CAP Program Tutorial: Model Fitting in Python

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

June 24, 2015

Please retrieve the zip file from http://github.com/capprogram/FittingTutorial and unzip the file in your working space. In the folder there are python codes (paramfit1.py and paramfit2.py) that contain partial answers for you to fill out. Two additional activities are codes to write on your own. Solutions are also available for all of the activities (with the extension .sln).

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: Least Squares Fitting & Maximum Likelihood Estimation of Model Parameters:

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 N²* $$\alpha = \frac{\frac{1}{N^2}\sum x_i \sum y_i - \frac{1}{N}\sum (x_i y_i)}{\frac{1}{N^2}\left(\sum x_i\right)^2 - \frac{1}{N}\sum x_i^2}$$

$$\alpha = \frac{\bar{x}\bar{y} - \overline{xy}}{\bar{x}\bar{x} - (\overline{x^2})}$$ *where the bar over the variable signifies the mean value of that quantity*

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) Run the program paramfit1.py and plot the data. What does **npr.normal** do?

b) Read over the MLE derivation for the linear least squares analytical solution (above) and compute the slope and y-intercept for the fake data set. Plot the best fit solution on top of the data. What is the assumption that we made for the uncertainties on our fake data?

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, 1, 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 expect from the equations above. The np.polyfit function squares the weight value within the source code.

In this example 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 MLE calculation.

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

$$C=\begin{pmatrix} \square_{\square}^{2} & cov(\square,\square) \\ cov(\square,\square) & \square_{\square}^{2} \end{pmatrix}$$ 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?

Optional Activity 2: zombies 1

If you are running low on time, skip this activity and move on to Part II. For this activity you will write your own code, but remember that you can take portions of previous code to make the process faster. Use the python quick reference card (http://user.physics.unc.edu/~sheila/PythonQuickReference.pdf) 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 percentzombie.txt where time=0 is the present day (time = -14 is 14 days ago when you started taking data).

a) Read in your data and plot it as % human vs. time as blue stars. The uncertainties on both time and % zombie are unknown.

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 (% human). How could you use **np.polyfit** to fit the data minimizing residuals in the x-direction (time)? 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? Does this change your conclusion as to whether you are a zombie? In which variable should you minimize residuals to get the most accurate prediction of total zombification?

d) Now assume your uncertainty on each % zombie measurements is ~3%. In a new plotting window, plot the residuals in % human from the linear fit from part b as green stars (residuals can be computed for either % human or time, but use % human since we have an estimated error for each data point of about 3%). Evaluate the reduced χ^2 for your data using residuals in % human as that is the measurement for which we have an uncertainty estimate (refer to the Correlations & Hypothesis Testing Tutorial). Is your model a good fit to the data? If 3% is an over- or under-estimate of the errors, how will this affect the reduced χ^2? 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.

e) What happens when you increase the order of the fit (% humans vs. time)? 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 optionally **np.polyval**)? Overplot the new residuals in time compared to the residuals from the linear fit from part b 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?

Part II: Bayesian Estimation of Model Parameters:

Bayesian analysis presents an entirely different philosophy of determining model parameters compared to the traditional MLE. In part I we determined the likelihood of the data given the model, which involved one model and many data sets. The one data set in activity 1 was imagined to be an example data set drawn from many repeat experiments, which can be mathematically modeled assuming the other experiments would provide data points distributed in Gaussian fashion around the actual data points or alternatively could be computationally modeled using Monte Carlo techniques. In the Bayesian framework, we determine the likelihood of the model given the data, which involves many models and one data set.

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

$P(D|M)$ --- the likelihood (as in part I), this is read as "probability of the data given the model"
$P(M)$ --- the prior probability (known information about the model set, an example is a flat prior, which weights all parameter possibilities hence all models equally)
$P(D)$ --- probability of the data, essentially a normalization factor
$P(M|D)$ --- the posterior probability of the model given the data

Whereas in MLE, we maximized the likelihood function to find the single "best" set of parameters, in Bayesian analysis we construct the posterior probability distribution, which is the distribution of probabilities for a grid of models spanning ranges of parameters where we think our parameters are likely to be (meaning we are setting the "prior" to 0 outside these ranges). To do this in practice, we compute the likelihood of the data given each model multiplied by the prior for each model over the entire model grid. In many problems scientists will assume a flat prior (all grid points weighted equally), and then the posterior probability distribution is proportional to the likelihood.

Activity 3: paramfit2.py

In paramfit2.py we use the same fake data set created in paramfit1.py. This time, however, we will determine the slope and y-intercept through Bayesian analysis by constructing a grid of possible values of the slope and y-intercept and evaluating the posterior probability at each grid point. Fill in any lines of code ending in "?".

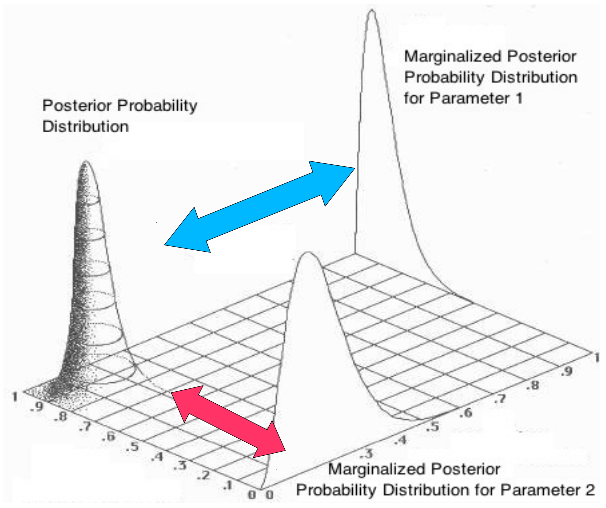
a) Run paramfit2.py and plot the fake data. Set up grids for the y-intercept and slope values that make sense. What values are we considering for the slope and the y-intercept? What is the implicit prior on the slope and y-intercept that we are considering?

b) Check that the model space from the choice of grid values for both the y-intercept and slope is also uniformly distributed. To do this plot a series of lines using all possible y-intercepts (y=x+beta_i) and then all possible slopes (y=x*alpha_i). Is the model space from the y-intercept parameter evenly spaced? Is the model space from the slope parameter evenly spaced? Note that uniform priors can be uniform in the parameter or in a function of the parameter, e.g. slope, log(slope), tan(slope), depending on what physically makes sense. What is a physically motivated definition of "evenly spaced" for the slope, based on studying the plots?

c) Read through: http://jakevdp.github.io/blog/2014/06/14/frequentism-and-bayesianism-4-bayesian-in-python/. How can we write a prior that compensates for the non-uniform weighting of the angles?
Note that in this article the definition of alpha and beta are reversed: y= α + β*x.

d) Using the fake data, compute the posterior probability distributions for the entire grid assuming 1) flat priors on the values of the slopes and y-intercepts (non-uniform in the angle) and 2) the prior that compensates and creates a uniform angular distribution. Pay attention to where the prior appears in the equation for computing the posterior probabilities.

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


Marginalized Posterior Probability Distribution for Parameter 1
Posterior Probability Distribution
Marginalized Posterior Probability Distribution for Parameter 2
1 .9 .8 .7 .6 .5 .4 .3 .2 .1 0
0 .3 .4 .5 .6 .7 .8 .9 1

Plot the marginalized posterior probability distributions of the slope using the two different priors (green for flat and red for compensating). Are there any differences (you may need to zoom in)? How do the marginalized posterior probability distributions of the slope compare with the MLE values from paramfit1.py? Estimate the uncertainty on the slope value from the plot by eye (you may need to zoom in on the region of significant probability). How does the uncertainty on the slope compare with the MLE value? Do the same for the y-intercept. What happens to the marginalized posterior probability distributions for the slope and y-intercept if you change the number of data points (try N=100, N=10)? What happens if you change the grid spacing (try slope ranges from 1-10 in steps of 0.1, y-int ranges from 1-10 in steps of 1)?

In Bayesian analysis, it is important to think about the questions you are trying to answer when setting up the problem. Starting with a well understood model grid and set of priors is key. In this case do you want a flat prior on the slope and y-intercept, or do you want a prior that compensates for the unequal distribution in angles? What range do you want to compute your data over? How finely should you bin the grids?

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). where time=0 is the present day (time = -14 is 14 days ago when you started taking data). Use your Bayesian skills now to perform this analysis.

a) Read in the data and plot it as % human vs. time in blue stars (Figure 1).

b) Set up grids for your parameter space assuming a set of linear models for the data. Based on your output from zombies1.py, what ranges should you try? Compute the posterior probability distribution. Which prior should you want to use?

c) Determine the marginalized posterior probability distribution for the percentage of humans today (time=0). Which parameter must you marginalize over? In Figure 2, plot the marginalized posterior distribution for the percentage of humans today (time=0) as red stars. What is the most likely % of humans alive today?

d) Using the prior that you yourself are not a zombie yet, recompute the posterior probability distribution and determine the marginalized posterior probability distribution for % of humans alive today (time =0). In Figure 2, over plot the new marginalized posterior probability distribution as green dots.

e) How does the Bayesian analysis for determining the % of humans today compare with the value from the MLE fit from the second activity?

For more information on the Bayesian approach, tutorials are available on the website below, which are all provided in python notebooks) http://jakevdp.github.io/blog/2014/06/14/frequentism-and-bayesianism-4-bayesian-in-python/ .

Optional - Activity 5:Mathematical Optimization

Mathematical optimization is finding the best option within a set of possibilities using a certain criterion (see https://en.wikipedia.org/wiki/Mathematical_optimization for more information). For instance in Least Squares Fitting, we have optimized the rms (a single number value), by determining the values of the slope and y-intercept that minimized the rms. In general, one could define any mathematical criterion for optimization, where optimization is defined by a variety of operations (although it is usually maximizing or minimizing the criterion). Read through optimizationTutorial.pdf, written by Zane Beckwith, (a former CAP REU graduate assistant). It provides an overview of the topic of optimization along with several different examples of different types of optimization algorithms.

At the end of the write up, Zane has a short tutorial for which he has provided 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. To complete the activity, you would use the function **minimize** from neldermead.py to find minima of the Himmelblau function, which is provided by the function **simulate** in scatter.py. I have provided my own example of how to do this in zanesprob.py.