Lab Report #0: Matlab Practice

Revised: September 16, 2014

This should be completed before the second class, but you can easily do it before the term starts. It will not be collected or graded, but you should do it anyway. Seriously. You'll regret it later if you don't.

Matlab is a popular high-level computer language, which means it's easier to use than C++ or Java and (far) more powerful than Excel. It's widely used in quantitative segments of the business world, including consulting, banking, and asset management.

What follows will get you started. It should take you about an hour.

1. *Accessing Matlab at NYU.* You can access Matlab in three ways at NYU:

   - Online with NYU id: vcl.nyu.edu
   - Online with Stern id: apps.stern.nyu.edu
   - On your own computer: You can buy a student version for $99 from the Mathworks. (This takes a while. Do it when you have a fast connection.)

   Choose one of the above. Once you do, start it up and type `demo` on the command line to get a demonstration of Matlab's basic features.

2. *Scalar operations.* We'll start by entering commands on the command line in what Matlab calls the Command Window. If you can't find it, look for "`>>`" (the "prompt").

   (a) Type each at the prompt:

   ```
   x=7
   x = 7
   x = 7;
   x
   x/2
   7^2
   ans
   ```

   What do each of these do? What does the semi-colon do? What if you skip it?

   (b) Now type these: `pi`, `exp(1)`, `x = exp(2)`, `log(x)`. What do they do?

3. *Vector operations.* Matlab treats everything as a vector or matrix. What this means is that you can do a bunch of things at once rather than copying the same command over and over again as you would in (say) Excel. Here's an example:

   (a) Generate a grid with the command: `x = [-3:0.5:3]`. What does `x` look like? (Type `x` at the prmpt to find out.)

   (b) Now compute x^2 for all values of x:

```
xsquared1 = x.^2
xsquared2 = x.*x
```

These two lines do the same thing, namely square each element of x. The dot or period here means do the operation element by element.

(c) Plot the results this way:

```
plot(x,xsquared1,'b')
hold on
plot(x,xsquared2,'mo')
```

What does the command `hold on` do? Type `help hold` to find out. Type `help plot` to learn how to control the type and color of lines. Can you make the line red? Magenta? Dashed? Can you replace the line with circles?

(d) Using the same `x`, plot the functions `normpdf(x)` and `normcdf(x)`. (You can enter these expressions directly in the plot command.) Use the help command to find out what these functions are — for example, `help normpdf`. What do you see? How would you make the line smoother?

4. *Vector operations (continued).* We can also work with data vectors. Suppose, for example, we have five observations of some variable `x` defined with the command `x = [1; 7; -2; 4; 8]`.

   Explain each of the following: `mean(x)`, `sum(x)`, `length(x)`, and `sum(x)/length(x)` As we noted earlier, you can get more information about these commands by typing `help command` at the Matlab prompt. You can also Google "matlab command".

5. *Symbolic math.* There are lots of computer programs that do symbolic math: solve equations, differentiate functions, and so on. We'll use Matlab to derive the moments of the normal distribution from its moment generating function. You'll use similar commands in future assignments, so pay attention.

   Type these commands in the command line:

```
syms mu sigma s
mgf = exp(mu*s + (sigma*s)^2/2)
mu1 = subs(diff(mgf,s,1),s,0)
mu2 = subs(diff(mgf,s,2),s,0)
mu4 = subs(diff(mgf,s,4),[mu sigma s],[0 1 0])
```

   What do they do? What happens if you skip the first line? What is `mgf`? What is `diff(mgf,s,2)`? How would you use `help` to find out more about `diff`? What does `subs` do?

6. *Data input and scripts.* Our last task is to input data from a spreadsheet using a "script": that is, we put the commands in a file so we can run them over again without typing everything in from scratch.

   (a) Enter the following data into a spreadsheet:

$x1$	$x2$
1	2
3	4
5	6

Save it as an Excel workbook with name (say) `testdata.xls`. Make sure you know what directory it's in.

Comments: (i) You can use any file name you like, but the same name must be used in the script below. (ii) You need to do something to tell Matlab what directory the file is in. How that works depends on what version you're using. (iii) **Warning for Mac users:** In Windows, all Excel formats are recognized, but Matlab for Macs may not recognize xlsx files.

(b) We're going to wite a Matlab script to read the contents of the spreadsheet into Matlab. Inside Matlab, go to the upper left corner, click on File, and choose New and Script. This should open the Matlab editor. Now enter these commands:

```
%  Practice script for data input from spreadsheet (this is a comment)
format compact
disp('Spreadsheet input')
data = xlsread('testdata.xls')
```

Save these commands under some appropriate file name in the same directory as the spreadsheet. It will automatically be given the file extension `m` (for Matlab).

Now run the program by clicking on the green triangle at the top of the editor. (An alternative is to type the file name, without the `.m`, on the command line.) Then go to the Command Window to see if it works. If not, welcome to the world of programming, where most of your time is spent fixing bugs. Remind yourself that patience is a virtue. And speak to others about your problem, that often helps even if they don't have the answer.

(c) We now have the spreadsheet contents in `data`. To extract the columns, type

```
x1 = data(:,1)
x2 = data(:,2)
```

The notation `data(i,j)` refers to the entry in row i and column j of `data`. It's analogous to what you would see in a spreadsheet, where the rows are labeled by numbers and the columns by letters. A colon ":" means take all the elements of this type. So `data(:,1)` says take every row of column 1; in other words, the whole column. For more about this, see Matlab's online help.

(d) Add to your program a line that plots x1 against x2 and marks each data point with a circle.

7. *Functions.* Matlab allows you to write functions in a couple ways.

(a) The most common way is to store a function in a separate file. (I view this as a design flaw of Matlab, but that's the way it is.) If you put a function `helloworld` in the file `helloworld.m`, then any time you type `helloworld` it will execute that function. (Assuming Matlab finds it; for now, we'll hope for the best.) That's what many of the commands we've used so far do: they call the Matlab file of the same name.

Here's an example:

```
function helloworld(x)
% Comments
```

```
% If you type "help helloworld" you'll get these comments
% The program prints "Hello World!" and the value of the variable x
% Examples:
%   helloworld(7)
%   helloworld([1:7])
disp('Hello World!')        % displays text in quotes
x                           % print x
return                      % end program and return
```

Enter these commands in the file `helloworld.m` and save them. Now type at the prompt:

```
help helloworld
helloworld(pi)
```

(b) A more compact way to write a short function is with an "anonymous function." The terminology is obscure, but here's how it works. Type the commands:

```
myfunction = @(x) disp(x^2);        % the function
myfunction(7)                       % executing the function
```

You should get 49. We'll use something like this when we get to option pricing, where we need to use a long formula repeatedly. More on anonymous functions at the Mathworks.