**Sofía García del Barrio Cervera**

**FCBB - HW 8**

**04/08/2022**

**MCMC model for CpG island characterization of Martian DNA**

**Part 1: R-code writeup**

1. **Getting Started**

The different modules of the Metropolis.R function are:

* Generate the data: simulate the number of CpG sequences in 100 random Martial genomes samples of average length 50. The approximate proportion of samples corresponding to CpG islands is 20%.
* Logpost function: if the probability and proportion values in the theta vector are correct (between 0 and 1), it calculates the posterior theta given a prior. It computes the sum of the logarithm of the weight average likelihood function for both CpG and non-CpG islands.
* Proposal function: generate a new prior with a small jump from the previous prior that we had. The jump is randomly generated with a standard normal distribution.
* Set starting values: set the initial values for the theta vector and generate an empty MCMC vector with the desired length.
* Construct the Markov chain: for each row in the MCMC vector, generate a new theta and calculate the posterior for the new theta and the previous one. If the difference between both is small enough, the new theta is accepted and the value is added to the Markov chain. If not, the previous one is used. It adds 1 to the acceptance count every time a new theta satisfies the condition.
* Visualize: plots all the values of theta in three different graphs and prints the average value of each parameter and acceptance ratio.

The elements of the theta vector are . The meaning of each is:

* is the proportion of CpG islands in the Martian chromosome 2, which is the probability of a reading being a CpG island.
* is the proportion of CpGs in CpG islands, which is the probability of a sequence being CpG if it is in a CpG island.
* is the proportion of CpGs in non-CpG islands, which is the probability of a sequence being CpG if it is not in a CpG island.

because the probability of finding a CpG sequence is higher in a CpG island.

1. **Initialization**

The initial values of the theta vector are . The length of the MCMC chain is

The correct values used to simulate *y* are (. The same values are used as initial parameters for the MCMC chain and for the simulation of *y* to create the data.

1. **Default run**

The acceptance ratio and mean values printed are:

* Acceptance Ratio: 0.439666666666667
* Lambda Mean: 0.178874118500302
* p1 Mean: 0.581191449487045
* p2 Mean: 0.103209377113332

The correct values are . The estimation of p2 is pretty accurate, while p1 and lambda are a bit further, especially lambda. However, in general we could say that the estimation is good.

1. **MCMC performance**

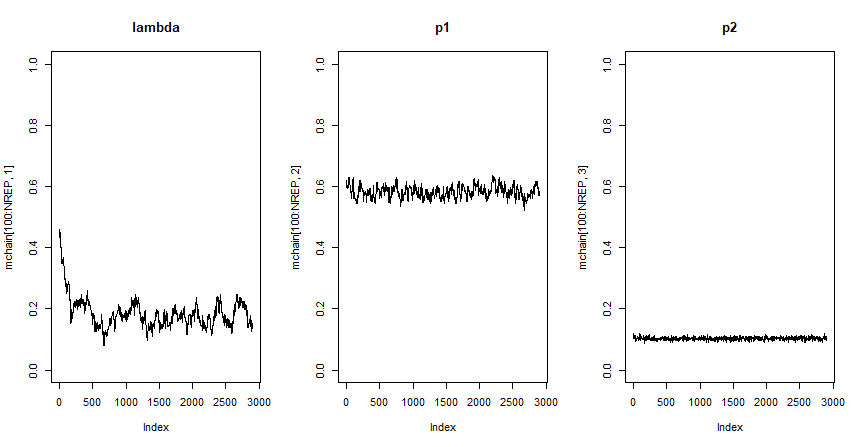
The new parameters used are . I have tried to change as much as possible to see a difference but keeping the logical relation between the three (all of them between 0 and 1, and p1>p2) The acceptance ratio and mean obtained are:

* Acceptance Ratio: 0.428
* Lambda Mean: 0.184127805506899
* p1 Mean: 0.58240982623228
* p2 Mean: 0.103198587336914

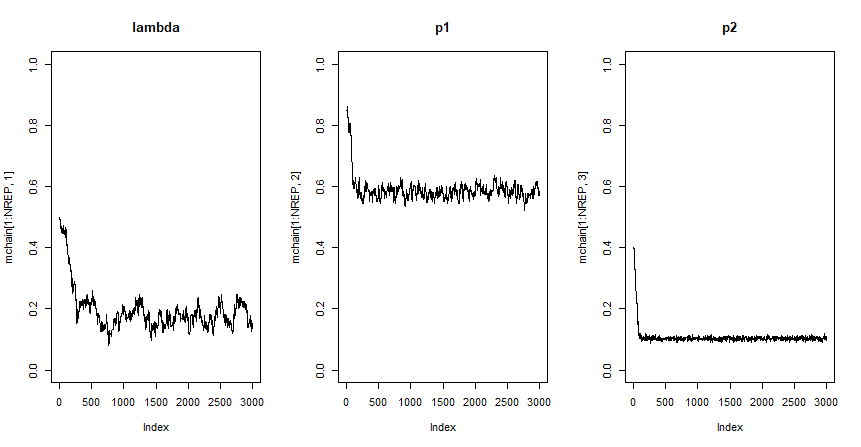
The results are very similar to the previous running. Lambda is slightly closer to the real value, but I think it is because I start from a higher value than increases the average. The estimation is good.

1. **Burn-in**

I keep the same values as in the last running, The plot excluding the first 100 rows is:



And the plot with all the rows:



When we start from initial values far from the correct ones, the first iterations are used to approach the correct values. For this reason, in the second plot we see a steep decrease (could also be an increase if the initial conditions were lower) in the estimation of the parameter until it reaches the correct value. From then, the results are relatively constant, and the correct value can be estimated from the plot. Excluding the first 100 rows from the plot of p1 and p2 avoids showing the burn-in region, and only shows the estimation that will be closer to the real values. The plot for lambda has a larger burn-in period that can still be seen in the second graph. We could start from repetition 200 to avoid it.

1. **Proposal (jumping) distribution**

The normal function parameters are: *rnormal(n, mean, standard deviation).* The default is *rnormal(n, 0, 1),* which is the standard normal distribution.

Making the standard deviation 0.5 gives the following result and plot:

Chart, histogram

Description automatically generated

Acceptance Ratio: 0.639

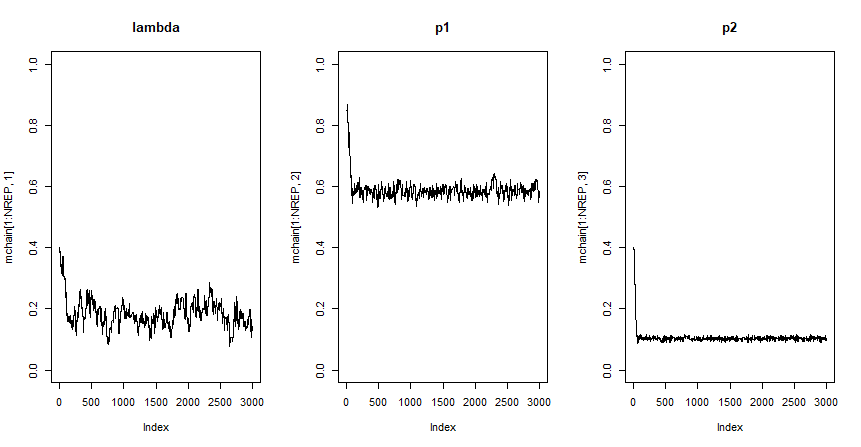
The jump for the new prior (theta) is closer to 0, so the difference between the previous and candidate theta is small and the condition

*alpha = logpost(thetaCandidate)-logpost(theta)*

*if( runif(1) <= exp(alpha) ) {*

will nearly always be satisfied, giving a high acceptance ratio. The new row in the posterior Markov chain will have a similar value to the previous one, so changes in the distribution are smaller and approaching the correct value will take longer, giving a longer burn-in period (especially for lambda). The three estimations approach the correct values.

Making the standard deviation 1.7 gives the following result and plot:



Acceptance Ratio: 0.273

The jump between consecutive priors oscillates more around zero due to the larger standard deviation, so there is higher change between consecutive priors and the acceptance ratio is lower. There are bigger jumps in the posterior distribution, which allows the Markov chain to approach the correct value faster, resulting in a shorter burn-in period. The three estimations also approach the real value.

As a summary:

* Higher standard deviation: shorter burn-in period, lower acceptance ratio. The jumps between consecutive posteriors in the Markov chain are bigger.
* Lower standard deviation: longer burn-in period, higher acceptance ratio. The jumps between consecutive posteriors in the Markov chain are smaller.