Assignment: Marcov Chain Monte Carlo

Introduction

In this assignment, you have to fit a Gaussian PDF (probability density function). You have to find the standard deviation (std) σ and the mean value μ for a given dataset using the Metropolis-Hastings algorithm.

The data were generated individually for each student and are available for download at Moodle in the archive "Datasets.zip". The file "ListOfStudents.txt" in the archive contains the names of all students with the assigned number of the dataset to fit. The 1D-data are contained in the files named "Dataset*.txt". I have also plotted the PDFs of each dataset so that you can look at them.

The algorithm. A short summary of one step of the Metropolis-Hastings algorithm:

1. Begin with a plausible starting value for μ and σ. I recommend using the mean and std of your dataset obtained with numpy.mean and numpy.std $\pm$ a random fraction of them (so that you get a different starting point each time).

2. Generate a new proposal (new possible values) for μ and σ by taking the previous value and adding some random noise. The random noise has to be generated from a probability distribution. You should use a Gaussian distribution. You can use the function norm(parameter,sigma$_{mh}$).rvs() from the package scipy.stats to generate a new proposal. You can assume sigma$_{mh}$ = 0.1 to generate your new proposals for μ and σ (sigma$_{mh}$ determines the average size of an individual step).

3. Calculate the likelyhood for the the new proposal. For each datapoint p_i (your dataset contains 200 points), calculate:

$$L_i = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(p_i-\mu)^2}{2\sigma^2}\right) \tag{1}$$

   As you can see, this is simply the normal distribution. The likelyhood is the sum of logs of this array (i.e., just one number). This will give you *the probability of your dataset given your model*. The log is used to avoid very low numbers, but in principle you can also obtain the likelyhood by multipling all elements in the array (I recommend using the sum of logs though).

4. Compare the likelyhood of your set of μ and σ from the previous step and the new proposed values. If the proposal is more likely, accept it as a new value. If it is less likely, it can still be accepted with a probability proportional to the ratio of the likelyhoods of the old value and the proposal. In terms of programming, this can be implemented as follows: use your likelyhoods obtained in the previous step to calculate the number $\alpha = \exp(\text{likelyhood}_{\text{proposal}} - \text{likelyhood}_{\text{old}})$. Then, draw a random number r from 0 to 1 and compare it with α. If $\alpha > r$, accept the proposal for μ and σ. Otherwise reject the

proposal and accept the old values as your new set (i.e., your new set of values repeats the old).

5. Since you sometimes accept less probable values, the algorithm allows you to draw from the whole distribution.

Assignment:

1. Read the dataset from the file into your program. Calculate the mean and the standard deviation of your data (you can use numpy.mean and numpy.std from the package numpy).

2. Generate the starting values of μ and σ for your algorithm as described above. NB: don't forget to check that your initial σ is larger than 0. Calculate the likelyhood of your initial starting set.

3. Implement the Metropolis-Hastings algorithm to obtain the new proposal for μ and σ. You can assume a standard deviation of the distribution of your random walk of 0.1 (you can also test different values).

4. Calculate the likelyhoods for the new proposal and the old value (initial value or the value of the previous step). Accept or reject it as described above.

5. Repeat the step multiple times (several thousand). You should see that your values of σ and μ converge towards the ones calculated with numpy.mean and numpy.std. Check at every step if your proposal for σ is larger than 0. If it is not, do not accept it and draw from the distribution again (otherwise you will get NaNs or other errors).

6. Test different starting values of μ and σ that are far from the ones estimated with numpy.mean and numpy.std. You should see the burn-in phase of your algorithm during which the values will converge to the real mean and std.

Plot the results. The plots you have to produce:

1. The Marcov chain for the values of μ

2. The Marcov chain for the values of σ

3. The plot μ versus σ (you will be able to see the parameter space the algorithm explores)

4. The histogram of the original data (normalize it using "normed=True" in plt.hist) plus the PDF of your best fit of μ and σ (you can plot it on top of the histogram using the function norm.pdf() from the package scipy.stats).

What to be careful of: if your initial guess for μ and σ is too far from the real values, the algorithm may never converge. If your sigma$_{mh}$ (the width of the distribution to generate proposals) is too small, it will take it a long time to converge. If sigma$_{mh}$ is too large, the precision of your algorithm will be very low. I've tested the value of 0.1 and it works well for all datasets.

Output should look like that (but you will have different datasets and starting points):