Chapter 1: Matlab Basics

1.1 Entering data into memory with Matlab

```
>>x=3 <enter>
x =
    3
>>
```

A ; (semicolon) in matlab is the same thing as pressing <enter> inside a program; its use is necessary as when you execute a program you are not expected to hit <enter> a bunch of times (we will cover more of this later). When you use ; in the consul, it doesn't do much:

```
>>x=3;
>>
```

It simply causes the program not to repeat what your assignment is like in the first example. If you forget what x is equal to later on, do this:

```
>>x <enter>
x =
    3
>>
```

Vectors: Now we will make x a vector as follows.

```
>>x(2)=4
x =
    3    4
>>
```

Or try this:

```
>>x(2)=4;
>>
```

To see what x is now equal to:

```
>x
x =
    3    4
>>x(1)
ans =
    3
>>x(2)
ans =
    4
>>
```

Now let's take a line of data from a .txt file, you should highlight the data (skip the header Wavelength Absorption) and copy it (ctrl c), here is just an illustration:

Wavelength Absorbance (***SEE Dataset1_1.txt***)
500 0.1
501 0.2
502 0.3
503 0.2
504 0.1
<ctrl c>

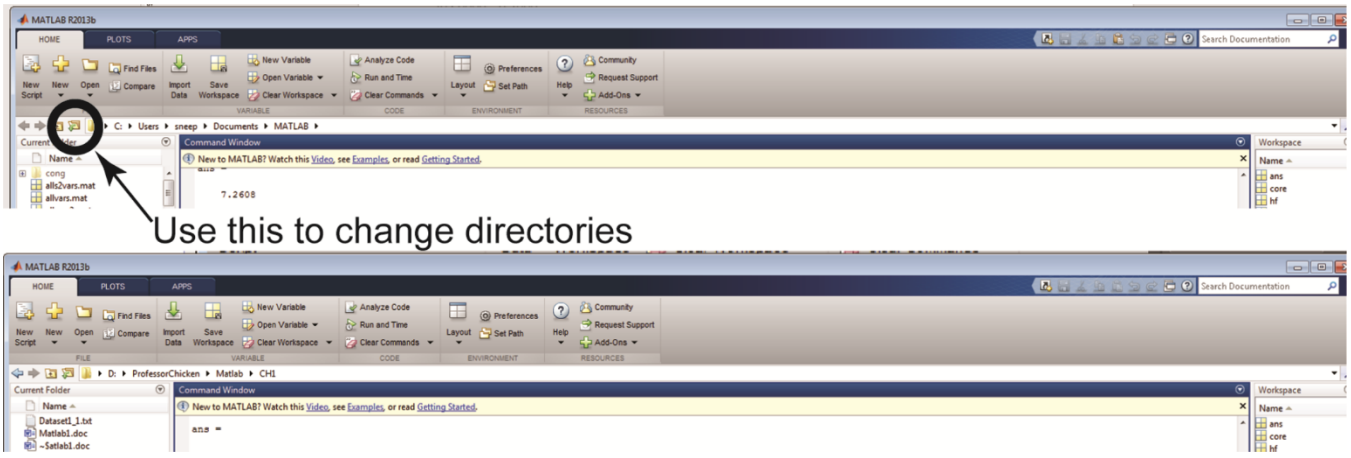
Now lets go back to the matlab window and type the following:

>>spectra=[(now hit <ctrl v> and this becomes)
>>spectra=[500 0.1
501 0.2
502 0.3
503 0.2
504 0.1
(now type in] and hit enter, making it look like this):

>>spectra=[500 0.1
501 0.2
502 0.3
503 0.2
504 0.1
] <enter>

spectra =
500.0000 0.1000
501.0000 0.2000
502.0000 0.3000
503.0000 0.2000
504.0000 0.1000
>>

Congratulations! You have now loaded in a spectrum from your data. Alternatively, you can change the directory that Matlab is in as shown here:



Move to the one that holds Dataset1_1.txt and type in the following:

```
>> spectra=load('Dataset1_1.txt');
```

Regardless of how you load the data, the x-axis can is now the first column of the Matlab variable spectra:

```
>>spectra(:,1)
ans =
  500
  501
  502
  503
  504
```

And the y-axis is

```
>>spectra(:,2)
ans =
   0.1000
   0.2000
   0.3000
   0.2000
   0.1000
```

Notice what : does? In the above examples, it means “show me all the numbers in this column.” In short, : means “everything”. Here is another use:

```
>>spectra(:,2)=spectra(:,2)-0.1
spectra =
  500.0000        0
  501.0000    0.1000
  502.0000    0.2000
  503.0000    0.1000
  504.0000        0
>>
```

See! You just baseline corrected the data! Here is another example:

```
>>spectra(:,2)=spectra(:,2)*10
spectra =
  500.0000        0
  501.0000    1.0000
  502.0000    2.0000
  503.0000    1.0000
  504.0000        0
```

Great if you want to scale your data for visual purposes. Here is another example:

```
>>spectra(:,2)=1
```

```
spectra =
   500     1
   501     1
   502     1
   503     1
   504     1
```

Notice you just make a mass-assignment of the second column in spectra. Useful, but you can wipe out all your work fairly easily. Now cut and paste the original spectra data from above back into Matlab:

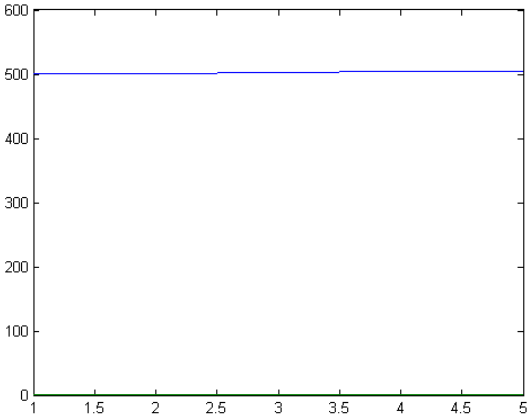
```
>>spectra=[500          0.1
501       0.2
502       0.3
503       0.2
504       0.1
] <enter>
```

1.2. Figures

Let's make a figure!

```
>>plot(spectra)
```

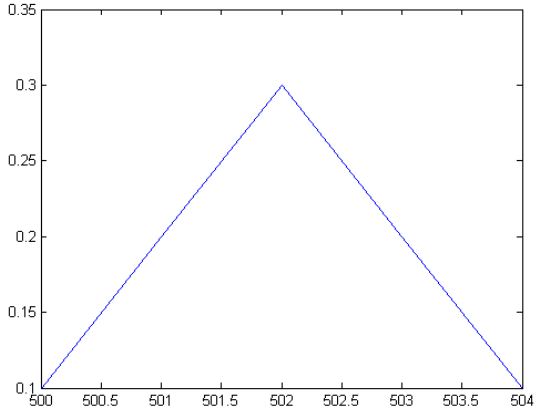
and you get this:



The reason this looks like cat vomit is because matlab doesn't know what you mean to use for the x-axis and the y-axis. The format for the plot command is plot(x-axis,y-axis) so type in:

```
>>plot(spectra(:,1),spectra(:,2))
```

and you get:



Try these commands in this order, and watch what happens after you type each one in:

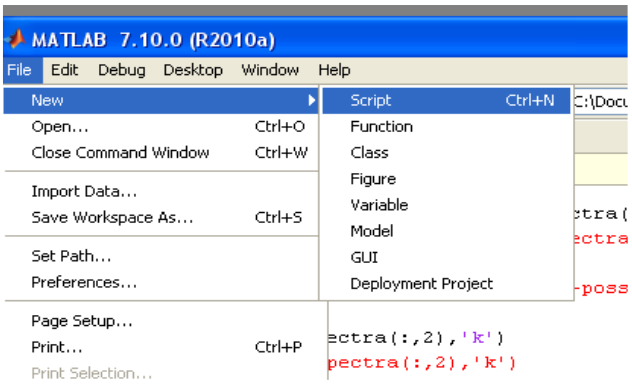
```
>> plot(spectra(:,1),spectra(:,2), 'ro')
>> hold on
>> plot(spectra(:,1),spectra(:,2),'g')
>> hold off
>> plot(spectra(:,1),spectra(:,2),'k')
>> xlabel('Wavelength')
>> ylabel('Emission')
```

Already better than Excel, isn't it! To save this kind of figure for your report, type the following:

```
>>print -djpeg figure1.jpg
```

1.3. Now let's make a script

A script is a series of commands stored in a file which will be executed all at once. This is much better for when you have to make a series of analysis, but if you screw up, you don't have to retype anything. It also makes it easier to spot an error before you have executed a command.



To make a script, go to file, and select new M file. A window like notepad will open. In the is window, type the following commands (i.e. don't be an idiot- cut and paste the following):

```
clear all;
close all;
spectra=[500    0.1
501     0.2
502     0.3
503     0.2
504     0.1];
spectra(:,2)=spectra(:,2)-0.1;
plot(spectra(:,1),spectra(:,2));
hold on;
plot(spectra(:,1),spectra(:,2),'ko');
```

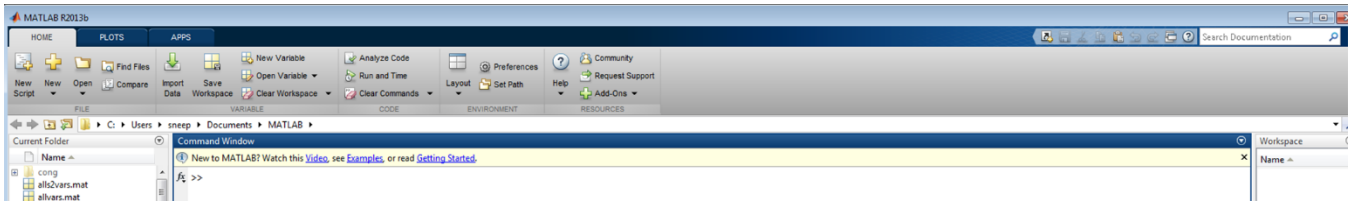
Note the use of the semicolon on every line!!!! Go to file and save. You have to make up a name, I will use snarf. Now, go back to the Matlab consule and type:

```
>>snarf  <enter>
```

And what happens? The data are created, baseline corrected, and a figure is plotted with the data! All at once, so you can see, if you make some kind of mistake you can just easily edit the .M file without having to type everything all over again.

1.4 Loading data

The cut and paste thing is the easiest way to load data, but what if you have a lot of data and that's just impractical? Matlab can load data directly, but there are just two issues you have to address. First, notice that the status bar ("Current Folder:") at the top of the matlab window had your present working directory:



Generally, it is something like: C:\Documents and Settings\My Documents\MATLAB. Problem- the data file has to be in the same directory, or you have to move the present working directory to the one with the data. To move directories, hit the […] button next to "Current Folder:" which will allow you to move into another directory. Next issue, your data must be a table of numbers with no other extraneous information. For example, let's look at the data set (Dataset1_2.txt) that is provided in this packet:

```
Wavelength        Absorbance
  5.0000000e+02   2.6837674e-02
  5.0100000e+02  -1.9768694e-02
  5.0200000e+02   3.6358854e-02
…
```

Now type this to load it in (and note the response after):

```
>>mydata1=load('Dataset1_2.txt');
Error using load
Unknown text on line number 1 of ASCII file Dataset1_2.txt
"Wavelength".
```

Whups! Matlab doesn't know what to do about the text in the first line, and how that meshes with the numbers that come after. Here is what you do- you cut out the first line in notepad, and then save it into your present matlab working directory. With that part out of the way, do the following in the consul or in your script:

```
>>mydata1=load('Dataset1_2.txt');
```

Did it work? Hard to tell because matlab hasn't acknowledged anything yet. There are two ways to know: first, do what I already showed you, type in the variable name and hit enter:

```
>>mydata1 <enter>
mydata1 =
  5.0000000e+02   2.6837674e-02
  5.0100000e+02  -1.9768694e-02
  5.0200000e+02   3.6358854e-02
  …
>>
```

Yipee! Here is another way: type who

```
>>who <enter>
Your variables are:
ans     mydata1   spectra
```

It lists all the variables you presently have saved to memory. You can use this at anytime to remember what you are working with and what you are not.

1.5 Programming commands

Let's write our first simple program on the consul. This is a for loop which will repeat a command several times (here, 5 times). The variable i is equal to 1 on the first loop and 5 on the last. Let's use this:

```
>>for i=1:5 kitty(i)=i/10; end; <enter>
>>
```

Again, you don't know what matlab did. Ask it!

```
>>kitty <enter>
kitty =
   0.1000    0.2000    0.3000    0.4000    0.5000
>>
```

Now lets say you meant to create a vector from 0 to 0.4. There are two ways to fix it:

```
>>kitty=kitty-0.1;
```

Here is another: hit the up arrow until the original program appears:

```
>>for i=1:5 kitty(i)=i/10; end;
```

Now move the cursor left and alter the command as so:

```
>>for i=1:5 kitty(i)=(i-1)/10; end; <enter>
>>kitty
kitty =
       0    0.1000    0.2000    0.3000    0.4000
>>
```

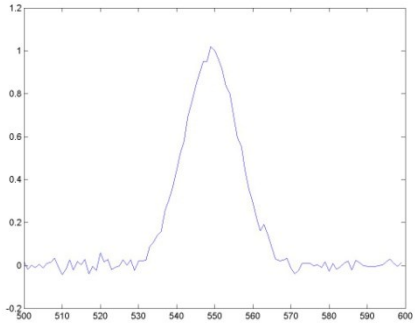
See? You can redefine the variable i within the loop to suit your needs. Note, you cannot have fixed this problem by using:

```
>>for i=0:4
```

Just try it! Now you can imagine that this can get complicated, which is why you normally put programming instructions in an M file because your going to make a lot of mistakes when your typing this stuff in the consul (and you will screw up the data often).

Here is the best part- the for loop is pretty much the only programming function you need to use!

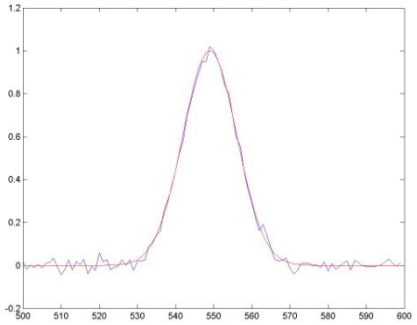
1.6 Error Analysis with Matlab; calculating R^2



Let's look at the set of data we just loaded in (it's an emission spectrum) and fit it. While we can do a good job doing that, I want to know how good with numbers. In this case, we want to calculate the "goodness of the fit", aka R^2. Let's use matlab to calculate this quantity on the data set that you have already loaded (mydata1). First, plot mydata1 so that you have an idea what your data look like:

```
>>plot(mydata1(:,1),mydata1(:,2))
```

Looks like some sort of emission spectrum, fairly Gaussian (bell-shaped). Now our job is to calculate R^2 of our fit to this data. First, I will define my best fit to the data:



```
>> for i=1:100 fit(i)=exp(-(i-50)^2/100); end; <enter>
```

Where did I get the parameters for this fit (50, 100)? We will be doing that part in the next rotation. For now, let's see if I did a good job as evident from a high R^2 value:

```
>> hold on
>> plot(mydata1(:,1),fit,'r') <enter>
```

Looks good! But how good? R^2 is calculated from the sum of the squares of the differences of the data from the fit: $\sum(\text{data} - \text{fit})^2$. This result is divided by the sum of the square of the data points minus the average value of all those same data points: $\sum(\text{data} - \text{mean}(\text{data}))^2$. Finally, R^2 is calculated as:

$$R^2 = 1 - \frac{\sum(\text{data} - \text{fit})^2}{\sum(\text{data} - \text{mean}(\text{data}))^2}$$

I will show you how to translate this into Matlab-ese below.

I first note that I need to do a summation. A for loop can be used for this purpose. Here is an example:

```
>> part1=0;
>> for i=1:100 part1=part1+(mydata1(i,2)-fit(i))^2; end;
```

This calculates the numerator. As for the denominator:

```
>>part2=0;
>> meandata=mean(mydata1(:,2));
>> for i=1:100 part2=part2+(mydata1(i,2)-meandata)^2; end;
```

See your done!

```
>>r2=1-part1/part2
r2 =
   0.9957
```
And there is your answer! An R^2 of .99 or greater represents a very good fit.

1.7 Conclusion: Other useful commands

Now we have done some processing of the variable spectra (we baseline corrected it), you don't want to have to this over and over again. To save the data, close matlab and come back to it later, just type the following:

>>save mydata1 mydata1 <enter>

Now you can load it in later. DO NOT FORGET TO TYPE THE VARIABLE NAME TWICE! It's a long story, but that can cause some major problems later. If you do it accidentally, just retype

>> save mydata1 mydata1 <enter>

again, and it will be fixed. More on what is happening with this later on..

ls or dir -lists all files in your present working directory

```
>>ls
.          Thumbs.db     snarf.m       figure1.jpg   mydata1.txt
..         temp.txt          mydata1.mat
>>
```

Don't forget we have a .M file present when we wrote the "snarf.m" script and we have a .MAT file present as just saved the variable spectra. Thus, scripts are labeled .m and variables .mat.

cd .. -change directory one level up

cd (name) -move into the directory named name.

pi -π, or 3.1416

sin(pi) - the sine of pi. Anything can be given to sin(?), let's try this:

>>sin(mydata1)

```
ans =
  -0.4678    0.0268
  -0.9965   -0.0198
  -0.6090    0.0364
   …
```

See? The sine of every element of mydata1. Other trig functions are cos, tan, etc. See, its all fairly intuitive.

abs -absolute value

exit -quit the program