Chapter 2. Matlab Functions

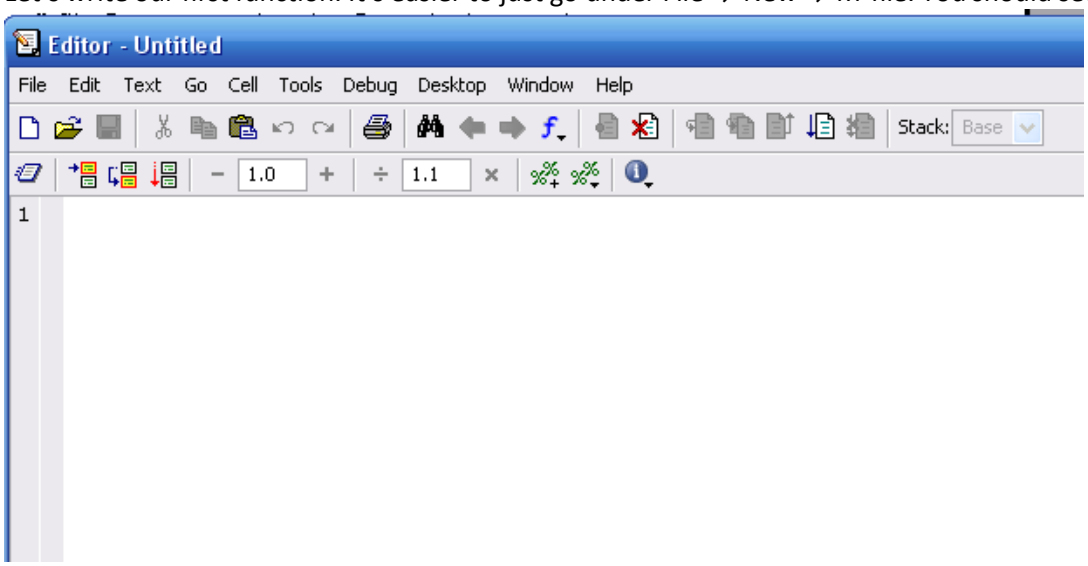
1.2 Writing functions in Matlab

In Matlab, every mathematical function (such as sin) is actually a series of instructions in a "function_name.m" file. In my example, when I type in the consul:

```
>>sin(3.14)
ans =
   0.0016
```

What is happening is the number 3.14 is being sent to file sin.m for processing such the function returns the value 0.0016. (in my computer, the file sin.m is located at: C:\ProgramFiles\MATLAB\R2006b\toolbox\eml\lib\matlab). The interesting thing is you can write your own functions that can do whatever you want, like calculate the error in a fit to your data.

Let's write our first function. It's easier to just go under File → New → M-file. You should see this:



On the first line type:

```
function [return_val]=fitter(x)
```

Now here is what these things do:

function → This tells Matlab that your writing a function. What this means is that the function is "blind" to the consul (the thing you're typing commands in). If dataset has been loaded into the consul, the function still cannot use it. Likewise, if dataset is altered in the function, it is not altered in the consul- several examples of what this means are given below.

[return_val] → This is the value that the function will return, in our example above for sin(3.14), it was 0.0016.

fitter → The name of the .m file. When you save it, make sure you save the name of the file as "fitter".

(x) → This is what you pass to the function, in our previous example of sin(3.14), x is equal to 3.14 inside the function. It can also be a vector, in other words, it can have two values; x(1) and x(2) for example.

Now let's write a function that multiplies two numbers together. In your .m file, write the following:

```
function [return_val]=fitter(x)
kitty=x(1)*x(2);
return_val=kitty;
```

Now save the .m file as fitter.m. In the consul, type:

```
>>x(1)=5; x(2)=6;
>>fitter(x)
ans =
   30
```

See? It's really easy. Note the following: change your .m file as:

```
function [return_val]=fitter(x)
i=444;
kitty=x(1)*x(2);
return_val=kitty;
```

Save it and type the following:

```
>>x(1)=5; x(2)=6; i=666;
>>i
i =
  666
>> fitter(x)
ans =
   30
>> i
i =
  666
```

Note that the value of i appears to be redefined in fitter.m from the value 666 to 444; however, it is actually unchanged from your assignment of 666 in the consul. This is what I mean when I say that the consul is "blind" to the actions of the function and vise versa.

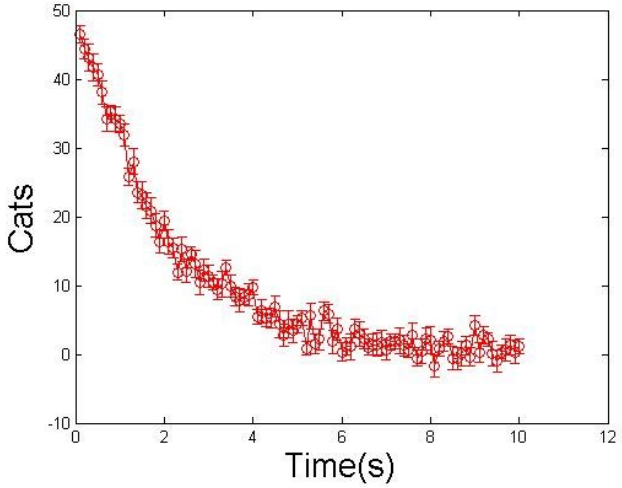
Now let's write something relevant- a function that can calculate the error of a fit. By error, I mean χ^2 (chi squared). From your laboratory manual, χ^2 is:

$$\chi^2 = \sum_{i=1}^{N} \left(\frac{1}{\sigma(i)^2}\right) [\text{data}(i) - \text{fit}(i)]^2$$

where the summation is over the total number (N) of data points and $\sigma(i)$ are the error(s) of each data point. Let's write a function that calculates χ^2 for a dataset that represents the decay of cats over time in a grinder. The dataset, called "Dataset2_1.txt" which a part of this packet, has time in the first row, number of cats in the second, and σ of each data point in the third row. First let's plot the data with error:

```
>> mydata2=load('Dataset2_1.txt')
>> errorbar(mydata2(:,1),mydata2(:,2),mydata2(:,3),'ro-')
>> d=xlabel('Time(s)');
>> set(d,'FontSize',20)
>> f=ylabel('Cats');
>> set(f,'FontSize',20)
```

The output is:



We wish to model the experimental data with an exponential decay:

$$\text{Cats} \cdot e^{-t/\tau}$$

and calculate χ^2 of that fit. The following .m file does just that:

```
function [return_val]=fitter(x)
mydata2=load('Dataset2_1.txt');

chisq=0;
for i=1:100
    fit(i)=x(1)*exp(-mydata2(i,1)/x(2));
    chisq=chisq+(1/mydata2(i,3)^2)*(mydata2(i,2)-fit(i))^2;
end;
return_val=chisq;
```

Here the number of cats is represented by x(1), the decay time constant τ is represented by x(2), the time variable t is the first column of mydata2, the experimental results are in the second column, and the error in the measured number of cats per unit time σ (as they tend to get mashed up together) is in the third column. Note that there are 100 data points. Let's look at some salient features of this file. You have to load the dataset mydata2.txt (and make sure your consul, your

fitter.m file, and mydata2.txt are all in the same directory!); this is a result of the "disconnect" between the consul and the function. Just because the dataset mydata2.txt may be loaded into the consul means nothing to the function! You must also pass to the function the vector x that has the fit parameters. The function then returns the value of chi squared.

To solidify this, let's use it and see that happens! I will first make a guess at what the parameters are that describe this horrible decay of cats. It seems that we started with 50 cats. Also, they are mostly gone within 3 seconds. As such, my guess is:

```
>>a(1)=50;
>>a(2)=4.0;
>>fitter(a)
ans =
  3.9663e+03
```

Where the ans is χ^2 of the exponential decay fit described the parameters a(1) and a(2). Note that we can use this to find the best fit parameter. For example, change a as:

```
>>a(2)=2.0;
>>fitter(a)
ans =
  124.0156
```

It's a better fit since the sum of the differences with the data is less! Let's keep going!

```
>>a(2)=1.0;
>>fitter(a)
ans =
  2.0833e+03
```

Oops, that's worse. Well, what you have to do is keep changing the variables until you get the lowest result. It may take hours!
Just kidding. Matlab has a solution: type the following:

```
>> fminsearch('fitter',a)
ans =
   50.2501    2.0113
```

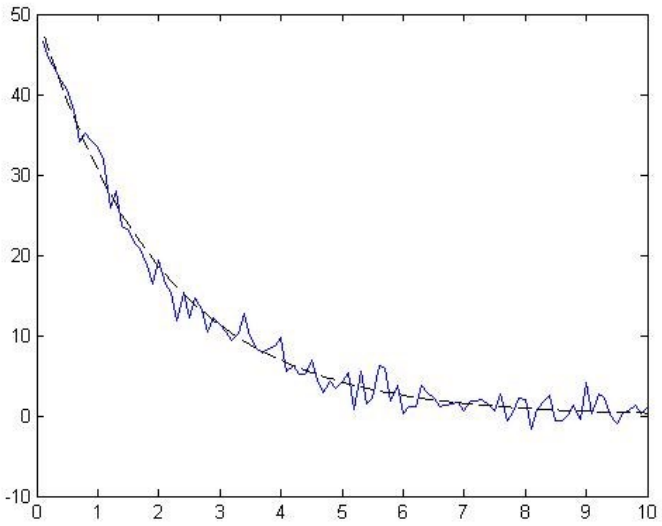
This is what is happening- fminsearch is querying the result of fitter.m for your value of "a". Next, it starts changing "a" in a directed way to lower the result. After a fair number of trials, it will give you its best result which is the value(s) of "a" that give the lowest possible sum of the differences in the data and the fit. Now type this:

```
>>a
a =
    50     1
>> fitter(a)
ans =
  2.0833e+03
```

```
>> fminsearch('fitter',a)
ans =
  50.2501    2.0113
>> a2=ans;
>> fitter(a2)
ans =
  123.2405
```

Clearly, the exponential decay described by the vector a2 is much better than our original guess. You should always double check the results however:

```
>> for i=1:100 fit(i)=a2(1)*exp(-mydata2(i,1)/a2(2)); end;
>> plot(mydata2(:,1),mydata2(:,2))
>> hold on
>> plot(mydata2(:,1),fit,'k--')
```



See? This looks like a very good fit. You can use this type of analysis for your gas effusion data!

Here is the most important point- when you make your own program, you don't have to necessarily use the equation of a line to fit non-linear data!

FYI: If you are totally stuck or make a mistake and can't figure it out, the file fitter.m is included in this tutorial.

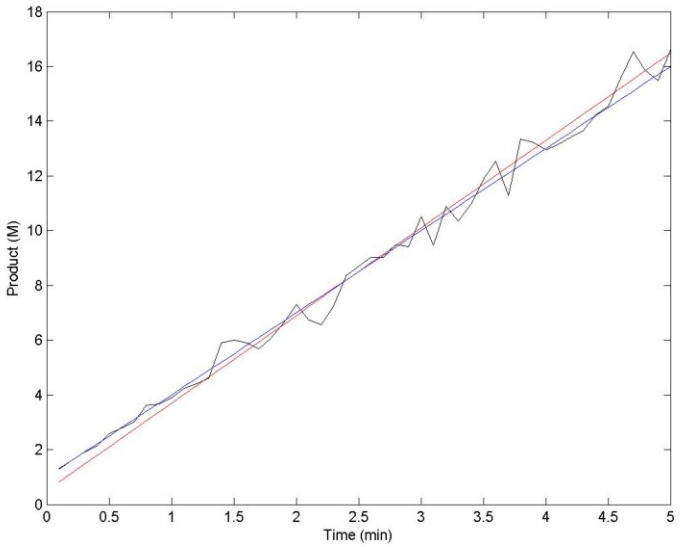
NOW YOU CAN FIT DATA TO ANY FUNCTIONAL FORM

Matlab Assignment

1. With this packet is a dataset (Problem2_1.txt) consisting of 2 by 50 points; the first column is time and the second is product formation. It is roughly linear. Your job is to calculate R^2 for three possible fits using Matlab:

a) A line described by Product(time)=3.2×time + 0.5
b) A line described by Product(time)=3.0×time + 1.0
c) A line described by Product(time)=2.8×time + 1.2

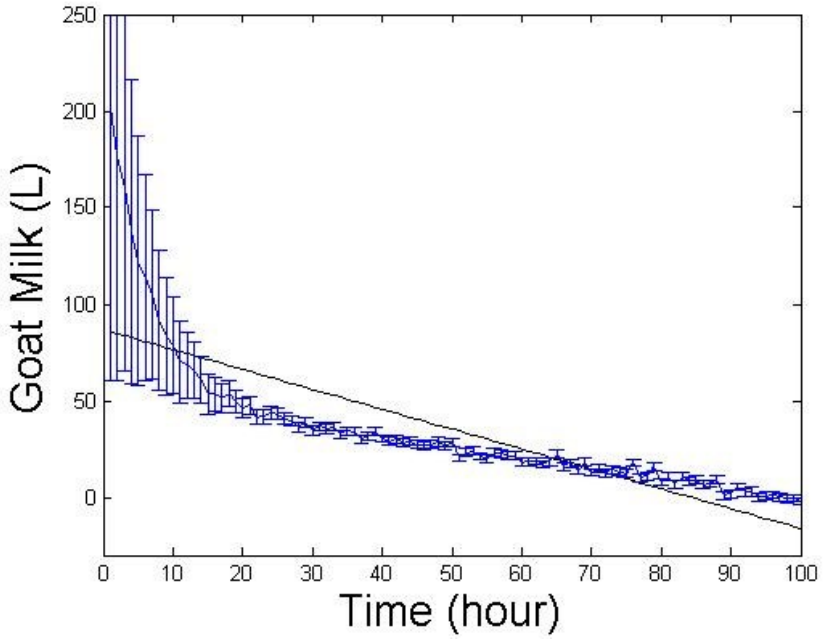
Last part: make a figure of the raw data and all three fits. Here is an example with two of the fits:



2. This exercise is used to show you how important error is to the result of a fit. As part of this packet is dataset "Problem2_2.txt", consisting of 3 by 100 points: the first column represents time, the second Liters of goatmilk, and the third the standard deviation [$\sigma_{(i)}$] of goatmilk that also has units of Liters. It is roughly linear although the first few points have a huge amount of error associated with them due to the early-time randomness of goatmilk production. Use the command:

```
>>errorbar(Problem2_2(:,1),Problem2_2(:,2),Problem2_2(:,3))
```

to see a graph with the standard deviations of the actual data points. First, calculate the linear least squares fit to the data using any available software you would like (note that I have a Matlab script, llsq.m, which calculates this for you and is included in this packet). With the parameters in hand, plot the data and the best fit line; it may look something like:



This is obviously a very bad fit. Your job is to determine the real best fit line by minimizing a MATLAB function that calculates:

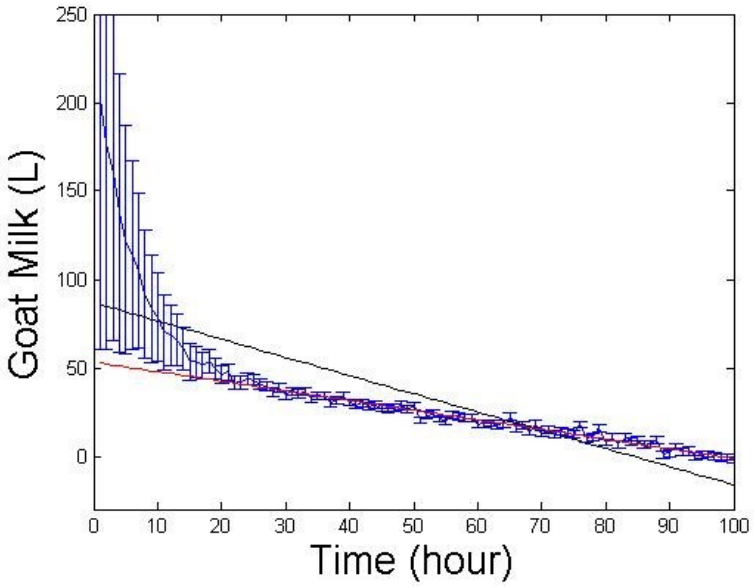
$$\chi^2 = \sum_{i=1}^{N} \frac{1}{\sigma_{(i)}^{\,2}} \cdot (\mathrm{data(i)} - \mathrm{fit(i)})^2$$

where the error of each data point σ(i) is considered. Here is an example to calculate χ^2 (albeit incomplete):

```
chisq=0;
for i=1:100
   chisq=chisq+(1/Problem2(i,3)^2)*(Problem2_2(i,2)-fit(i))^2;
end;
```

Complete the function to calculate χ^2 and then use fminsearch to determine the real best fit slope and intercept to the goatmilk data. When done properly, you should use the result of fminsearch to plot the new best linear fit. Last, send me for full credit a figure with the original data with errorbars, the linear least squares fit, and your own fit. The answer should look like the figure on the next page.

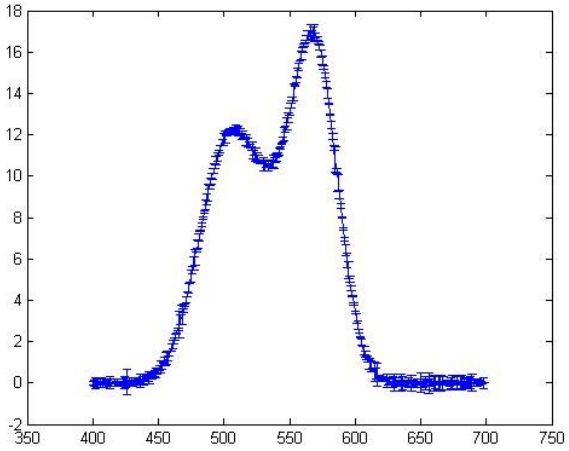
The lower line that takes into account the error of each datapoint is clearly the best fit and is mostly within the error bars.



3. Hidden data. We are using "Problem2_3.txt" as part of this packet. It contains 3 by 300 points: the first column represents wavelength in nm, the second is emission, and third are the error bars. It appears to be composed of two Gaussian peaks. A Gaussian has the functional form of:

$$f(t) = A \cdot e^{-(\lambda-\lambda_{center})/2\sigma^2}$$

where A is the amplitude, λ_{center} is the position of the peak, and σ is related to how wide the peak is.



However, I'm not so sure that this is a doubly peaked emission spectrum; perhaps there are three peaks and the third one is hidden under the other two. First, try to fit these data using fminsearch to a 2-Gaussian fit. Here is how you program a (single) Gaussian in Matlab:

```
for i=1:300
gausfit(i)=x(1)*exp(-(Problem2_3(i,1)-x(2))^2/x(3));
end;
plot(problem5(:,1),gausfit);
```

The amplitude is x(1), x(2) is λ_{center}, and x(3) is $2\sigma^2$. Use fminsearch to calculate the best fit to the data using two Gaussian functions. If (when) the data don't fit well, add a third Gaussian (try an initial guess between the main two peaks). In the three Gaussian fit, you will be optimizing nine parameters. Sometimes this causes Matlab to get "stuck" and give up; if so, you will see the following:

```
>>fminsearch('fitter',x)
Exiting: Maximum number of function evaluations has been exceeded
    - increase MaxFunEvals option.
    Current function value: 0.017758
ans =
  10.1798  500.2755  905.2207   15.0907  569.9621  600.7211    5.8533  535.3556
956.3240
```

If so, try "recycling" the last set of optimized variables as so:

```
>> fminsearch('fitter',ans)
```

You may have to do this several times. Eventually Matlab settles on a set of parameters it thinks is optimal and will no longer give you that error.

And, if you get completely stuck, programs are provided in the packet that answer the question.

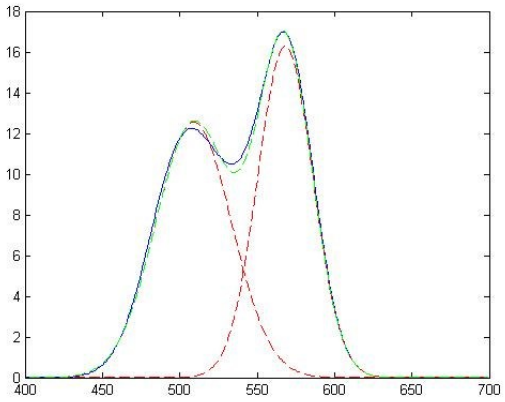
Answers:

1. a) R^2=0.9880 b) R^2=0.9891 c) R^2=0.9759

2. linear least squares fit slope= -1.0287 ± 0.0767326
intercept= 86.6580 ± 4.46337

Fminsearch slope=-0.5451 intercept= 53.23

3. If you use two Gaussians, you get this (and a χ^2 of 2.6817e+03).



If you use three Gaussians, you get this (and a χ^2 of 8.9445e-06).