks-software.html`

- A student Matlab subscription is currently £29[1] and is highly worth the spend. If you are currently a postgraduate student at Cardiff then you can get the academic license key for installation on your home computer from Paul Jewell (M/0.10 - `jewellp@cardiff.ac.uk`) for free.

- If you would rather not pay then you can either use Matlab at any of the workstations around the university, available through *Cardiff Apps*, or you can use GNU Octave.

- GNU Octave is an open-source version of Matlab which uses many of the similar commands. However, as with everything in life, you get what you pay for and thus, Octave is buggy, slow and lacking certain features that may make your life a misery.

- Installing Octave can also be a pain, although it is described in detail on the web at `https://www.gnu.org/software/octave/download.html`. If you are stuck with the installation, email `MorganSN@cardiff.ac.uk` and I'll help in any way I can.

- Installing octave on Ubuntu is easy and can be accomplished with

```
$ sudo add-apt-repository ppa:octave/stable
$ sudo apt-get update
$ sudo apt-get install octave
```

---

[1]Correct as of 19/07/2016

Using Matlab for the first time

- On opening Matlab for the first time you should be presented with a window similar to this:
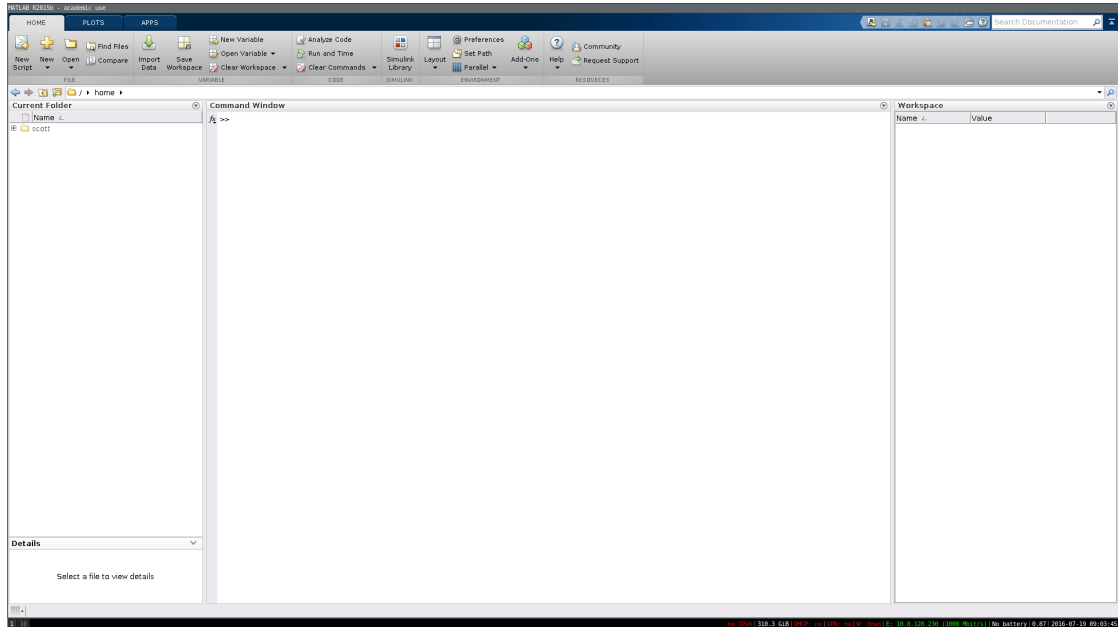
- The main window in the centre is the *command window* and contains the *command prompt.* You can enter commands directly at the command prompt and Matlab will give some sort of output. The left hand side shows the current directory and the folder structure. This will be important later.

- Calculations can be carried out in the usual way and follow the usual rules for ordering of operations. The figure below shows some example commands. You should also note now that the *workspace* on the right hand side contains a new value. This *ans* variable takes the value of the last entered command - in this case, 5.75.
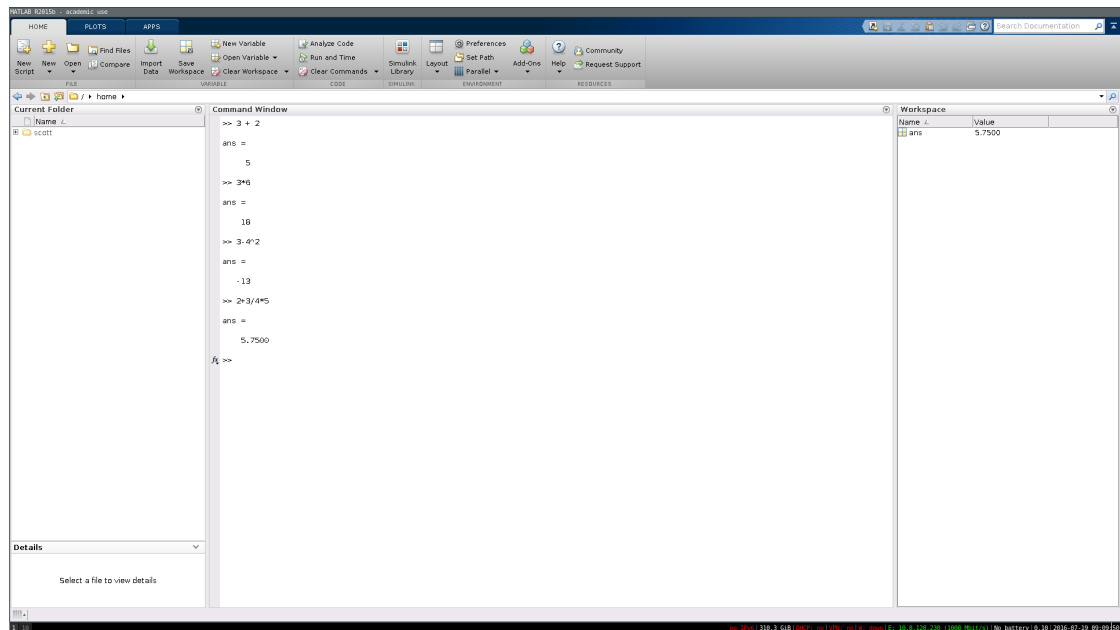
- While in theory you can work entirely in the command window, in practise this will get very untidy very quickly and so most of the calculations we will do will be inside what are called *scripts*.

- Scripts are basically text files which can be *called* from inside the command window. Follow the procedure below to create a script.

  - In the top left corner of the screen, click the yellow plus sign above the word *new*.

  - Click script.

  - A window called *Editor* should now appear above the command window, with the script named as *untitled*.

  - To use the script, type a calculation into the Editor and save the file *in the current directory* as *test.m*

  - The script is then *called* in the command window by writing

```
>> test
```

at the prompt. The figure below shows the process.
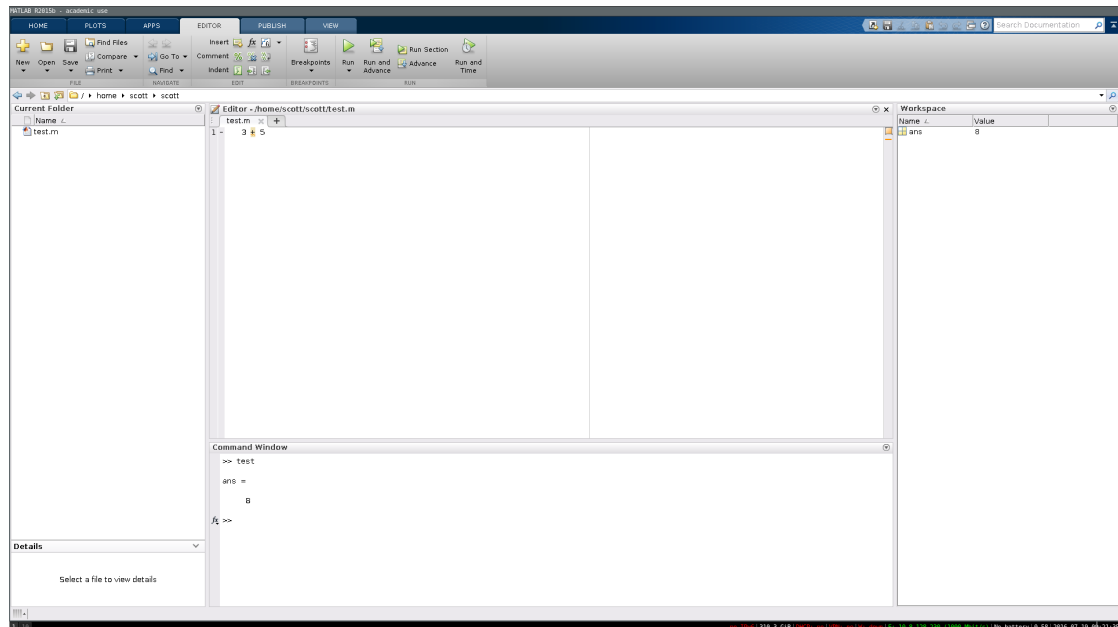
- Now that you know the basic ideas behind creating a script and running commands in Matlab, you can start to have a play around with the interface. Matlab has lots of built in functions like *sin, cos, tan, exp, log, abs, pi, sqrt* and many more. The best resource by far is Google! If you think Matlab may have a built in function that you want to use then chances are it does.

- The command

```
>> demo
```

is a great place to start learning about what everything does.

Remarks

- Before we start getting into the basics of coding, there are a few remarks to make.

  - There is more than one way to write a program! People have *styles*, as you'll begin to see soon. If your program is different to mine then that's not a problem - you will develop your own way of doing things, your own way of naming things and as long as you're consistent with yourself then it won't matter. Someone once described it to me as learning a language, but with an accent - we can all understand each other but we say things in a slightly different way! It's like that with programming, don't be worried if your program looks different - if the results are the same it shouldn't matter too much.

  - *However*, that being said, you should always strive to write code that is *readable by another person*! You should always write comments explaining what complicated lines do and write your algorithms in a coherent way. This is not exclusive to your code being readable by someone else either - if you write a code and don't look at it for a few months, if it's not properly commented and explained then you will not have any idea what it does the next time you look at it! This has happened to me so many times that I am determined to not let you make that mistake.

- A *comment* in Matlab is achieved with the *%* sign:

```
>> 3 + 4; %this is a comment – it won't alter the result
```

- Using a semi-colon will *suppress* the output of the command - and the answer will not be printed to the command prompt.

Basic Coding Ideas

Variables

- You can assign a value to a *variable* in the following way

```
>> x = 3; %assigns the value 3 to the variable x
```

This will give the variable x the value 3, which can then be used in future calculations such as

```
>> x = 3; %assigns the value 3 to he variable x
>> 3 + x %uses the assigned value in the calculation 3+x
ans =

    6
```

These values will remain until they are overwritten or *cleared*. You can clear specific variables using the *clear* command.

```
>> x = 3;
>> 3 + x
ans =

    6
>> clear x;
```

Trying to use the variable x again will result in an error.

- Unlike many other programming languages such as VB, Fortran or C++, there is no need to *declare* variables to have a certain *type* in Matlab. Matlab

will automatically figure out whether you are specifying an integer, a real number or a complex number.

- Variables can be named any combination of letters and numbers both lower case and upper case but must start with a letter.

- You can see the variables which are currently stored by Matlab in the *workspace*. Alternatively, you can see the same information by running the command

```
>> whos
```

Loops

If there is nothing else you take from this document, let the thing you learn be how to use *for* loops and *if* statements. These are central to most programs you will write during this course and usually play an important role in many other programs.

The very simple example below illustrates what these statements do.

```
for j = 1:10
    j %display the value assigned to j at each point in the loop
end

for j = 1:10
    if j == 5
        j + 3 %if j = 5, display the value 8 − i.e. 5+3
    elseif j == 6
        j + 1 %if j = 6, display 7 − i.e. 6+1
    else
```

```
            j %if j is not equal to 5 or 6, display j
      end
end
```

Remarks on Loops

- You can use any variable you like as a loop counter name, although most people use i or j. However, care must be taken when doing this. Matlab recognises i and j to mean complex i unless they are user-defined first (i.e. at the start of a loop). Therefore, standard Matlab practise is to use $1i$ as complex i and allow you to change the value of i in a loop. If you want to input a complex number in Matlab, you should write it like this

```
>>> 2 + 1i
```

- Inside an if statement you can use several identities to check for conditions. The main ones are

  - $==$ (is equal to)

  - $\sim=$ (is not equal to)

  - $>= >$ (is greater than or equal to, is greater than, similar for less than)

- There is another type of loop called a *while* loop. These are similar to for loops but are particularly advantageous when you don't know how many iterations you need to end the loop. An example is below and one will need to be used in the tasks that follow.

```
j = 1; %initialsise the variable j to have value 1
while j <= 10 %loop until j reaches 10
    j %output j at each step
    j = j + 1; %add one to j and loop again
end
```

Task

Write a script that will loop through the numbers 1 to N and print out all numbers that are a multiple of 3.

You may find the following useful.

```
>> help mod
```

Task

Write a script that will print out the first N Fibonacci numbers.

Extension: Look on Google for ways to format your output so that it's quickly readable.

Write a script that will check the Collatz[a] conjecture for some number N.

[a]This is an open problem - no one has proved it for all $N \in \mathbb{N}$

The Collatz conjecture goes like this:

1. *Pick a number.*

2. *If the number is even, halve it.*

3. *If the number is odd, multiply it by 3 and add 1.*

4. *Conjecture: the sequence will always reach 1 in a finite number of steps.*

Plotting

- There are an overwhelming amount of possibilities that Matlab offers with regard to plotting functions - just describing it could be a course of its own. Therefore, in the interest of concision, I will only outline the absolute basic way of plotting here. You will pick up more points as you play around, but for now just see the following:

```
>> x = linspace(0,1,10); %generates grid of 10 points in [0,1]
>> y = sin(x); %evaluates sin(x) at each of the 10 points
>> plot(x,y) %plots
```

Plot the graph of $y = \cos(3x)$. Experiment with changing the value of the number of points plotted across. Plot the graph with circles at each point instead of a line using

```
>> plot(x,y,'o')
```

Vectors & Arrays

- Matlab stores vectors in a very intuitive way and there are several ways to create them.

```
>> u = [1 3 6]; %gives row vector containing 1, 3 and 6
>> v = 1:4; %gives row vector containing integers 1,2,3,4
>> w = 1:2:9; %gives row vector containing 1,3,5,7,9
>> for j = 1:4
        p(j) = j+1; %gives row vector containing 2,3,4,5
   end
```

- The elements of the vector can be accessed by the notation $v(j)$, where v is your variable name and j is the number of the index you want to access. Note that Matlab indexing *starts* at 1. It is never possible to access $v(0)$.

- Therefore, we can get the value 9 from w by typing

```
>> w(5)
ans = 9
```

12

- Column vectors can be defined as transposes of row vectors or with the notation

```
>> c = [1; 3; 5]; %note semi—colons. Row vectors are r = [1 3 5];
```

- The transpose operation is

```
>> c' %this will trasnform a row into column and vice—versa
```

Task

Use your previous Fibonacci program to plot the convergence of the ratio of successive terms to the golden ratio.

It can be shown that $\frac{F_{n+1}}{F_n} \to \frac{1+\sqrt{5}}{2} =: \varphi$ as $n \to \infty$. You program should plot both the ratio calculated and the value φ on the same axis.

Task

Use your previous Collatz program to plot the route of the sequence to its final value 1.

Task - Difficult

Write a script that will count and list the number of primes less than a given number N. This is called $\pi(x)$.

Extension: Compare your answer with the estimates $\pi(x) \approx n/log(n)$ and $\pi(x) \approx Li(x) := \int_0^n \frac{dt}{ln(t)}$. *Hint: Matlab has a built in function for Li called logint. Hint: To check that a number P is prime, you only need to check divisibility by numbers up to $\sqrt{P}$*

Matlab Specific Coding Ideas

Functions

- Until now, everything you have written has been in the form of a *script*. While that is useful, many programs will require the use of *functions*.

- A function contains parts of a code that need to be called in several places, to save you having to repeat code several times. The syntax and an example is below

```matlab
function c = myfunction(a,b)  %function inputs a,b − outputs c
    ans = a*b; %multiplies a and b − stores in c
end
```

- The function is then called from the command prompt as

```matlab
>> c = myfunction(3,2)
c =

     6
```

- This function should be written in a script and saved with the *same name as the function* in the current directory.

> **Task**
>
> Go back through your programs and change them into functions, with N as an input wherever possible.

Matrices

- One of Matlab's huge advantages over its competitors is the way it handles operations with matrices.

- There are some immensely powerful tools that make many matrix calculations simple in Matlab that are very difficult to implement in some other programming languages.

- For example, Matlab has inbuilt commands for matrix multiplication, inversion, calculating eigenvalues, solving systems of equations, creating diagonal matrices, determinants and many many more. Before you go any further, think of what it would take to write a program for any of these things. Could you write a code that calculates the eigenvalues of any given matrix?

- Matrices are formed in Matlab to a very similar way to vectors, and simply take two indices in an intuitive way

```
for i = 1:N
    for j = 1:N
        A(i,j) = %some condition on the entries of A
    end
end
```

- The elements of A are then accessed similarly to vectors. For example, $A(1,3)$ will give you the entry in the 1st row and 3rd column of A.

- Some standard commands are explained below. If there is a command which I've missed that you think should exist - chances are it probably does. Google is the answer!

```
>> det(A); %determinant
>> inv(A); %inverse
>> A'; %transpose
>> A*B; %multiplication (note #columns of A = #rows of B)
>> A.*B; %Hadamard (entrywise) multiplication
>> ones(10); %creates a 10x10 matrix of ones
>> zeros(3,4); %creates a 3x4 matrix of zeros
>> diag(1:4); %a 4x4 diagonal matrix with 1 to 4 on the diagonal
>> eye(3); %the 3x3 identity matrix
>> rand(3,2); %a 3x2 matrix of random numbers
```

Task - Quick

Write a script that will create an $N \times N$ identity matrix using for loops and if statements. Check against eye(N).

Systems of Linear Equations

- Matlab has a particularly powerful tool for solving systems of equations called *mldivide* or, more commonly, *backslash*.

- Given an matrix A and a right hand side vector b, the solution vector x which solves $Ax = b$ is calculated simply by

```
>> A = rand(10); b = rand(10,1);
>> x = A\b;
```

- Of course for this example the following would be equivalent

16

```
>> A = rand(10); b = rand(10,1);
>> x = inv(A)*b;
```

but this is considered bad practise as there is no need to calculate the inverse to solve the problem - leading to an increase in computational time. If you try to solve the problem in this way, Matlab will shout a warning.

- Backslash can also be used to solve over- and under- determined systems of equations in a similar way and uses least-squares fitting to do so. It is a very powerful tool which has made Matlab very popular for matrix operations.

- The command

```
>> help mldivide
```

will give a good explanation of the methods used and the best practice usage of this operator.

- Hopefully the following example will convince you to use backslash!

```
>> A = diag(rand(5000,1)); b = rand(5000,1);
>> tic, x = inv(A)*b; toc
Elapsed time is 30.292078 seconds.
>> tic, x = A\b; toc
Elapsed time is 0.040788 seconds.
```

Eigenvalues & Eigenvectors

- Another important feature of Matlab's matrix capabilities is its ability to calculate eigenvalues and eigenvectors of matrices.

- Given a matrix A, the eigenvalues and eigenvectors can be calculated simply

using

```
>> [V,D] = eig(A);
```

- This effectively solves the problem

$$Av = \lambda v$$

and stores the vectors v in V and the corresponding eigenvalues λ along the diagonals of the diagonal matrix D.

- Time-permitting, we may revisit this later in the course, but for now it is useful to just know that it exists.

Some More Advanced Ideas - with Applications in Applied Mathematics

- The following section will be organised into four/five distinct parts. We may not have time to cover them all, but they will be included for reference anyway.

  1. Numerical integration via the trapezium rule.

  2. Newton's method for root-finding.

  3. Solving linear ODEs with inbuilt functions.

  4. Simple finite difference solutions to ODEs.

  5. A brief introduction to spectral methods.

- Note that this course is NOT intended to be about the mathematical justification of the methods involved. We will likely be sloppy with assumptions

and not get too heavily involved in the analysis. Take me at my word for now that these methods are valid - the MAGIC course on Numerical Analysis as well as Prof. Phillips' MMath course on CFD will reinforce the ideas presented here if the interested student would like to know more.

Numerical Integration

- Matlab has a useful inbuilt function for calculating integrals via the trapezium rule called *trapz*. Example usage can be found via

```
>> help trapz
```

> **Task**
>
> Write a function which will integrate some input function y over a user-defined interval $[a, b]$. Your function should compare the results with the exact answer and make it simple to alter the function being integrated if required.
>
> -
>
> Experiment with the accuracy obtained by altering the number of grid points taken.

Newton's method

- Given an equation $f(x) = 0$, where f is differentiable, we can iterate towards the root x_r by means of Newton's method in the following way

$$x_{n+1} = x_n + \frac{f(x_n)}{f'(x_n)}$$

for some initial guess x_0. Provided we have a good initial guess, this will produce approximate solutions to any degree of accuracy required. In some

cases, Newton's method requires the guess x_0 to be *fairly close* to the true solution - something which is not always easy to do.

> **Task**
>
> Write a code that implements Newton's method for the following equations:
>
> - $f(x) = x^2 - 5$ with starting guess $x_0 = 2$.
>
> - $f(x) = sin(x) + x * cos(x)$ with starting guess $x_0 = 1.5$.

Solving linear ODEs

ode23 and ode45 plot and interpret

Finite Differences

Simple finite differences. Error analysis and the Mesh-Peclet number. Solve simple odes using finite differences. Look at TP - CFD.

Spectral Methods

Spectral methods and solving odes using Chebyshev methods - differentiation matrices and the like.

Curve fitting - spline and ppval.

Extra Ideas

Pathological examples.

Some software development techniques. Commenting, portability and readability.

Task: Write unseen program. Give to friend. Do they understand it?