DRAFT VERSION FEBRUARY 9, 2013
Preprint typeset using LaTeX style emulateapj v. 11/10/09

HW 1 SOLUTIONS: FITTING A STRAIGHT LINE WITH MCMC

Harvard University
Department of Astronomy
60 Garden Street MS 10
Cambridge, MA 02138
Draft version February 9, 2013

ABSTRACT

We use a Metropolis-Hastings Markov Chain Monte Carlo (MCMC) method to determine the best-fit parameters for a straight line to noisy data.

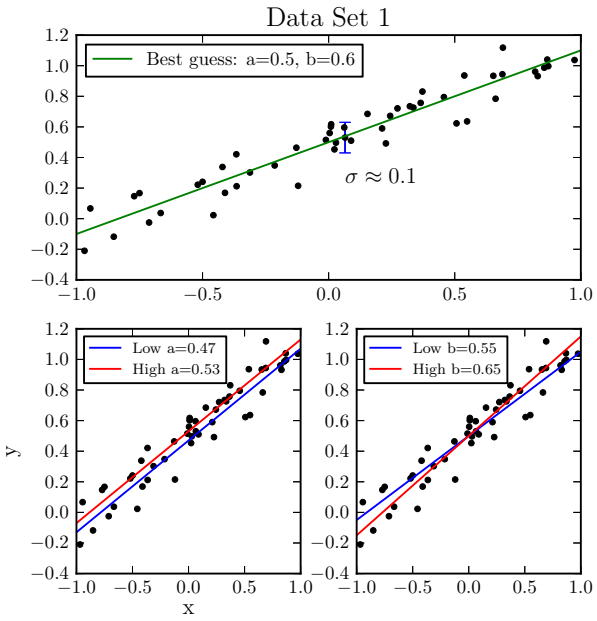

Figure 1. Parameter estimates done by eye. σ can be estimated from the scatter about the best fit line, done by eye. σ_a and σ_b can be estimated by varying a or b while keeping the other fixed, and finding the range of values for which the line still fits. Parameter estimates are shown in Table 1.

1. DATA AND PARAMETER ESTIMATES

Before writing any code[1] to fit the data, you should first plot the data to get an idea of what parameter space you are dealing with. Estimates of the parameters are made as shown in Figure 1 and listed in Table 1. We are also told from the handout that $\sigma_i = \sigma = 0.1$. The estimates of a and b (a_e and b_e) are made by simply playing around with a ruler on a printout of the data, varying the parameters until we arrive at what looks like a best fit. The estimate in σ (σ_e) is made by assuming the best fit line and then characterizing the typical scatter about this line. Estimates of σ_a and σ_b ($\sigma_{a,e}$ and $\sigma_{b,e}$) are made by keeping b fixed while varying a (or vice-versa) in order to determine the range of a (or b) values which would still give an acceptable fit to the data.

2. THE LIKELIHOOD OF THE MODEL

iczekala@cfa.harvard.edu

[1] All code used for analysis and plotting is available at https://github.com/iancze/AY193/tree/master/HW1

Table 1
Parameter Estimates

Parameter	Value
a_e	0.5
b_e	0.6
σ_e	0.1
$\sigma_{a,e}$	0.03
$\sigma_{b,e}$	0.05

Note. — Your own values may vary slightly, after all, they are estimates by eye.

As detailed in the handout, let x_i and y_i be our data, and the line model described as

$$f(p_n, x_i) = a + b x_i \tag{1}$$

where $p_n = \{a, b\}$.

We characterize the goodness of fit by the χ^2 statistic

$$\chi^2 = \sum_{i=1}^{N} \left[\frac{y_i - f(p_n, x_i)}{\sigma_i} \right]^2 \tag{2}$$

where in this problem we have assumed $\sigma_i = \sigma$. The "likelihood" of having a particular data set generated by the parameters is

$$\mathcal{L} = \exp(-\chi^2/2) \tag{3}$$

Calculating $\mathcal{L}$ for a given p_n is likely to be computationally intensive, since Eqn 2 involves squaring and summing operations for N data points. If we have a simple model and a moderate number of data points, it is sometimes quickest to just sample $\mathcal{L}$ over a grid of p_n, and by visual inspection find the maximum. We can of course use any other gradient optimization techniques to find the maximum. Even if the model is complicated, it might be worth it to sample the parameter space with a coarse grid, simply to get an idea of what you are dealing with beforehand. The $\mathcal{L}$ space of our problem is shown in Figure 2. This is the probability space which our Markov chain will sampling.

3. THE MCMC ALGORITHM

We sample the likelihood space (Figure 2) with the Markov Chain Monte Carlo (MCMC) algorithm. First, we assume some starting combination of parameters, $p_{\rm start} = \{a_{\rm start}, b_{\rm start}\}$.

In each iteration of the algorithm, we jump in the likelihood space

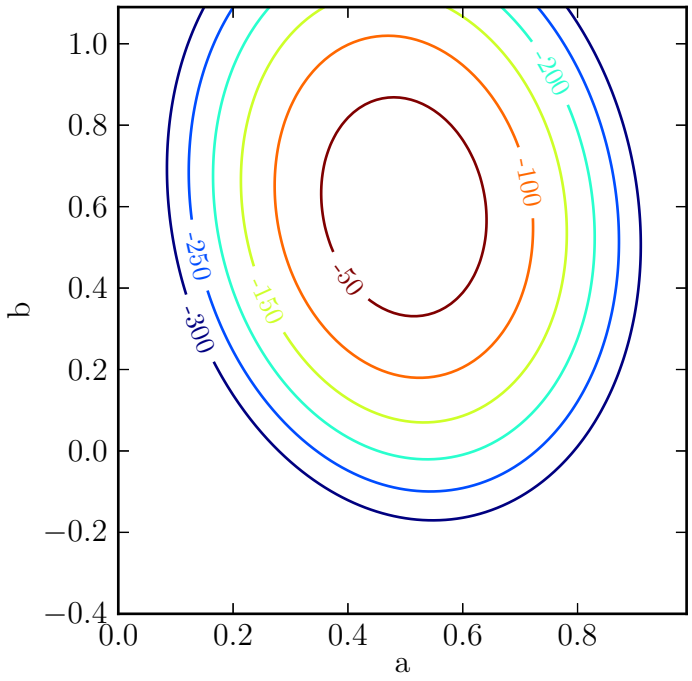

Figure 2. The $\mathcal{L}$ space sampled by a grid. Contours show $\log_{10}(\mathcal{L})$, and so many of the values are negative. Note how strongly peaked the probability distribution is around $a = 0.5, b = 0.6$. This likelihood space will be sampled by our Markov Chain Monte Carlo algorithm.

$$a_j = a_{j-1} + \Delta a_j \tag{4}$$
$$b_j = b_{j-1} + \Delta b_j \tag{5}$$

where Δa_j and Δb_j are drawn from Gaussian distributions

$$\Delta a_j \propto P_G(\mu = 0, \sigma = \sigma_{\Delta a}) \tag{6}$$
$$\Delta b_j \propto P_G(\mu = 0, \sigma = \sigma_{\Delta b}) \tag{7}$$

where P_G is a generic Gaussian distribution with mean μ and standard deviation σ. You may use a Gaussian random number generator within your code. For example, one `Python` routine is `np.random.normal`. We use our estimates in Table 1 for $\sigma_{a,e}$ and $\sigma_{b,e}$ to set the jump distribution width ($\sigma_{\Delta a}$ and $\sigma_{\Delta b}$) accordingly

$$\sigma_{\Delta a} \approx \sigma_{a,e} \tag{8}$$
$$\sigma_{\Delta b} \approx \sigma_{b,e} \tag{9}$$
$$\tag{10}$$

Later on in the problem set, we will vary $\sigma_{\Delta a}$ and $\sigma_{\Delta b}$ to asses their impact on the efficiency of the MCMC algorithm. After taking a jump (moving from $j-1$ to j), we compare the ratio of the likelihood (Eqn 3) at the new parameters to the likelihood at the old parameters

$$r_j = \frac{\exp(-\chi_j^2/2)}{\exp(-\chi_{j-1}^2/n)} = \exp(-(\chi_j^2 - \chi_{j-1}^2)/2) \tag{11}$$

Now, as in the class handout, calculate r_j and apply the Metropolis-Hastings algorithm:

1. if $r_j \geq 1$, then accept this step, because your proposed step has a higher (or equal) $\mathcal{L}$
2. if $r_j < 1$, then the likelihood of your proposed step is lower, so generate a uniform random variable, $u \propto P_U$, which has the range $[0, 1]$.
   (a) if $r_j \geq u$, accept the step, allowing you to climb out of local minima
   (b) if $r_j < u$, reject the step.
3. you should record a_j and b_j at the end of this step. If the new values were accepted, record them. Otherwise, record a duplicate of the previous values. Do not record the proposed values if they were rejected.

Table 2
Acceptance

Run #	$\Delta\sigma_a$	$\Delta\sigma_b$	Acceptance (%)
1	0.03	0.05	24
2	0.01	0.02	60
3	0.05	0.10	9

Note. —

Once the Markov Chain has settled down and bounced around the most likely parameters for a while (for me, this was $\sim$ 1000 steps for me), you can exit the algorithm. Congratulations, by saving your list of parameters you have generated a Markov chain. In later problem sets, we will discuss how to asses convergence of this chain in a more quantitative manner. If you are curious, check out the MCMC chapter in Gelman et al. (2004).

4. STUDYING YOUR MARKOV CHAIN

In every Markov chain, there is usually a period of "burn in," where the jumps in parameter space

Afterwards, discard the burn-in steps. These should be apparent as the steps that are much different from each other, kind of a trai

You should

This algorithm works because large jumps in χ^2 are possible, due to the fact that sometimes $u \approx 0$.

Because this calculation wasn't too time intensive, I calculated the $\mathcal{L}$ over a wide range of $\{a, b\}$ and plotted this is Figure 2. Normally, this would be computationally prohibitive for problems that would require MCMC, although it might be fine to calculate with a coarse grid.

[add class handout to website]

List of parameter steps is available in my online code, here.

Changing step size to vary acceptance ration

5. CHANGING THE STARTING ESTIMATES TO STUDY PARAMETER ESTIMATES

REFERENCES

Gelman, A., Carlin, J. B., Stern, H. S., & Rubin, D. B. 2004, Bayesian Data Analysis, 2nd edn., Texts in Statistical Science (Chapman & Hall/CRC)

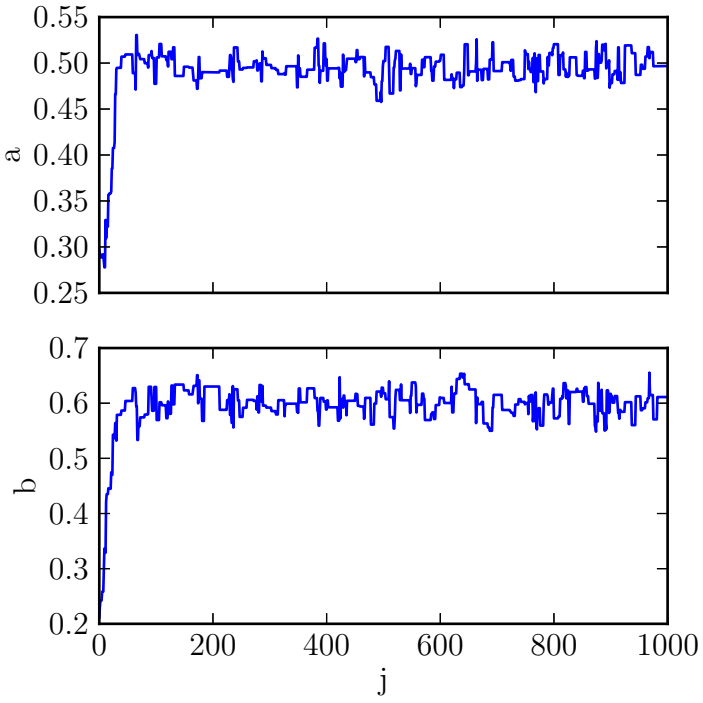

Figure 3.

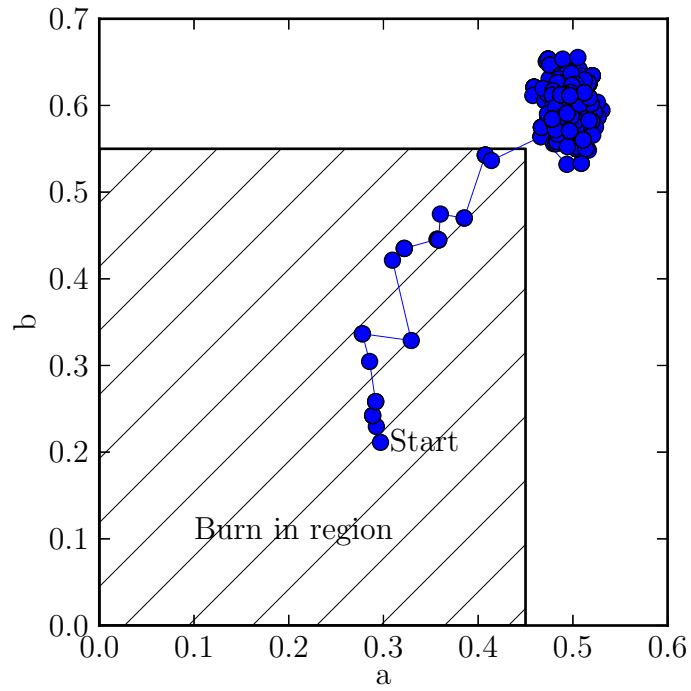

Figure 4.

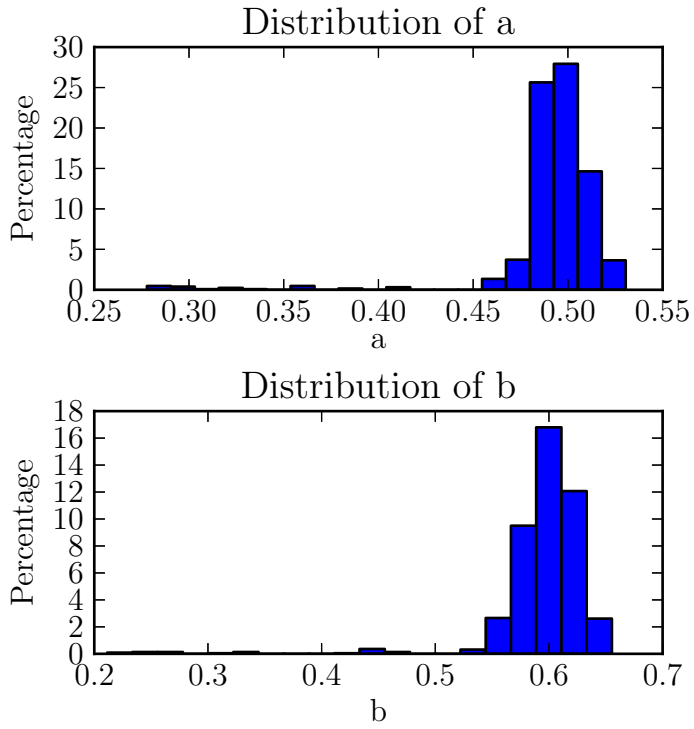

Figure 5.

Table 3
Starting

Run #	$a_{\rm start}$	$b_{\rm start}$	$a \pm \sigma_a$	$b \pm \sigma_b$
A	0.0	1.0	0.498±0.010	0.595±0.021
B	1.0	1.0	0.499±0.013	0.604±0.022
C	0.0	-0.3	0.499±0.009	0.604±0.020

Note. — All done with σ_a = 0.03, σ_b = 0.05. Runs in Figure

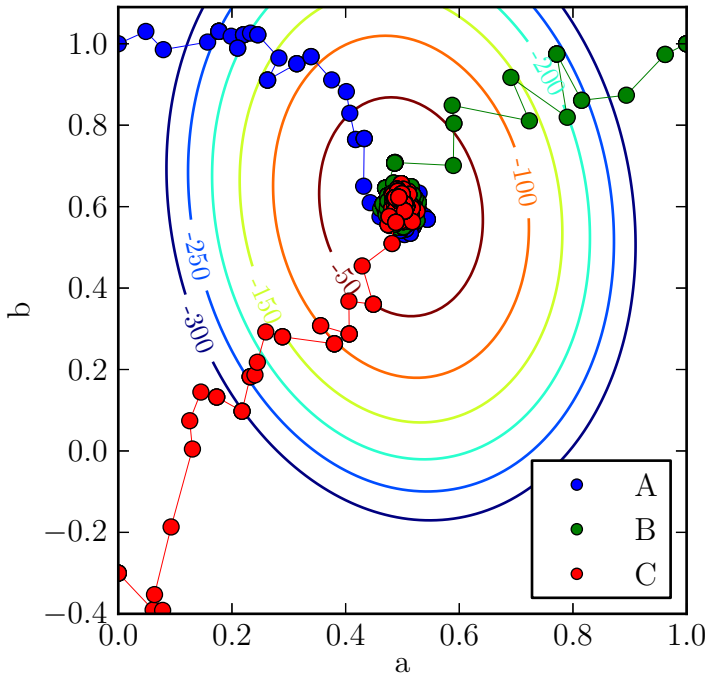
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
b
0.0
0.2
0.4
0.6
0.8
1.0
a
-200
-100
-250
-300
-150
-50
A
B
C

Figure 6.