Chapter 3. The Variance-Covariance Matrix

3.1 Non-linear Error Analysis

Our biggest feat so-far has been fitting a linear function to a set of data by minimizing the least squares differences from the fit to the data with fminsearch. When analyzing non-linear data, you have to use a program like Matlab as many types of data cannot be linearized such that Excel can analyze it. This also means that you do not have a direct route to calculating the error of your fit, as you have been using the error calculated from the linear least squares result (this doesn't work on non-linear data!).

Now you can make one of two assumptions- 1) error cannot be calculated from a non-linear fit or 2) error can be calculated from a non-linear fit, I just don't know how.
Unless you're a fre@¿*ħ↗≈*in' moron, you should pick 2.

Here is how! The variance-covariance matrix.

I will not cover the derivation, not that I don't understand it (I so totally do) but it is several pages of algebra long. I will show you the formula for the error analysis and prove it works by applying it to a linear set of data. Then we will use it for other problems…

Step 1: define a matrix of partial derivatives.

This is the only semi-hard part, you have to calculate the partial derivative of the function you are fitting your data to for each variable that you are minimizing in your least squares fit (remember fminsearch from the previous lesson?). Let's say you are fitting data to a function f(m,b) which has two variables that were fit my minimizing the error with fminsearch, call them m and b. The first step to calculate σ_m and σ_b is to derive the partial derivative matrix:
M=

$$\begin{vmatrix} \frac{\partial f(m,b)}{\partial m}\cdot\frac{\partial f(m,b)}{\partial m} & \frac{\partial f(m,b)}{\partial m}\cdot\frac{\partial f(m,b)}{\partial b} \\ \frac{\partial f(m,b)}{\partial b}\cdot\frac{\partial f(m,b)}{\partial m} & \frac{\partial f(m,b)}{\partial b}\cdot\frac{\partial f(m,b)}{\partial b} \end{vmatrix}$$

Where $\frac{\partial f(m,b)}{\partial m}$ is the partial derivative with respect to the variable your fitting (m) and likewise for $\frac{\partial f(m,b)}{\partial b}$ with variable b. Note that the diagonal (1,1) and (2,2) elements are **NOT** equal to the second derivative $\frac{\partial f(m,b)^2}{\partial^2 m}$ but are in fact just the first derivative squared, i.e. $\left(\frac{\partial f(m,b)}{\partial m}\right)^2$. The off-diagonal elements are the partial derivatives multiplied by each other. If this is confusing now, we will make it a lot clearer with an example on the next page. For now, let me simplify the matrix **M** as:

$$\begin{vmatrix} \left(\frac{\partial f(m,b)}{\partial m}\right)^2 & \frac{\partial f(m,b)}{\partial m}\cdot\frac{\partial f(m,b)}{\partial b} \\ \frac{\partial f(m,b)}{\partial b}\cdot\frac{\partial f(m,b)}{\partial m} & \left(\frac{\partial f(m,b)}{\partial b}\right)^2 \end{vmatrix}$$

Now let me be totally honest here: the matrix **M** is actually a sum over all data N points and is properly expressed as:

$$\begin{vmatrix} \sum_{i=1}^{N}\left(\frac{\partial f(m,b)}{\partial m}\right)^2 & \sum_{i=1}^{N}\left(\frac{\partial f(m,b)}{\partial m}\cdot\frac{\partial f(m,b)}{\partial b}\right) \\ \sum_{i=1}^{N}\left(\frac{\partial f(m,b)}{\partial b}\cdot\frac{\partial f(m,b)}{\partial m}\right) & \sum_{i=1}^{N}\left(\frac{\partial f(m,b)}{\partial b}\right)^2 \end{vmatrix}$$

Now before this gets confusing, let's cement everything with a simple example. Here is a set of data that we can fit (Dataset3_1.txt):

x_i	y_i
0	1.9
1	4.2
2	5.9
3	7.8
4	11
5	12.2
6	14.1

You should know how to load these values into Matlab, which we call mydata3. So let's start with a linear fit; first we define the partial derivatives of the function:

$$f(x(i)) = y(i) = m \cdot x(i) + b$$

Given this definition, we can determine that:

$$\frac{\partial f(m,b)}{\partial m} = x(i)$$

and:

$$\frac{\partial f(m,b)}{\partial b} = 1$$

Done! Let's put this into the matrix:

$$\begin{vmatrix} \sum_{i=1}^{N}\left(\frac{\partial f(m,b)}{\partial m}\right)^2 & \sum_{i=1}^{N}\left(\frac{\partial f(m,b)}{\partial m}\cdot\frac{\partial f(m,b)}{\partial b}\right) \\ \sum_{i=1}^{N}\left(\frac{\partial f(m,b)}{\partial b}\cdot\frac{\partial f(m,b)}{\partial m}\right) & \sum_{i=1}^{N}\left(\frac{\partial f(m,b)}{\partial b}\right)^2 \end{vmatrix}$$

which is now:

$$\begin{array}{|c|c|}\hline \sum_{i=1}^{N}(x(i)\cdot x(i)) & \sum_{i=1}^{N}(x(i)\cdot 1) \\ \hline \sum_{i=1}^{N}(1\cdot x(i)) & \sum_{i=1}^{N}(1\cdot 1) \\ \hline \end{array}$$

Simplifying gives:

$$\begin{array}{|c|c|}\hline \sum_{i=1}^{N}x(i)^2 & \sum_{i=1}^{N}x(i) \\ \hline \sum_{i=1}^{N}x(i) & N \\ \hline \end{array}$$

Let's calculate each element with Matlab to show you how simple these formulas are:
To define the 2-by-2 matrix above (called m):

```
>> mydata3=load('Dataset3_1.txt');
>> m(1,1)=0; for i=1:7 m(1,1)=m(1,1)+mydata3(i,1)^2; end;
>> m(1,2)=0; for i=1:7 m(1,2)=m(1,2)+mydata3(i,1); end;
>> m(2,1)=0; for i=1:7 m(2,1)=m(2,1)+mydata3(i,1); end;
>> m(2,2)=0; for i=1:7 m(2,2)=m(2,2)+1; end;
>> m
m =
   91   21
   21    7
```

YOU'RE NOW DONE WITH THE HARD PART!!

Step 2- This matrix must be inverted.

Now what does it mean to invert a matrix? The inverse of matrix **M** (called $\mathbf{M}^{-1}$) has the property such that $\mathbf{M}\cdot\mathbf{M}^{-1} = \mathbf{1}$. Not quite the number 1, but a matrix with the same number of elements as M, each diagonal element being the number 1 (others are 0). So in this example:

a	b
c	d

The inverse is (see http://mathworld.wolfram.com/MatrixInverse.html for more info):

$$\begin{vmatrix} \left(\frac{1}{ad-bc}\right)\cdot d & -\left(\frac{1}{ad-bc}\right)\cdot b \\ -\left(\frac{1}{ad-bc}\right)\cdot c & \left(\frac{1}{ad-bc}\right)\cdot a \end{vmatrix}$$

Multiply the matrix with its inverse using the rules of matrix algebra and you get:

$$\begin{vmatrix} 1 & 0 \\ 0 & 1 \end{vmatrix}$$

Here is the great thing- **don't worry about any of this**- Matlab does everything for you!

```
>> m_inv=inv(m)
m_inv =
   0.0357   -0.1071
  -0.1071    0.4643
```

Let's double check it:

```
>> m_inv*m
ans =
    1    0
    0    1
```

See, once you set up the **M** matrix and its inverse, you don't have to do anything else!

YAY! YAY MATLAB!!

Step 3- Lets first talk about the data we are fitting: we will define N as the number of data points to fit, and lets call the actual data we wish to fit y(i). In our case, there are 2 variables to the function we are fitting the data to (m and b), so let's define p as this quantity (p=2). For step three, we have to know something about the deviation of the data from the fit; that obviously must play a part in the errors of m and b. To do this part, define s_y^2 as:

$$S_y^2 = \frac{\sum_{i=1}^{N}(y(i) - fit(i))^2}{N - p}$$

Almost done! Define the best fit from fminsearch for the line (which are the same parameters off my Excel spreadsheet oddly enough):

```
>> for i=1:7 fit(i)=2.060714*mydata3(i,1)+1.975; end;
```

Next step:

```
>> sy2=0;
>> p=7-2;
>> for i=1:7 sy2=sy2+((mydata3(i,2)-fit(i))^2)/p; end;
>>sy2
sy2 =
   0.1748

>> varcovar=m_inv*sy2
varcovar =
   0.0062   -0.0187
  -0.0187    0.0812
```

This is it! The Variance-Covariance Matrix!!

Actually, this is kinda like learning that the ultimate answer to the ultimate question in the universe is the number 42. You have to fully understand the question to truly get the answer.

The variance-covariance is actually equal to:

σ_m^2	$\sigma_m \times \sigma_b$
$\sigma_b \times \sigma_m$	σ_b^2

So if you want to know the error of the slope you type:

```
>> sqrt(varcovar(1,1))
ans =
```

0.0790

Likewise for the intercept:

```
>>sqrt(varcovar(2,2))
ans =
```

0.2849

Rememer the result from the Excel spreadsheet? The linear least squares result was:

slope = 2.0607 ± **0.0790**

intercept: 1.975 ± **0.285**.

Looks like the variance-covariance matrix works!

Last bit, let's look back at the variance-covariance matrix:

```
>> varcovar
varcovar =
   0.0062   -0.0187
  -0.0187    0.0812
```

How did we know that the square root of the (1,1) element (upper left-handed) is the error in the slope? Easy, that was defined when you took the derivative of the equation for a line with respect to the slope as: $\frac{\partial f(m,b)}{\partial m}$, likewise for the intercept.

Now what do the off-diagonal elements $\sigma_m \times \sigma_b$ and $\sigma_b \times \sigma_m$ mean? First, the off-diagonal elements are the "covariances." The covariance elements tell you that if your calculated slope is too low, then your calculated intercept it too high (as in this case the covariances are negative.). If they are positive, then an underestimate in the slope means that your intercept is also underestimated (i.e. they are "going-together"). A large off-diagonal element means that there is a lot of "cross-talk" between the m and b variables. Ideally, the covariances are 0, meaning that if you made a bad fit to the intercept, it did nothing to your estimate of the slope. Thus, the variables are independent. This is unfortunately rarely the case with real data.

Here is the most important part- starting from the beginning, who the heck sez you have to work with linear fits?

YOU NOW CAN CALCULATE THE ERROR OF ANY FUNCTION YOU CHOOSE AS THE BEST TO FIT YOUR DATA

3.2 More advanced fitting, and exponential decays

An exponential decay is described by the equation:

$$f(t) = A \cdot e^{-k \cdot t}$$

where A is the amplitude, t is time, and k is the decay constant. Let's figure out how to make a variance-covariance matrix from this equation. Noting that we are only fitting two variables, and starting from the beginning. Here we are going to add to the fact that the matrix elements should be weighted by the values of the errors of the individual data points, if they are known:

M =

$\sum_{i=1}^{N} \left(\frac{1}{\sigma(i)}\right)^2 \left(\frac{\partial f(A,k)}{\partial A}\right)^2$	$\sum_{i=1}^{N} \left(\frac{1}{\sigma(i)}\right)^2 \left(\frac{\partial f(A,k)}{\partial A} \cdot \frac{\partial f(A,k)}{\partial k}\right)$
$\sum_{i=1}^{N} \left(\frac{1}{\sigma(i)}\right)^2 \left(\frac{\partial f(A,k)}{\partial k} \cdot \frac{\partial f(A,k)}{\partial A}\right)$	$\sum_{i=1}^{N} \left(\frac{1}{\sigma(i)}\right)^2 \left(\frac{\partial f(A,k)}{\partial k}\right)^2$

Now let's fill in the individual pieces:

$$\frac{\partial f(A,k)}{\partial A} = e^{-k\cdot t}$$

And:

$$\frac{\partial f(A,k)}{\partial k} = -A\cdot e^{-k\cdot t}\cdot t$$

Now let's plug this into the matrix, and we get

M=

$\sum_{i=1}^{N}\left(\frac{1}{\sigma(i)}\right)^2\left(e^{-k\cdot t}\right)^2$	$\sum_{i=1}^{N}\left(\frac{1}{\sigma(i)}\right)^2\left(e^{-k\cdot t}\cdot -A\cdot e^{-k\cdot t}\cdot t\right)$
$\sum_{i=1}^{N}\left(\frac{1}{\sigma(i)}\right)^2\left(-A\cdot e^{-k\cdot t}\cdot t\cdot e^{-k\cdot t}\right)$	$\sum_{i=1}^{N}\left(\frac{1}{\sigma(i)}\right)^2\left(-A\cdot e^{-k\cdot t}\cdot t\right)^2$

This can be simplified as:

$\sum_{i=1}^{N}\left(\frac{1}{\sigma(i)}\right)^2 e^{-2\cdot k\cdot t}$	$\sum_{i=1}^{N}\left(\frac{1}{\sigma(i)}\right)^2\left(-A\cdot t\cdot e^{-2\cdot k\cdot t}\right)$
$\sum_{i=1}^{N}\left(\frac{1}{\sigma(i)}\right)^2\left(-A\cdot t\cdot e^{-2\cdot k\cdot t}\right)$	$\sum_{i=1}^{N}\left(\frac{1}{\sigma(i)}\right)^2 A^2\cdot t^2\cdot e^{-2\cdot k\cdot t}$

Now try to apply the above to Dataset3_2.txt, if you recall that from the last module where you fit an exponential fit to some data as shown here:

When you did the fitting to a function:

$$a(1)\cdot e^{-t/a(2)}$$

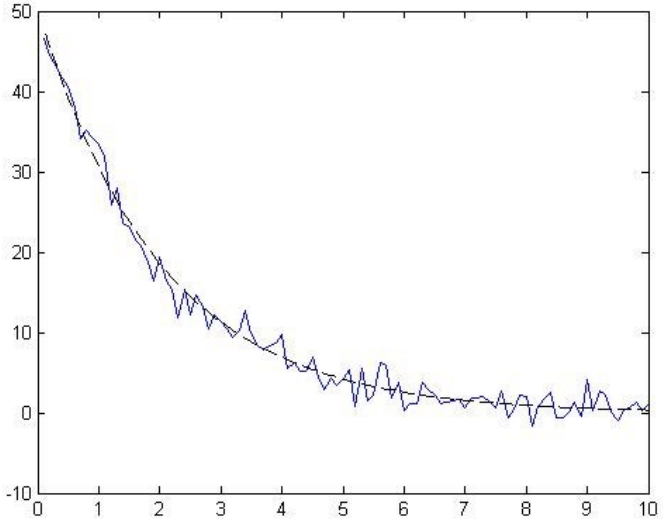
you found an amplitude of a(1)= 50.2501 and a time constant a(2)= 2.0113. If you do the above error analysis, you should find:

Amplitude: 50 ± 3

Time constant: 2.01 ± 0.18

Note that we are using proper error reporting values in the last step.

Included in this packet is "chapter3_cheatcode.m", a script which will do this analysis.



Matlab Assignment

1. Exponential Decay. Use the file "Problem3.txt" enclosed in this packet. It contains 3 by 500 points: the first column represents time in minutes, the second Liters of Cat, and third are the error bars. It is roughly exponential, which is a function of the form:

$$f(t) = A \cdot e^{-k \cdot t}$$

where A is the amplitude and *k* is the rate or decay constant. Here is a good way to make an exponential decay in Matlab-ese:

```
for i=1:500
fit(i)=x(1)*exp(-x(2)*Problem3(i,1));
end;
```

The amplitude is x(1) and the decay constant is x(2). We often call $\tau = 1/x(2)$, where τ (tau) is called the time constant.

Use fminsearch to calculate the best fit amplitude and decay constant. Don't forget you have to make a guess at the amplitude (it's about 10 Liters of Cat) and decay constant (it's about 1 minute)! Here is a shorter way to calculate χ^2 in your function:

```
chisq=0;
for i=1:500
   chisq=chisq+(1/Problem3(i,3)^2)*(Problem3(i,2)-x(1)*exp(-x(2)*Problem3(i,1)))^2;
end;
```

2. Next, calculate the error of these variables using the variance-covariance matrix. First, calculate the following matrix, which we will call m(1:2,1:2) :

$\sum_{i=1}^{N} \left(\frac{1}{\sigma(i)}\right)^2 e^{-2 \cdot k \cdot t}$	$\sum_{i=1}^{N} \left(\frac{1}{\sigma(i)}\right)^2 \left(-A \cdot t \cdot e^{-2 \cdot k \cdot t}\right)$
$\sum_{i=1}^{N} \left(\frac{1}{\sigma(i)}\right)^2 \left(-A \cdot t \cdot e^{-2 \cdot k \cdot t}\right)$	$\sum_{i=1}^{N} \left(\frac{1}{\sigma(i)}\right)^2 A^2 \cdot t^2 \cdot e^{-2 \cdot k \cdot t}$

As an example, the m(2,2) (i.e. the lower right hand one) is shown here:

```
m(2,2)=0;
for i=1:500
   m(2,2)=m(2,2)+(1/Problem3(i,3)^2)*(x(1)*Problem3(i,1)*exp(-x(2)*Problem3(i,1)))^2;
end;
```

Remember x(1) and x(2) are your optimized results from #1, not the guess!

Once you have this and the other three elements, invert the matrix:

```
>>m_inv=inv(m);
```

Calculate s_y^2 and then multiply the inverted matrix by it and multiply by it:

```
>>varcovarm=m_inv*sy2;
```

Remember that varcovar(1,1) isn't the variance in the amplitude, but the square of the variance in the amplitide!

Some hints: Write the matrix calculator in a script so that you will be easily able to make edits to it. Also, remember that variables in scripts are "seen" by the consul, which is not true of functions.

ps if you forgot how to calculate s_y^2:

```
sy2=0;
for i=1:500
sy2=sy2+((problem3(i,2)-fit(i))^2)/p;
end;
```

Remember that p is the number of points minus the number of fitted parameters, and if you get stuck, then you can look at the scripts that were included in this packet.

Answer:

The file “Ch3_problem_answer” provides the following:

$A = 10.2026 \pm 0.0988$ $k = 0.9741 \pm 0.2444$