CAP Program Tutorial: Model Fitting in Python

Katie Eckert (adapted from tutorials by Sheila Kannappan & Amy Oldenberg)

June 24, 2015

Please retrieve the tutorial information from http://github.com/capprogram/FittingTutorial and copy it to your working space. These files contain partial answers to some of the activities, that are for you to fill out. Solutions are also available for all of the activities. I have also included a derivation sheet.

Why do we fit models to our data?

- To understand the underlying distribution of our data
- To test hypotheses or competing theoretical models
- To predict values for future observations

These are just a few examples of why we might want to fit a model to our data. In this tutorial we will go through the basics of fitting model parameters to data using two different techniques: Maximum Likelihood Estimation and Bayesian Analysis.

Part I: Linear Least Squares Fitting & Maximum Likelihood Estimation:

Least Squares Fitting is based on the assumption that the uncertainty in your measurements follows a Gaussian distribution. In the linear case, the best solution is given by the slope & y-intercept parameters that minimize the root mean square (rms) of the residuals in the y-direction.

The rms squared is given by: $rms^2 = \sum \frac{(y_i - (\alpha x_i + \beta))^2}{\sigma_i^2}$ where x_i is the independent variable, y_i is the dependent variable with uncertainty σ_i, and α and β are the slope and y-intercept parameter values.

Least Squares Fitting falls within the category of parameter fitting known as *Maximum Likelihood Estimation (*MLE). In this method, we measure the likelihood for a given model using the χ^2 statistic:

The likelihood is given by: $L = \exp\left(\frac{-\chi^2}{2}\right)$ where $\chi^2 = \sum \frac{(y_{value,i} - y_{model,i})^2}{\sigma_i^2}$

To find the MLE solution to our model, we maximize the likelihood function by taking the derivative with respect to each parameter (the slope and y-intercept in the case of a linear fit) and by solving for where each derivative is equal to 0. To simplify the math, we first take the natural log of the likelihood function. For least squares fitting, it is possible to obtain an analytical solution for the slope and y-intercept. The derivation is shown below:

take the natural log of the likelihood function $\ln(L) = -\left(\frac{1}{2}\right)\chi^2 = -\left(\frac{1}{2}\right)\sum \frac{(y_i - (\alpha x_i + \beta))^2}{\sigma_i^2}$

take the derivatives of ln(L) with respect to α and β and set those equations to 0

$$\frac{d\ln(L)}{d\alpha} = -1 \frac{\sum (y_i - (\alpha x_i + \beta))(-x_i)}{\sigma_i^2} = 0 \qquad \frac{d\ln(L)}{d\beta} = -1 \frac{\sum (y_i - (\alpha x_i + \beta))(-1)}{\sigma_i^2} = 0$$

if we assume σ_i are all the same, we have two equations for two unknowns to solve

eqn 1: $\sum y_i x_i - \alpha \sum x_i^2 - \beta \sum x_i = 0$ **eqn 2:** $\sum y_i - \alpha \sum x_i - N\beta = 0$

multiply eqn 1 by N and multiply eqn 2 by $\sum x_i$

eqn1: $N \sum y_i x_i - N\alpha \sum x_i^2 - N\beta \sum x_i = 0$ **eqn2:** $\sum x_i \sum y_i - \alpha \sum x_i \sum x_i - N\beta \sum x_i = 0$

now we can set these two equations equal to each other and solve for α

$$N \sum y_i x_i - N\alpha \sum x_i^2 = \sum x_i \sum y_i - \alpha \left(\sum x_i\right)^2$$

$$\alpha = \frac{\sum x_i \sum y_i - N \sum (x_i y_i)}{\left(\sum x_i\right)^2 - N \sum x_i^2}$$ *divide top and bottom by N^2* $$\alpha = \frac{\frac{1}{N^2} \sum x_i \sum y_i - \frac{1}{N} \sum (x_i y_i)}{\frac{1}{N^2} \left(\sum x_i\right)^2 - \frac{1}{N} \sum x_i^2}$$

$$\alpha = \frac{\bar{x}\,\bar{y} - \overline{xy}}{\bar{x}\,\bar{x} - \left(\overline{x^2}\right)}$$ *now we can go back and solve for β:*

from **eqn 2** $N\beta = \sum y_i - \alpha \sum x_i$ **====>** $\beta = \frac{1}{N} \sum y_i - \frac{\alpha}{N} \sum x_i$ **====>** $\beta = \bar{y} - \alpha \bar{x}$

For more complicated functions or if the uncertainties are not uniform, the derivatives of the likelihood may not be possible to solve analytically, and we can use programs such as **np.polyfit** to determine the parameters numerically.

Activity 1: paramfit1.py

In paramfit1.py we create fake data with known slope and y-intercept. We then compute the maximum likelihood estimated slope and y-intercept for the fake data. Fill in lines ending with "?" Solutions are provided in paramfit1.py.sln

a) Plot the data. What does **npr.normal** do?

b) Read over the MLE derivation for the linear least squares analytical solution and compute the slope and y-intercept for the data set. Overplot the best fit solution.

c) For linear least squares fitting, we can obtain analytical solutions to the uncertainties on the slope and y-intercept estimates, which I have provided below.

$$\sigma_\alpha = \sqrt{\frac{\sum (y_i - (\alpha x_i + \beta))^2}{(N-2) \sum (x_i - \bar{x})^2}}$$ $$\sigma_\beta = \sqrt{\left(\frac{\sum (y_i - (\alpha x_i + \beta))^2}{N-2}\right) * \left(\frac{1}{N} + \frac{(\bar{x})^2}{\sum (x_i - \bar{x})^2}\right)}$$

See http://mathworld.wolfram.com/LeastSquaresFitting.html for the full derivation. Compute the uncertainties for the slope and y-intercept analytically. Which parameter has larger fractional uncertainty?

Read up on **np.polyfit**: http://docs.scipy.org/doc/numpy/reference/generated/numpy.polyfit.html

d) Use **np.polyfit** to compute the MLE slope and y-offset for the data set. Do you get the same result as in the analytical case from part b? Note that **np.polyfit** does not automatically take an array of uncertainties on the y value. If our uncertainties on each data point are different, we can input an optional weight vector: **fit=np.polyfit(xval,yval,w=1/sig)** where sig is an array containing the uncertainty on each data point. Note that the input is 1/sig (rather than 1/sig**2 as you might think from the equations above). The np.polyfit function squares the weight value within the source code.

e) Another method to determine the uncertainties is to use the covariance matrix:

$$C=\begin{pmatrix} \sigma_\alpha^2 & cov(\alpha,\beta) \\ cov(\alpha,\beta) & \sigma_\beta^2 \end{pmatrix}$$ which is the inverse of the Hessian Matrix (in pre-tutorial reading)

np.polyfit will compute the covariance matrix numerically if you add cov= “True” to the **np.polyfit** function call. Print out the uncertainties computed using the covariance matrix. Are they the same as the analytical solution? What happens to the uncertainties if you increase/decrease the number of data points? What happens to the percentage difference between the analytical and numerical methods if you increase/decrease the number of data points?

Activity 2: zombies 1

For this activity you will write your own code, but remember that you can take portions of previous code to make the process faster. Pull up the python quick reference guide and codes from previous tutorials to help you write the code for this activity. Please feel free to work together on this part. My solutions are provided in zombies1.py.sln

A virus has gotten out that is turning humans into zombies. You have been recording the % of zombies ever since the outbreak (~14 days ago). However the power has gone out for the past four days and your generator just kicked in allowing you to analyze the data and determine when there will be no humans left (% humans = [1-% zombies] = 0). Your data are in percentagezombie.txt

a) Read in your data and plot it as time vs. % human as blue stars. Our best estimate of the uncertainty on the time at which you take each data point about the zombie percentage that is about half a day. ***Note that we are putting time on the y-axis because it is the variable with the measurement errors.***

b) Evaluate the MLE slope & y-intercept and overplot the best fit line in green. What does the y-intercept value mean in the context of this situation? Are you a zombie?

c) In the above step you have fit the data minimizing residuals in the y-direction (time). How could you use **np.polyfit** to fit the data minimizing residuals in the x-direction (% humans)? Keep in mind that you can rewrite a line y=a*x + b as x =(1/a)*y – (b/a). Over plot this fit in red – how does the y-intercept value change? In which variable should you minimize residuals to get the most accurate prediction of the time when there will be no humans left?

d) In a new plotting window, plot the residuals of the linear fit from step 2 as green stars. Evaluate the reduced χ^2 for your data (refer to the Correlations and Tests Tutorial from June 9). Is your model a good fit to the data? Often times we think the R value from the Pearson correlation test tells us how good the fit is, but a reduced χ^2 actually gives a better estimate of how good your model is, not just whether the correlation is strong.

Optional: (if you are running low on time, skip ahead to Part II on Bayesian estimation)
e) What happens when you increase the order of the fit? Overplot the higher order fits on figure 1. What happens to the residuals if you increase the order of the fit (see **np.polyfit**, and **np.polyval**)? Overplot the new fits compared to the linear fit on figure 2.

f) Calculate the reduced χ^2 for these higher order fits – do they yield as good a fit to the data as the linear fit?

In the above examples we have assumed that the uncertainty on all of our data points is the same. This simplified assumption is often not the case. If the uncertainties are different, then must include each data point's uncertainty within the summations given in the derivation sheet.

Part II: Bayesian Estimation:
In *Bayesian analysis* we turn from maximizing the likelihood function to constructing the distribution of likelihoods for a grid of parameter space where we think our parameters are likely to be.

Bayesian analysis presents an entirely different philosophy in determining model parameters compared to the traditional MLE. In part I we determined the likelihood of the data given the model. In the Bayesian framework, we determine the likelihood of the model given the data.

Bayes's Theorem $$P(M|D)=\frac{P(D|M)*P(M)}{P(D)}$$

$P(D|M)$ --- the likelihood (as in part I)
$P(M)$ --- the prior probability (known information about the model)
$P(D)$ --- essentially a normalization factor
$P(M|D)$ --- the posterior probability of the model given the data

In Bayesian analysis we must set up the model with known *priors* on the possible range and distribution of the parameter space.

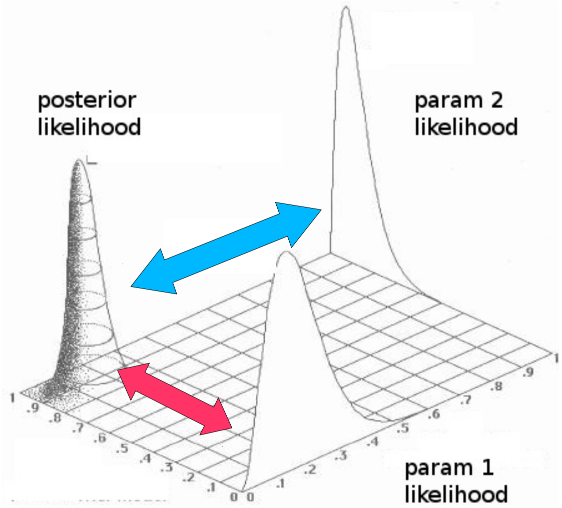
Activity 3:paramfit2.py
In paramfit2.py we use the fake data set created in paramfit1.py. This time, however, we will determine the slope and y-intercept through Bayesian analysis. grid of possible values of the slope and y-intercept and evaluate the likelihood of each grid point. Fill in any lines ending in “?”

a) Set up grids over the parameter space that we want to consider for the slope and y-intercept. What values are we considering? What is the implicit prior that we are considering?

b) Check that the priors are truly uniform. Plot lines of all possible y-intercepts and slopes? Is the y-intercept parameter evenly spaced? Is the slope parameter grid evenly spaced?

c) Read through:
http://jakevdp.github.io/blog/2014/06/14/frequentism-and-bayesianism-4-bayesian-in-python/
How can we compensate for the unequal spacing of the slope?

d) Compute the distribution of log likelihoods over all of parameter space using the assumption of a flat prior. How would you change your equation to take into account the correct prior from step c?



e) Now that we have our entire likelihood distribution over the entire parameter space, we can see the distributions of our individual parameters by summing the probability distribution of the other parameters (i.e., if we want to look at the posterior likelihood distribution of the slope, we sum over the likelihood distribution of the y-intercept). We call this procedure "marginalizing"

Plot the likelihood distribution of the slope and of the y-intercept. How do those values compare with the MLE values from paramfit1.py? How do the uncertainties compare with the MLE values? What happens if you change the number of data points? What happens if you change the grid spacing?

Optional - Activity 4:zombies 2

For this activity you will write your own code (but remember that you can take portions of previous code to make the process faster).

A virus has gotten out that is turning humans into zombies. You as a scientist have been recording the % of zombies ever since the outbreak (~14 days ago). However the power has gone out for the past four days and your generator just kicked in allowing you to analyze the data and determine when there will be no humans left (% humans = [1-% zombies] = 0). Use your Bayesian skills to perform this analysis.

a) Read in the data and plot it as time vs. %human

b) Create grids to test your parameter space assuming a linear model for the data. Compute the full likelihood distribution. What does your prior look like?

c) Determine the posterior likelihood distribution for the time at which there are 0% humans. Which parameter must you marginalize over?

d) Using the prior that you yourself are not a zombie yet and that you have provisions for another year, determine the likelihood distribution for the time at which there are 0%humans.

e) How does the Bayesian analysis for determining the time when there are 0% humans compare with the MLE fit from the second activity? Can you think of a better model for the data?

Don't worry if you did not through all of this tutorial. I have written up my solutions (which may look different from yours, but hopefully give the same answers) if you want to continue working on the activities at another time (feel free to ask me if you need any clarifications). If you are interested in any of the topics we covered today, there are plenty of references out there. For more info on the Bayesian approach check out the tutorials on the website below (they are all in python notebooks) http://jakevdp.github.io/blog/2014/06/14/frequentism-and-bayesianism-4-bayesian-in-python/ .

Also, I have provided Zane Beckwith's write up on the topic of optimization as well as two codes scatter.py & neldermead.py. Both of these python programs actually contain function definitions and can be read into your own code using the **import** command. The idea is that you would use the function **minimize** from neldermead.py to find minima of the Himmelblau function provided by the function **simulate** in scatter.py. I have provided my own example of how to do this in zanesprob.py.