Matlab 2

This week we are going to use matlab to do some more adventurous programming. We are going to fit some data with an exponential fit first we need to define a model which contains our fit.

Y=A+B*exp(-X/C)

In the file menu create a new function and type the following

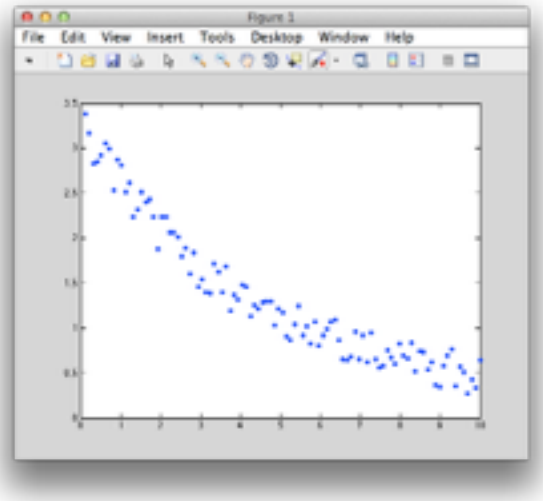
```
function F = expon(c,xdata)

F = c(1)+(c(2)*exp(-1*xdata/c(3)));
```

The first line defines a function which can take two arguments c and xdata. xdata will be the x data we are going to send to the function. c is a matrix with three numbers in it, they are c(1) the offset of the fit, c(2) is the amplitude of the exponential and c(3) is the decay constant of the exponential.

Save this file as **expon**. Now we can use this file to perform a fit on some data we have. First we need to make the data. Type this is to the command window.

X=linspace(0,10,100); This gives us 100 points from 0-10 stored in X
random=rand(1,100); This gives us 100 random points stored in random
Y=(random/2)+3*exp(-1*(X/4)); This gives us our noisy Y data.

Note that in the above data we have just created the offset as a noisy data set, the amplitude is 3 and the decay constant is 4.



Plot these using **plot(X,Y,'.');**

Ok lets try fitting this data.
To fit this data we are going to use the command **lsqcurvefit** this is a nonlinear curve fitter using a least-squares method. The syntax of this command is

x = lsqcurvefit(fun,x0,xdata,ydata)

Lets break this down first we can see that when we call the **lsqcurvefit** function we have to supply it with four pieces of information. The first one fun is the model we want it to fit too, in this case we are going to use the model we made at the start of the exercise. This will be **@expon**, which is just the function we saved earlier, the @ symbol tells matlab we are passing it a function (an operation) rather than data. The second piece of information we have to send is X0, these are our initial guesses to the problem and they also tell matlab how many parameters it has to play with, we know that when we build the model we gave it three parameters c(1) the offset, c(2) the amplitude and c(3) decay constant. We therefore need to give matlab a command to set our initial guesses, something like **x0=([1 1 1]).**

So this is the set of commands we would type in.

```
>> X=linspace(0,10,100);                    $This sets up our X data
>> random=rand(1,100);                      $This gives us some random data
>> Y=(random/2)+3*exp(-1*(X/4));            $This uses the above data to make
                                            $an exponential.
>> plot(X,Y,'.');                           $We plot that
>> x0=([1 1 1]);                            $We set up our initial guesses
>> xout = lsqcurvefit(@expon,x0,X,Y);       $We call expon to fit our data


>> Yfit=xout(1)+xout(2)*exp(-1*(X/(xout(3))));   $Now we can generate our fit data
>> plot(X,Yfit,'r',X,Y,'g.');                    $Plots our fit (Yfit) and our data (Y)
```

Before you type all this in you could type it in to a new script, to do this go to the file menu and select New->Script. Then type the commands (you should leave out the first plot command), in to the new window and save the resulting file. You can use any name but don't have any spaces in the file name.
If you saved the file as, for example, Fitting. Type Fitting in to the command window and it will run the script you have just written.

Here is what mine looks like,

```
% setting up some variable
x0=([1 1 1]);                               %My first guess for the fitting parameters
start=0;
stop=20;                                    %The length over which we will make our fit
scale=.1;                                   %How noisy our data will be
amp=40;                                     %the functions amplitude
dec=5;                                      %the functions decay rate

X=linspace(start,stop,100);
random=rand(1,100);
Y=(random/scale)+amp*exp(-1*(X/dec));
xout = lsqcurvefit(@expon,x0,X,Y);
Yfit=xout(1)+xout(2)*exp(-1*(X/(xout(3))));
plot(X,Yfit,'r',X,Y,'g.');
```

In Class Task
Download the gaussian data file from the website ashleycadby.staff.shef.ac.uk and fit it to a Gaussian. The equation for a Gaussian is;

Y=offset+(amplitude).*(exp(-1*((centre-X)^2)/width^2));

File handling

One of the most important operations that you will need to do in mat lab it to load data from experimental apparatus and process it. This data will normally be in one of a number of formats. In this section we shall learn how to load, comma separated values, tab delimitated values and tiff image files.

First download the CSV data file from the website
http://ashleycadby.staff.shef.ac.uk/Matlab/matlab.html
You may have to right click and use the save as function.

Place the file in the folder you have selected as your file store for matlab. The file should be called csvdata, but you may rename it to any name you want. To load this file in to matlab we can use the **csvread** command. The format of this command is "Matrix = csvread(filename);". So for the file we have just downloaded it would be,

M = csvread('csvdata');

If we use the **size** command then we can see how much data we have loaded in to memory,
size(M)
returns
ans =

2977 3
So we have three columns each with 2977 rows in them.
To look at the first 2 rows of data we can plot them using the
plot(M(:,1),M(:,2),'.');
command.
This data was collected using a super resolution microscope and each point in the data set corresponds to the position of an anti-biotic molecule attacking a bacterial cell. The bacteria in this case are responsible for MRSA.
Try plotting the data in 3D using the command
plot3(M(:,1),M(:,2),M(:,3),'.');
Here you can use the rotation icon at the top of the image place to rotate the data set.

You can see that the data is a little unclean, lets have a play at cleaning it up. The first thing about this data is that it has a higher resolution that the microscope can actually read, we can use a 3D histogram to map it to a sensible resolution. A 3D histogram collect the data we have and puts it in to a set of pixels with each pixel having its own value.

It will make more sense when we do it, because we have a lot of typing we should use a script to do this. Before we do that lets clear the data and reload the bacteria data.
clear
M = csvread('csvdata.txt');

```
subplot(2,2,1);
plot(M(:,1),M(:,2),'b.');

subplot(2,2,2);
plot3(M(:,1),M(:,2),M(:,3),'r.');


MX=[M(:,1),M(:,2)];
hissy = hist3(MX,[50,50]);
subplot(2,2,3);
colormap(hot);
imagesc(hissy);

subplot(2,2,4);
surf(hissy);
```

The script above can be broken down in to several little sections, as we have some new commands we will go through them one by one.

```
%this is a fun little script
X=linspace(0,2*pi(),100);

for i=1:25
    subplot(5,5,i);
    plot(sin(i*X));
end
```

The first new command is
subplot(m,n,p);
This allows us to put multiple plots on a single graph, **m** and **n** tell matlab how many figures you want across and high, the **p** tells mat lab which plot the current one will be.

We then have a 2D plot in plot (1) and a 3D plot in plot (2).

Now it gets exciting we are now going to plot the 2D plot as a set of pixels, using the 3D histogram function **hist3** this command needs us to supply X and Y coordinates so we have to make a new matrix without the Z data.
The command
MX=[M(:,1),M(:,2)];
Take the first two columns and makes an new matrix from them called MX.
The hist3 function we use is
hissy = hist3(MX,[50,50]);
This stores the output of the function (which is a matrix) in hissy. The command its self takes two inputs the first is the matrix of X,Y points which we have just made called **MX.** The second is a matrix telling **hist3** how many pixels to use, in this case we have used 50 by 50 **[50 50].** We could have used any numbers.
The commands
colormap(hot);
imagesc(hissy);

Are used to change the colour of the matrix we plot and then plot it as an image.
The command **imagesc,** scales the image intensity when its plotted. We could also use the command **image** which would not do that. You can try them both.

Finally for a last plot, we plot **hissy** as a 3D plot using the surf function.

Lets get back to loading and saving data, we previously used the command
M = csvread('csvdata');

```
X=linspace(0,100);              %Sets up our X data


for i= 1:10                     % For loop from for each file
C=[1 1 (i*5)];                  % define the constans for our exp (changes one)
Y=expon(C,X);                   %build the exponential and puts it in Y
Y=Y';                           % These two line swap our data from
X=X';                           % rows to columns
data=cat(1,X,Y);                % Combines our data in to one array rather
                                % than two
filename=strcat('bunnykitten',int2str(i));  %builds the file name
csvwrite(filename,data);        % saves the data

end
```

To read data in to an array **M** the file name is 'csvdata' and this is a string, i.e it is not a set of number but a bunch of letters strung together, 'a string' of letters. If we want to load or save a number of data sets we don't want to have to type in the name of every file. So we are going to create some numbered file names now.
Again there are several new command here so we will go through each line.
X=linspace(0,100);
The above line sets up an array X with the number 0 ->100.
C=[1 1 (i*5)]
Y=expon(C,X);
The next line sets up the variable we are going to pass to the exponential function we created earlier. If you remember the function takes two inputs, the X data and the constants needed to create an exponential from the function given here, we can see that C(1) is the offset of the function C(2) is the amplitude of the function and C(3) is related to the decay constant. By using the command **C =[1 1 i*5];** we are keeping the offset and the amplitude constant and altering the decay constant each time the loop is executed. The line **Y=expon(C,X);** calls our function and puts the resulting data in Y.

```
function F = expon(c,xdata)

F = c(1)+(c(2)*exp(-1*xdata/c(3)));
```

data=[X Y];
filename=strcat('bunnykitten',int2str(i));
The next line makes a new array called data by combining the two arrays X and Y.
The line **filename=strcat('bunnykitten',int2str(i));** is a little complicated, it has two functions in it the first **strcat** joins two strings together and stores them as the string filename. The two strings it joins together are held with in the brackets (X,Y). Where X in this case is a constant 'bunnykitten' and Y is a function that returns a string. We could

have used two constants here if we had wanted to. However, we used the function **in2str** this function take an integer and turns it into a string. So for example if we have the integer '123' it would return '123', sounds stupid doesn't it. However, the first case is a number and the second is as we had written "one two three'. One is a number (as the computer sees it) and one is the computers representation of the numbers in human format. So the **int2str(i)** function return a string each time it is called and the string has the written value of (i) so it goes from one to ten.
This means that the
csvwrite(filename,data); command writes the array data to the filename called filename which will be something like bunnykitten1 or bunnykitten2.

In Class Task
Write a script to load the 10 files you have just saved. Load each data set in to memory, then store the Y data sets in a single array (100,10) there are 100 points in each curve and 10 curves. The plot on the same graph all ten curves.

Reading other types of data is just as simple using the
tdfread(filename,delimiter)
command, this command will read in any data set with any delimiter you specify, the default option is tab so if you loading a tab delimitated file in you can ignore the delimiter. However, you can use any delimiter you want, examples are

' ' or 'space'
'\t' or 'tab'
',' or 'comma'
';' or 'semi'
'|' or 'bar'

```
while evalResponse > 0
      evalResponse = input('Can I have a number please ');
            Y=magic(evalResponse);
      disp(magic(evalResponse));
end
```

User input
It is often useful to have a user make decisions while running a script, look at the script below;

The above code takes a user input and produces a magic square with a width given by the user. If you type in zero i.e '0'. the while condition will not be met and the program will end.

Guess a number (class problem)

```
prompt='Give me a name ';

strResponse = input(prompt, 's');
len = length(strResponse);

for i= len:-1:1
    display(strResponse(i))
end
```

The function **random('unid',10,1)**, will return a random integer between 1 and 10. Use this to see if you can guess the number the computer generates. Keep a count of the times you get it correct and the times you get it wrong. As the random command does not generate a zero you can use the users input of '0' to terminate the loop.
If you wanted to use strings rather than numbers you can alter the command to
response = input(prompt, 's'),
The 's' tells matlab to expect a string. here is a simple example.
Type this is code and work out what it does. Can you alter is so that it counts how many 'a' some one has in their name?

Images

Copy the single Tiff file from the website http://ashleycadby.staff.shef.ac.uk/Matlab/matlab.html to your file store space in matlab.
then type in to the command window.
filename='reference00005101.tiff';
The above command defines a string variable called filename with the path to our image, the next command loads the data in to the matrix Z
Z = double(importdata(char(filepath)));

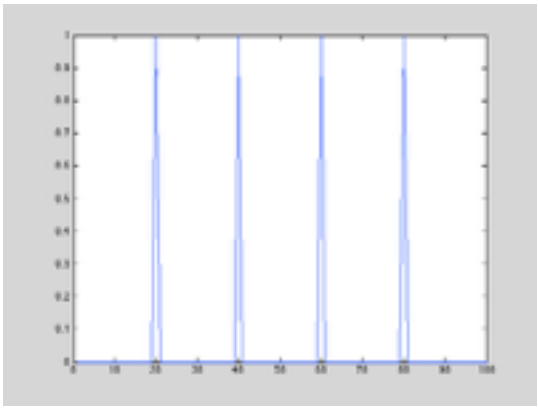
Lets have a look at the image we have just loaded type
Z
You can see that the data is stored as a matrix of numbers, this is not very helpful if we want to "see" the data. To view a matrix as an image type the following command. imagesc(Z) or image(Z), you can see that with that using the image(Z) command and the imagesc(Z) give very different results, this is because the imagesc(Z) command scales the display to make the data presentable, where as the image(Z) just presents the raw data. The data you have just loaded is a microscope image of single molecules.

The next few examples concentrate on some basic image analysis, there are here simply as an example of how easy it is to use matlab to perform some very complex actions. If you ever need to analyze data in an involved manner it is likely that matlab can do it in a simple way. The help menu will solve many problems. Anyway lets look at images.

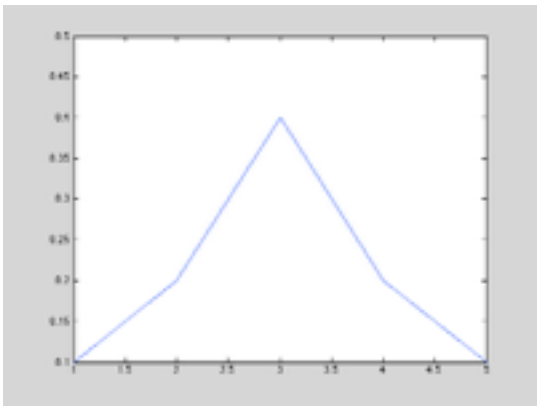
The first thing I want to do is to look at a histogram of my data, this tells me the distribution of noise to signal. If I type (try it)
hist(Z);
Matlab will try to give me a histogram for each column, which is not useful. Instead we need to change Z from a matrix in to a 1D array to do this we can type.
p=Z';

W=p(:);
The size command shows us that we have converted the 2D matrix to a 1D array
>> size(Z)
ans =
256 256
>> size(W)
ans =
65536 1
Now the command
hist(W,100); note the 100 gives me 100 bins, shows me the distribution of my data.

Filtering methods.
In the next few pages we are going to look at some custom filters, before we start lets look at one of the many filter function in matlab. The function we are going to look at first in the filter function and the syntax is given below.
dataout=filter(b,a,datain);
The function filter takes three inputs 'datain' is the data you want to filter**.** 'b' is a numerical coefficient vector, this is used to tell matlab over how many data points to smooth the data and the weighting of the data points. 'a' is a scaling factor for the smoothing.
Lets have a play, first we need to make some data which will be smoothed, we shall do this using the zeros function,
Y=zeros(1,100);
Y([20 40 60 80])=1;
plot(Y);
The code above has taken a matrix filled with zeros and added to it four spikes, one at 20 one at 40 one at 60 and one at 80. Plot it using;
plot(Y);



You should see something that looks like the plot to the right. We are going to smooth this data. The first this we need to do is build out filter inputs, for b i am going to use **b=([0.1 0.2 0.4 0.2 0.1])** and for a i'm using **a=1**. Type in the values for a and b then type **plot(b)**, this will show us the shape of the filter we are applying, it should look like the figure to the right. This means at each point (i) in your data, the new value =(0.1) x (i-2) + (0.2) x (i-1) + (0.4) x (i) + (0.2) x (i+1) + (0.1) x (i+2). Where (i) is the position in the array of the data point being modified.

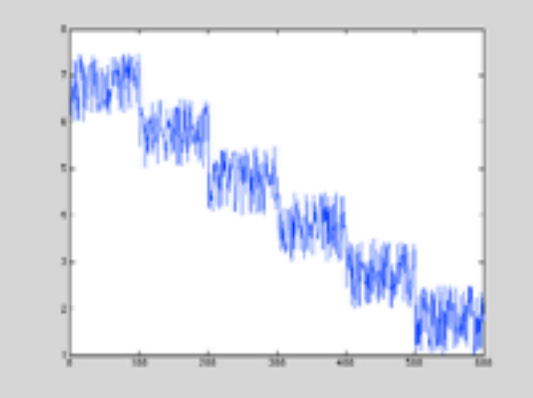


So to use this we would type;
newy=filter(b,a,Y);
To look at the filtered data type plot(newy); If you filter the already flittered data over and aver again you will see one of the major problems with using filters like this and that is the shift in the data, try it with,
newy=filter(b,a,newy);
plot(newy);

```
s=10000;                          %This is the number of points our data will have
plo=5;                          %This is the number of loops (it can be any number squared
plo=plo*plo;                        % this squares plo
X=linspace(0,2*pi(),s);             %sets up our x data
random=rand(1,s);                   %gives us some random noise
Y=1*random+sin(6*X);                % gives us some noisy data try changing to 1 to a 6
st=zeros(s,plo);                                %somewhere to store out filtered data
count=1;                          %a counter so we know which column to store our data in
for q=3:2:(plo*2)                 %first for loop which loops plo times in odd numbers
                                   and picks each data point in an array
        for i=q+1:s-q             %second loop goes sums up the data either side out to a
                                       width of plo


        temp=0;
            for m=1:2*q

                 temp=temp+(Y(i-q+m))/q;  %the summation is stored in temp and
                                         normalized by the number of cells summed


            end

        st(i,count)=temp;                          % the data is stored in the matrix st

        end
        count=count+1;
end

subplot(sqrt(plo),sqrt(plo),1)    %this just plots the data in a square subplot matrix
plot(X,Y);

for j=1:plo-1
    subplot(sqrt(plo),sqrt(plo),j+1);
    plot(X,st(:,j));
 title(2*j+1);
end
```

Lets look at another filter method, a simple smoothing filter as given above,
This filter is written as a script to show the power of smoothing filters, type the following in to a new script window and save it.

The code above makes a noisy Sine wave and smooths the data using an odd number of points to sum over starting at 3 and finishing at what ever the users picks its default is 49. The effect of this increased smoothing is shown in the plots, try changing the number of points in the first line of the code to 100.



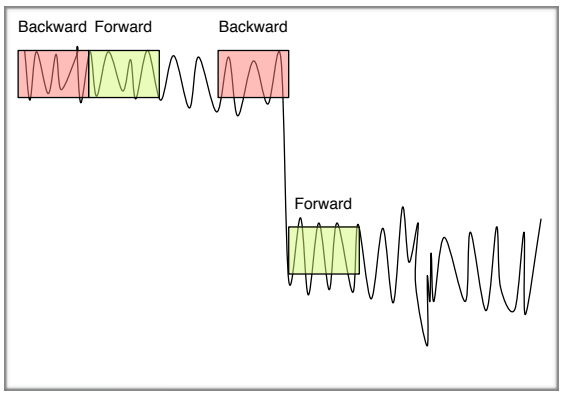
We are going to now build a conditional filter, this filter will use a condition to decide which value to select. The first thing we need to to is to get some data, I

have made a data set up and put it on the website
http://ashleycadby.staff.shef.ac.uk/Matlab/matlab.html
The data file is a matlab matrix and the link is called the 'Chung Kennedy filter data', if we plot the data we get the figure to the right. The data is saved as a matrix called out.

```
%new chung kennedy

filtered=filter([0.2 0.4 0.2], 1, data);          %Smooth the data

[len temp] = size(data);                          % calculate data length

allo=2;                                           % this is the tollerance
                                                  % which we use in the
                                                  % condition
st=zeros(1,len-given);                            % build a place to store
 temp=zeros(1,len-given);                         % the data

for i=given:len-given                             % our for loop
                                                  %
                                                  % these next variable
                                                  % store data
    forward=0;                                    %
    backward=0;
    for n =1:given                                % This loop does all the
                                                  % work, it calculate the
                                                  % average of the (given)
                                                  % number of data point
                                                  % behind and in from of
                                                  % the current data point
                                                  % being read

    forward=forward+filtered(i-n+1);
    backward=backward+filtered(i+n-1);
    end


    if abs(forward-backward) < allo               % This condition statment
                                                  % look for case when the
                                                  % difference between the
                                                  % reverse average and the
                                                  % forward average are the
                                                  % same
        st(i-(given-1))=backward/given;           % if they are there is no
         temp(i-(given-1))=abs(forward-backward); % step and the backward
                                                  % data is used
    else
        st(i-(given-1))=forward/given;            % if there is the forward
        temp(i-(given-1))=abs(forward-backward);  %  data is used
    end
end
st=st';
```

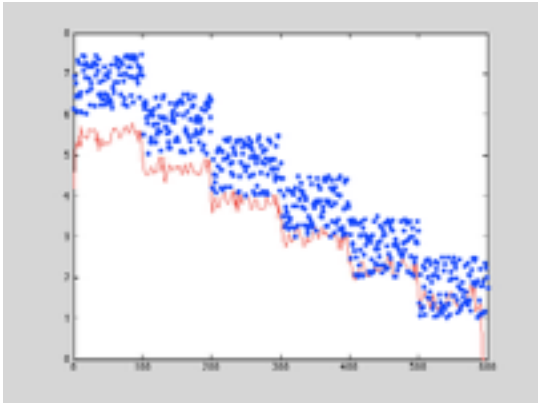
plot(out);

The data is a set of noisy steps, it could be anything from the optical output of several molecules, to the switching on and off of nerves in the body. The original Chung Kennedy filter was used to study neurons signaling. The filter we have is not a true Chung Kennedy (CK) filter but a very simple copy. The reason why there is a filter called the Chung Kennedy is because Chung and Kennedy need a filter which highlighted step edges, nerves and neurons are intrinsically noisy but the brain has very special filters which can see changes in the signal even though the signal is noisy. If you could do this with stock market data you would be a very rich person. The script for a CK filter is below.



To use this script I have written you need to have saved as variables the data you want to smooth and the range over which the function operates. To do this you will need the following commands, once you have loaded the data from the website.

data = out;
given = 3;

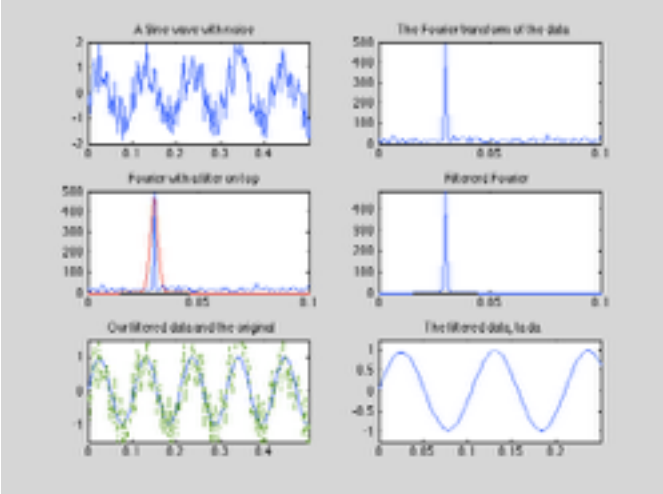


Because we are using a script rather than a function we have to have the data in memory. The way the filter works is as follows.
It first smooths the data, then for each data point it works out the average value of that a the previous few data points, this is stored in the variable **backward**. The number of data points it averages is defined by the given variable, so in this case it would be 3. The script also does this for the data points in front of the current data point this is stored in the variable **forward.** The script then compares the values of forward and backward, if they are similar then the current data point, is not near a step and the script stores the value of backwards. However, is they are different, then the script knows that the current data point is near a step and it stores the value of forward. This stops the script from smoothing out the edges in data sets like this. In the script, the variable **temp** holds the value of the difference between forward and backward for each point, if you plot this variable you can see the points at which the script switched between backwards and forwards.
The output of this script is stored in a variable called **st**, plot this and compare it with the original data (the data you loaded was called **out**). You can see from the plot to the right that the steps in the data are clearly visible. The CK filter is a very useful filter in nanoscience, but the point in using it here is to show you how you can build a filter that makes decisions, these types of filters can be extremely powerful. **Class exercise:** Convert this filter script in to a function taking the inputs data and given, where data is the data to be filtered and given is the range over which the function looks for changes in forward and backward.

The final filter we are going to look at is a filter based on a Fourier transform. The Fourier transform will be a topic covered in detail during you degree. For now we will say that a Fourier transform converts data taken in time to data as a function of frequency. The following filter will take a noisy but regular signal and remove the noise from the signal. It does this very effectively because the noise happens at many different frequencies but out

```
position = 0.03 ;                                   %This is the centre of our filter
width = 0.0025;                                     %This is the width of our filter

X=linspace(0,pi(),1000);                            %This is our X data in real space
random=rand(1,1000);                                %This is some noise
Y=((random-.5)*2)+sin(60*X);                        %This is our noisy data in real space
subplot(3,2,1);                                     %setting up the plot
plot(X,Y);                                          %plot our data
axis([0 .5 -2 2]);                                  %set up the axis
title('A Sine wave with noise');                    %Give the plot a title

[temp L]=size(X);                                   %Get the number of X data points
T=1/1000;                                           %Set up a scaling factor to plot
                                                  %frequency space
t = (0:L-1)*T;                                      %Set up our frequency space X data
Fy=fft(Y);                                          %perform a fourier transform of our
                                                    %data
maxv=max(abs(Fy));                                  %Find the maximum of the FFT, we use
                                                    %the abs
subplot(3,2,2);                                     %function because the FFT gives us
                                                  %imaginary numbers
plot(t,abs(Fy));                                    %plot the FFT using the abs function
axis([0 .1 0 500]);                                 %set up the axis
title('The Fourier transform of the data');         %give it a title

window=Gausf([0 maxv position width ],t);           %Here we use the gaussian you made
                                                  %earlier
subplot(3,2,3);                                     %This just shows the position
plot(t,window,'r',t,abs(Fy),'b');                   %and width of the gaussian
axis([0 .1 0 500]);                                 %with respect to the Fourier
actualwindow=Gausf([0 1 position width ],t);        %transform I actually make two
title('Fourier with a filter on top');              %gaussians one for the plot called
                                                  %window
                                                    %and one to use use for the
                                                    %filter called actualwindow
                                                    %the difference is just the scale

 YY=actualwindow.*Fy;                               %if we multiple the gaussian by the
                                                    %fourier transform it acts
                                                    %as acts as afilter
                                                    %selecting on the part of
                                                    %the fourier transform we
                                                    %are interested in we plot
                                                    %that here

subplot(3,2,4);
plot(t,abs(YY));
axis([0 .1 0 maxv]);
title('Filtererd Fourier');
FF=ifft(YY);                                        %This is the clever bit we perform
                                                    %a reverse fourier
                                                    %transform of our filtered
                                                    %data and this is plotted
                                                    %here with the original data

subplot(3,2,5);
plot(X,real(2*FF),X,Y,'-.');
axis([0 .5 -1.5 1.5]);
title('Our filtered data and the original');
subplot(3,2,6);
plot(X,real(2*FF));
axis([0 .25 -1.25 1.25]);
title('The filtered data, ta da');
```

data only happens at one. So if we perform a Fourier transform and only select data at the frequency of our signal we will have removed most of the noise from our data. The script is given below



This is the longest script we have written so far and will be on the website but you should look over every line and make sure you understand what it is doing. When you run the script it will produce six figures as given in the image to the right. The top left plot is a noisy Sine wave, the top right plot is the Fourier transform of the data in the first plot. You can see the spike of Fourier transform, this corresponds the frequency of our Sine wave. Note that in the left image the X axis in in time and on the right the X axis in in frequency.

On the middle line we plot the Fourier transform and a Gaussian which we have defined in the script using the variables **position** and **width,** these control the position and width of the Gaussian. The left plot shows this Gaussian and the Fourier transform, the right shows the resulting data set when the Fourier data is multiplied by the Gaussian, here we use a Gaussian with the same width and position but we set the peak to unity so that we don't amplify our data. In the code you will see two Gaussian filters one called 'window' and one called 'actual window'. I have used widow to have a similar amplitude to the Fourier transform so that when they are plotted they look the same. However, when I want to filter my Fourier data I use 'actualwindow' which has an amplitude of unity. You can see in the right middle image the filtered Fourier data, everything but the frequency we are interested in is set to zero.

Now we have the filtered Fourier transform, when we perform and inverse Fourier transform to go from frequency space back in to the time domain, we will have only data with the same frequency as that under our Gaussian window. This data is given in the bottom two plots, on the left is the filtered data with the original data overlaid, on the right is just the filtered data. Try adjusting the position and width of the Gaussian filter.