Advanced Statistical Inference
Getting familiar with Matlab

1 Aims

- To see a working examples on how to use Matlab for:
  - Loading data.
  - Manipulating data.
  - Writing scripts.
  - Writing functions.
- To load and have a look at the Olympic data.

Remember - you can use any other programming language that you are familiar with - this sheet gives the instructions for Matlab in case you don't have any preferences on the programming language to use for the labs

2 Matlab

2.1 Introduction

Matlab is a flexible language for performing mathematical computations. It has toolboxes and functions for almost everything – always check before you start writing something!

Matlab is installed on the lab machines. Once you've found it and started it up, the first thing to do is to change your workspace. *This is where Matlab will save your files and look for functions and scripts that you have written. You can change this to something convenient by either clicking in the address bar at the top of the Matlab window or using unix-like* `cd` *commands. e.g.*

```
cd <my directory>
```

Two useful commands that are worth introducing are `help` *and* `save`. *Help allows us to quickly find out the syntax for any Matlab command. For example,*

```
help save
```

tells us all about the `save` *command. If you prefer your help to be in a more colourful form, try the following instead:*

```
doc save
```

As you can see, the `save` *command has more than one form. However, for the time being we'll just use the following:*

```
save <filename>
```

which saves all of the current variables into a file named `filename`. *Test it by typing the following commands:*

```
x = 3;
x
save testfile;
clear all;
x
load testfile;
x
```

The first line creates a variable called `x` *that is assigned the value 3 (note that in Matlab, we don't have to specify the types of variables - it does this for us). The second line, just outputs the value stored in* `x` *on the screen. In the fourth line, we save all current variables (i.e.* `x`*) into a file named* `testfile`. `clear all` *deletes all variables from memory and so, when we try and display the value in* `x` *again (line 5) we get an error because* `x` *no longer exists. Line 6 loads the file we have just created and hopefully, line 7 will now work and display the value of* `x`.

In this example, I've put semi-colons at the end of each line. This has the effect of suppressing any output that Matlab might like to produce. Experiment by putting the same (or similar) commands in with and without the semi-colon.

2.2 Scripts, functions and other things

In the example above, we typed commands directly into the command window. Commands can also be placed in scripts that can be run over and over again. To create a new script choose `file,new,m-file`. *This opens the Matlab editor. Commands are separated by newlines with or without semi-colons depending on whether or not you wish to suppress the output. Create a new script and insert the following commands;*

```
x = [0,pi/8,pi/4,3*pi/8,pi/2];
y = sin(x);
plot(x,y,'r-')
```

(Note that `pi` *is the Matlab name for π) Save it (it needs the extension .m e.g.* `aname.m`*) and then run it by typing the filename (without the extension) in the command window (you'll need to save it in whatever directory you set the workspace to). For example, if I saved it as* `testscript.m`, *I would run it by typing*

```
testscript
```

In this example, the variable `x` *is a vector (can be thought of as an array). The commas in the definition separate the individual values. Individual values can be accessed by, e.g.*

```
x(3)
```

or

```
x(i)
```

I have also used the `sin` *function (use* `help sin` *to find out more) and* `plot`. *Notice how the* `sin` *function is able to operate on a vector and will produce a result (*`y`*) that is also a vector. This is common in Matlab and it is always faster to perform operations on vectors and matrices than to write loops. For example, the command* `y = sin(x)` *is always faster than writing:*

```
for i = 1:length(x)
y(i) = sin(x(i));
end
```

Note the syntax of the `for` *loop (*`help for` *will provide more information). Notice also that before we start the loop we don't need to tell Matlab how big* `y` *needs to be. It will automatically make* `y` *bigger as necessary. This automatic memory allocation/management is handy but also slow. If we know how big* `y` *needs to be, it is always better to define it first. This can be done in any of the following three ways:*

```
y = zeros(1,5);
y = zeros(1,length(x));
y = zeros(size(x));
```

The first creates an array with one row and five columns. The second creates an array with 1 row and as many columns as there are elements in `x`*. The third creates an array of zeros which is the same size as* `x`*. As always, the* `help` *command will tell you more about the various commands used here.*

As well as scripts, we can write functions. For example, suppose we wish to write a function that plots a sine curve for us using equally spaced points between 0 *and* 2π *where the spacing is an argument passed to the function. Create a new m-file as above and insert the following:*

```
function f = sineplot(xspacing)
x = [0:xspacing:2*pi];
y = sin(x);
f = figure();
plot(x,y);
```

Save the file as `sineplot.m` *(it is important that the filename is the same as the name of the function. We can run the function by typing the following*

```
sineplot(0.1);
```

or

```
sineplot(pi/100)
```

etc. It should produce a nice sine curve which will get smoother as you change the argument passed to the function.

The first line tells Matlab that this .m file is a function definition. `f` *is a variable the the function will return. Matlab functions can return as many arguments as you like. If there is more than one, put them in a comma separated list inside square brackets, e.g.:*

```
function [a,b,c,d] = sillyfunction(x,y)
```

The second line is also new – we'll use ':' operation a lot. It allows us to create lists of equally spaced values. For example

```
x = [1:20];
```

produces a vector of the values from 1 to 20. In addition, we can add an extra argument that specifies the step size. For example, to make the list 1,5,9,13,17 we would use

```
x = [1:4:20]
```

The step size does not have to be an integer and doesn't have to be positive. Try some others.

3 Olympic data

In subsequent lectures, we will make heavy use of a dataset of Olympic data. We'll now open it up and have a look. Download the olympic matlab file from the ASI collaborative space (you'll need to save it in your workspace). Create a new script and add the following two lines

```
% Least squares Olympic script
clear all; close all; % Clear all variables and close all plots
load <olympicdata>
```

where `<olympicdata>` *should be replaced with whatever name you gave the downloaded file.*

Run the script and then type the command `whos` *in the command window. This lists all of the variables in the current workspace. You should see one (amongst others) called* `male100` *which holds the Olympic men's 100m data. Type* `male100` *in the command window so see the contents of this variable. It is a matrix (a 2-dimensional array). The first column holds the Olympic year and the second the winning time (in seconds). Matlab allows us to easily extract subsets of matrices by using the colon notation. For example, the command*

```
x = male100(:,1)
```

creates a variable called `x` *and copies into it the first column (a colon on its own just means 'all') of* `male100`*. Add the following two lines to your script:*

```
x = male100(:,1);
t = male100(:,2);
```

We'll now plot the data. Back in the command window, run the script and then try the following two commands:

```
plot(x,t);
plot(x,t,'b.');
```

The (optional) third argument specifies the plot format. Possible formats are listed in `help plot` *– experiment with them until you find something that looks good. Also experiment with adding the commands* `hold on` *and* `hold off` *between two plot commands. Add a suitable* `plot` *command to your script.*