STAT 431 Final Project: Metropolis-Hastings Sampling Algorithm Research Paper

Mihir Bhave

Minwoo Sung

Yeonchul Choi

STAT 431: Applied Bayesian Analysis

Professor Park

December 15th, 2021

Abstract

We chose the Metropolis-Hastings Sampling Algorithm for our research. We study what the Metropolis-Hastings Sampling Algorithm is and how it works. We would first create samples of distributions that we learned with certain parameters and will try to sample those parameters. By comparing distributions and Metropolis-Hastings Sampling Algorithm we will find parameters of the distributions with plots by backtracking.

Introduction

The goal of this project is to illustrate the Metropolis-Hastings Sampling Algorithm with different distributions which are hard to directly sample. We'll find different examples of distributions to review and analyze the algorithm. The Metropolis-Hastings algorithm is one of the methods of the Markov Chain Monte Carlo method and it is usually used for multi-dimensional distributions. Its goal is to sample and approximate from a certain distribution that is difficult for direct sampling. The Metropolis-Hastings algorithm works in the following order :

An arbitrary point x(0), which is the initial position of x, is needed to start the sampling process. Arbitrary probability density Q(x'(t+1)|x(t)), which is the conditional probability of x' being in (t+1) given x is at t, will be used to find out the next accepted sample. Q is called proposal density. For the Metropolis Algorithm, which is a special case of the Metropolis-Hastings Algorithm, Q is assumed to be symmetric which is same as Q(x(t+1)|x(t)) = Q(x(t)|x(t+1)) (Chib & Greeberg, 1995). Using this proposal density, candidate x' is generated and the acceptance ratio is calculated for each candidate to see if that candidate is acceptable for the next sample. If it is acceptable, then x'(t+1) would become the next sample, x(t+1). The algorithm proceeds by randomly accepting the samples and randomly moving spaces and ends up staying in high-density regions which leads to the targeting distribution. The sequence from this method can be used in estimating the distribution or computing the integral. This approach can take samples from any symmetric probability distribution as long as the value of the function f(x) can be determined.

It is evident that the Metropolis-Hastings Algorithm is extremely useful, but some potential shortcomings need to be noted. First off, the algorithm relies heavily on the initial values, since the first value sets the path for the rest of the algorithm to succeed. If the first value in the chain is not adequate, it can throw off the rest of the algorithm. One way to resolve this is by doing a burn-in, which means removing a set number of values at the beginning of the chain. We can generate the number of iterations to burn by running various tests such as the Gelman Diagnostic plot, which gives us a general range of where to remove or burn values. We will employ this method later on in our paper. Furthermore, another issue that can arise is autocorrelation. To resolve this, we can use a concept called thinning, which means selecting a specific nth value or values in the iteration to keep or discard. For example, we can double the number of simulations and set the thinning factor to two, which means we will discard every other value. Using a specified thinning algorithm can help reduce autocorrelation and provide a more accurate reading and overall model.

Method

The Metropolis-Hasting algorithm was implemented using R. In order to create samples from certain distributions with specified parameters, 1000 random samples for each distribution were created using R code.

```
a = 2
b=.4
x <- rgamma(1000,a,b)
hist(x,30)
```

```
a = 5
b= 1
x <- rbeta(1000,a,b)
hist(x,30)
```

Figure 1

As shown in Figure 1, Gamma and Beta distributions were chosen for this research. For the gamma distribution, alpha was set to 2, and beta was set to 0.4. Meanwhile, for beta distribution, alpha was set to 5, and beta was set to 1.

```r
#chain1
theta0 <- c(1, 1)
prop_sd <- c(0.1, 0.1)
its <- 10000
burn <- 0
chain1 <- metropHastingsMCMC(theta0, prop_sd, its, burn)
```

```r
#chain2
theta0 <- c(0.1, 0.1)
prop_sd <- c(0.1, 0.1)
its <- 10000
burn <- 0
chain2 <- metropHastingsMCMC(theta0, prop_sd, its, burn)
```

```r
#chain3
theta0 <- c(0.01, 0.01)
prop_sd <- c(0.01, 0.01)
its <- 10000
burn <- 0
chain3 <- metropHastingsMCMC(theta0, prop_sd, its, burn)
```

Figure 2

As shown in figure 2 above, 3 chains in total were used with different sets of initial values of parameters and standard deviations. For each iteration, new candidates for the parameters should be sampled, and the Metropolis-Hastings algorithm uses a distribution that is easy to sample with. In order to sample the candidates of the new parameters, the Normal distribution was chosen as the prior distribution. To find out if the newly proposed candidates were appropriate, we needed to calculate proposal density. The following formula was used to find the proposal density :

Q = f(new_candidate) * g(current_parameters | new_candidate) / f(current_parameters) * g(new_candidate| current_parameters)

In this formula, f in the numerator was the likelihood of being in new_candidate, and g in the numerator was the probability of being at the proposed state of new_candidate given the current_state of parameters. This proposal density was then compared with the acceptance ratio which was randomly assigned by the Uniform distribution from 0 to 1. If the proposal density was greater than the acceptance ratio, then the proposed candidate became the next sample of the parameters. If the density was lesser, we

used the previous sample until we got a new proposal density. This calculation was done for each iteration. The samples in which we were able to get the proposed density above the acceptance rate were included in the final samples.

After getting the final samples, we had to do a burn-in, or removal of a certain number of samples or iterations. To determine the burn-in period or the number of iterations removed from the beginning of the chain, we performed 10000 iterations initially. We then ran a Delman Plot to determine how many values to burn-in. Figure 3 below demonstrates this plot and we can see that at about 2000 iterations for the gamma distribution, the plot flattens out, so this is our cutoff number for the burn-in. With the Gelman plots, the burn-in period could be found for each distribution.

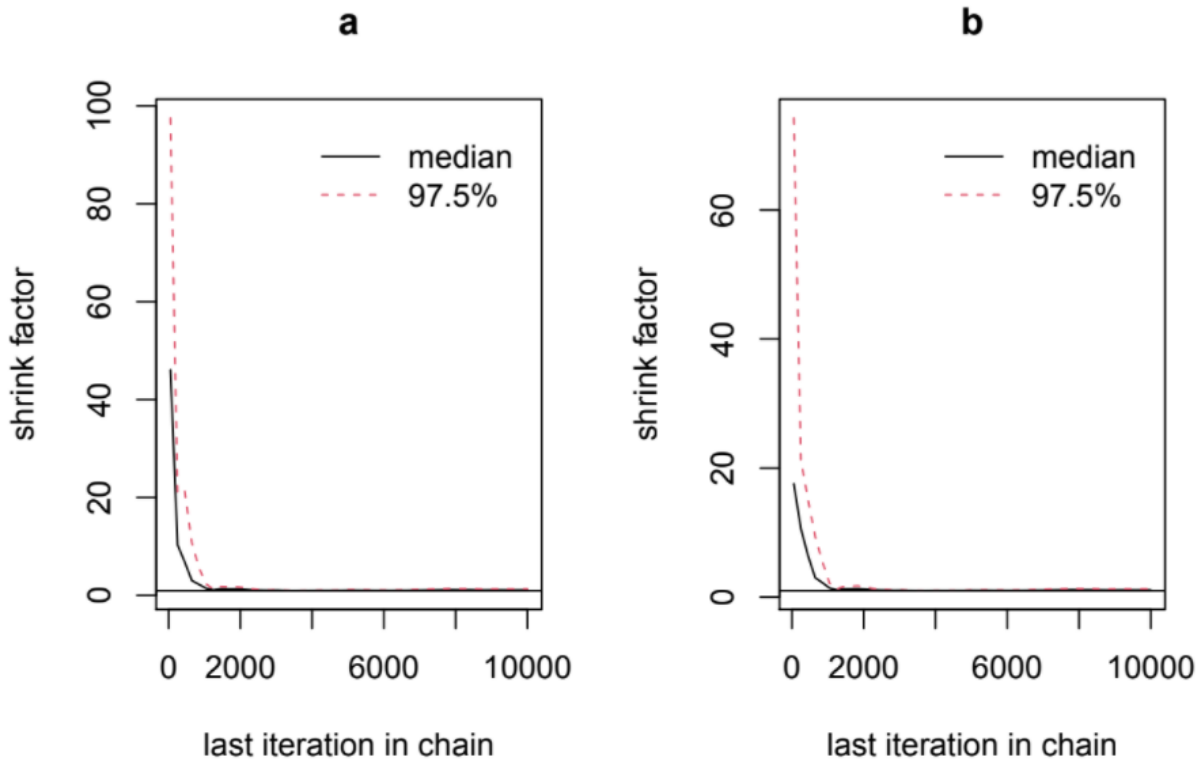

Figure 3

To find the burn-in period for the Beta distribution and the number of iterations to remove, we performed the same algorithm with the Beta distribution and used a similar process with the Delman Plot. Figure 4 below demonstrates this plot and we can see that at about 5000 iterations for the beta distribution, the plot flattens out, so this is our cutoff number for the burn-in. Afterward, we can run the Delman diagnostics for convergence, which we will further elaborate upon in the results section.

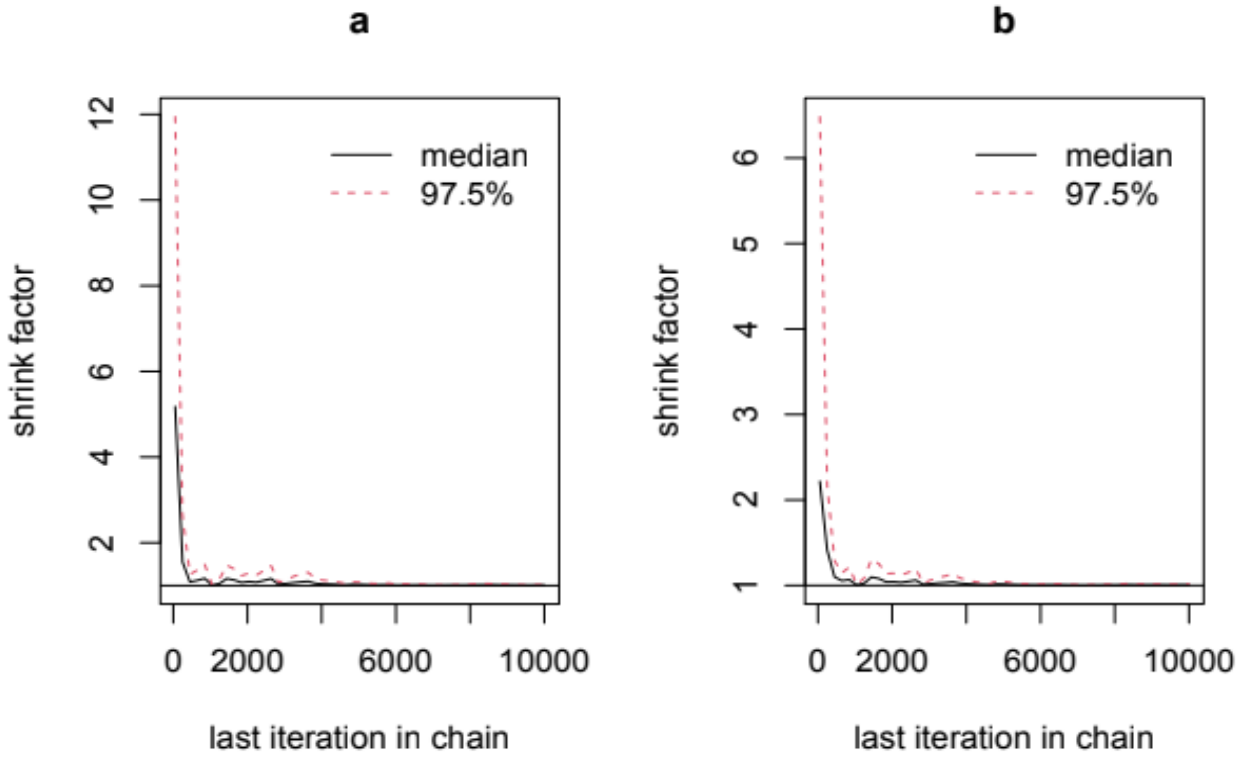

Figure 4

Results

Our burn-in period was 2000 iterations, so we set our new chains with the burn-in period and used our algorithm again to predict the alpha and beta of the original distribution.

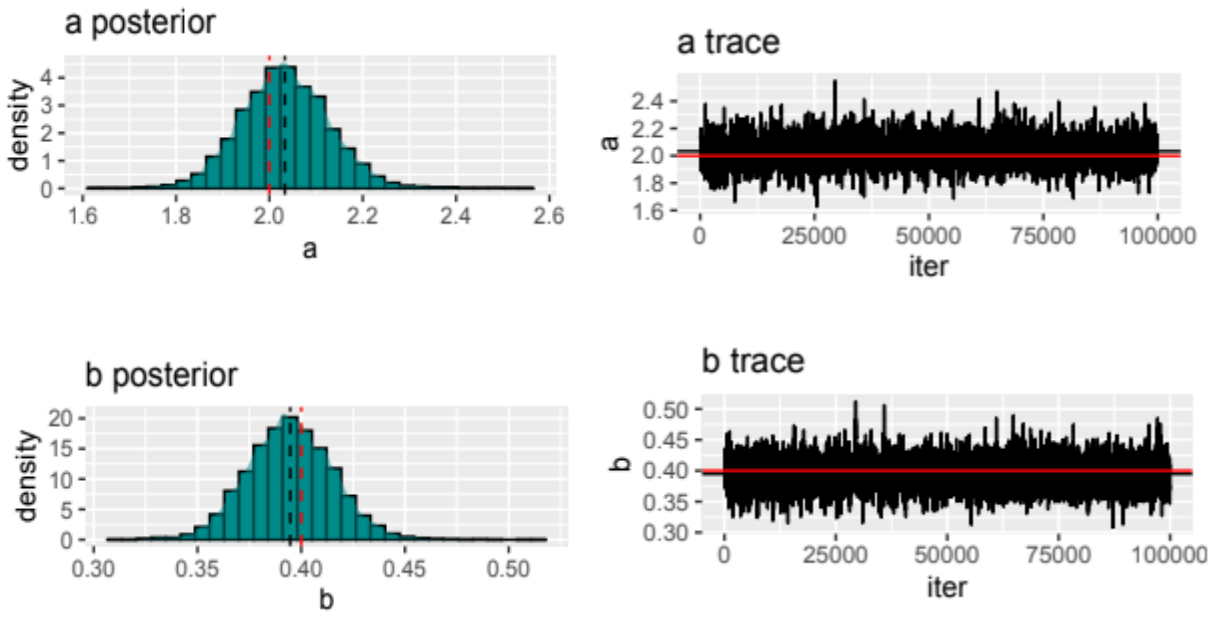

Figure 5

Figure 5 shows the prediction of the alpha and beta of Gamma distribution by using chain 1.

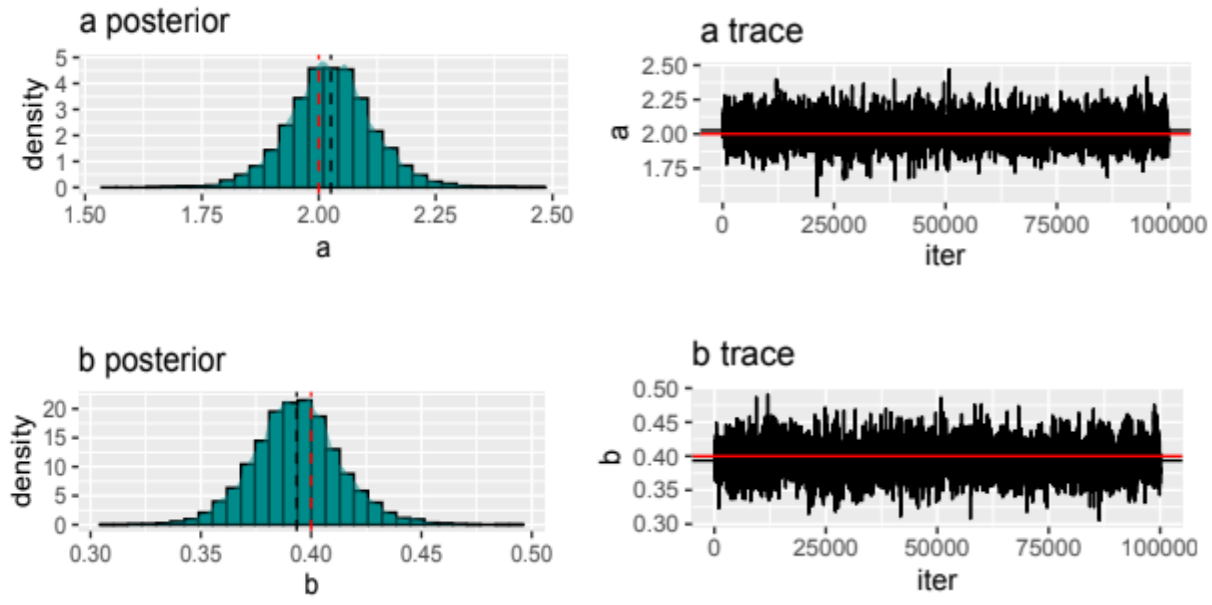

Figure 6

Figure 6 shows the prediction of the alpha and beta of Gamma distribution by using chain 2.

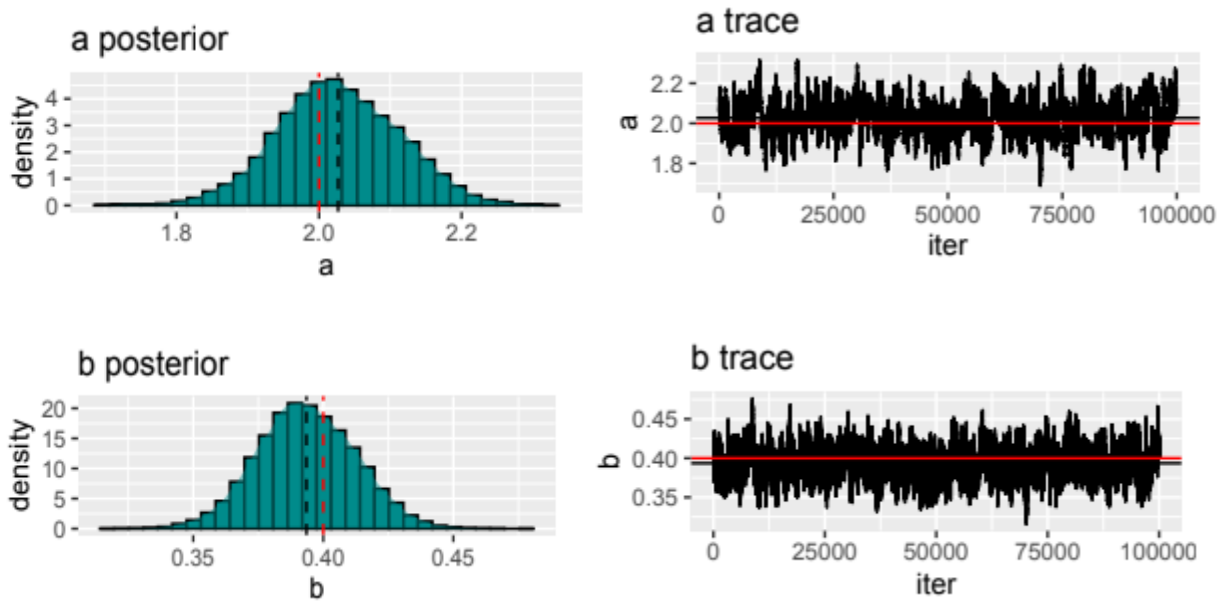

Figure 7

Figure 7 shows the prediction of the alpha and beta of Gamma distribution by using chain 3.

The red lines in the figure represent the value of the alpha and beta of the original distribution. All results with 3 different chains show the same prediction with 2.0 for alpha and 0.4 for beta.

```
## Potential scale reduction factors:
##
##   Point est. Upper C.I.
## a          1          1
## b          1          1
##
## Multivariate psrf
##
## 1
```

Figure 8

After applying the burn-in period, Figure 8 shows that the point estimation of both alpha and beta are 1, so we can conclude that all chains have converged.

For beta distribution with 5000 iterations found above, we performed an algorithm again as what we did for gamma distribution to predict values of alpha and beta.

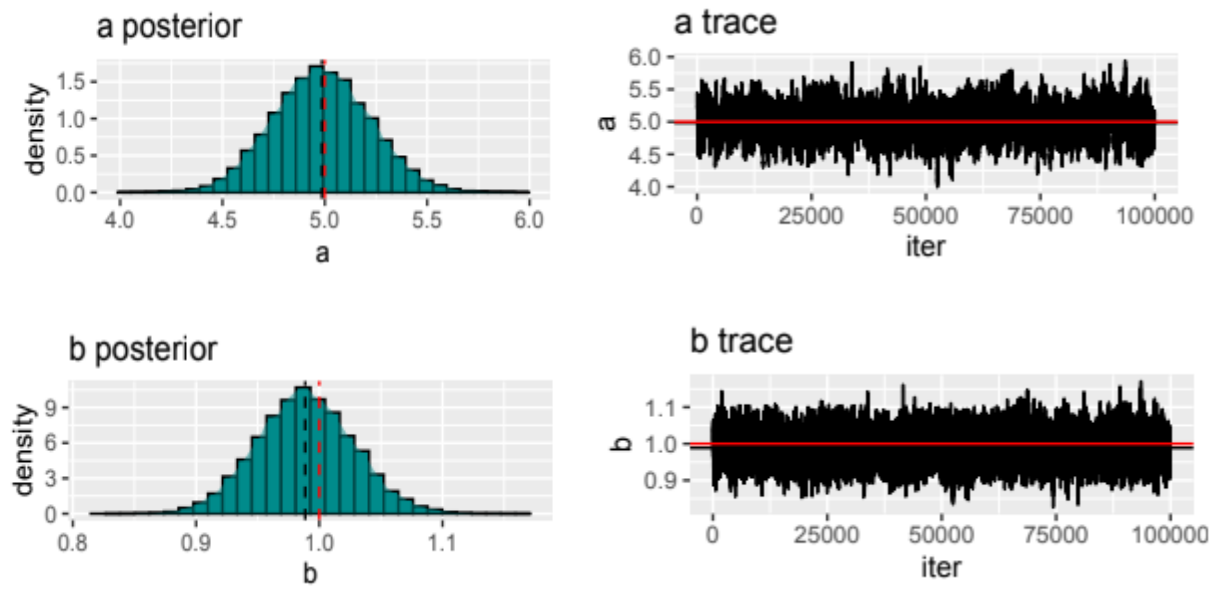

Figure 9

Figure 9 shows the prediction of the alpha and beta of Beta distribution by using chain 1.

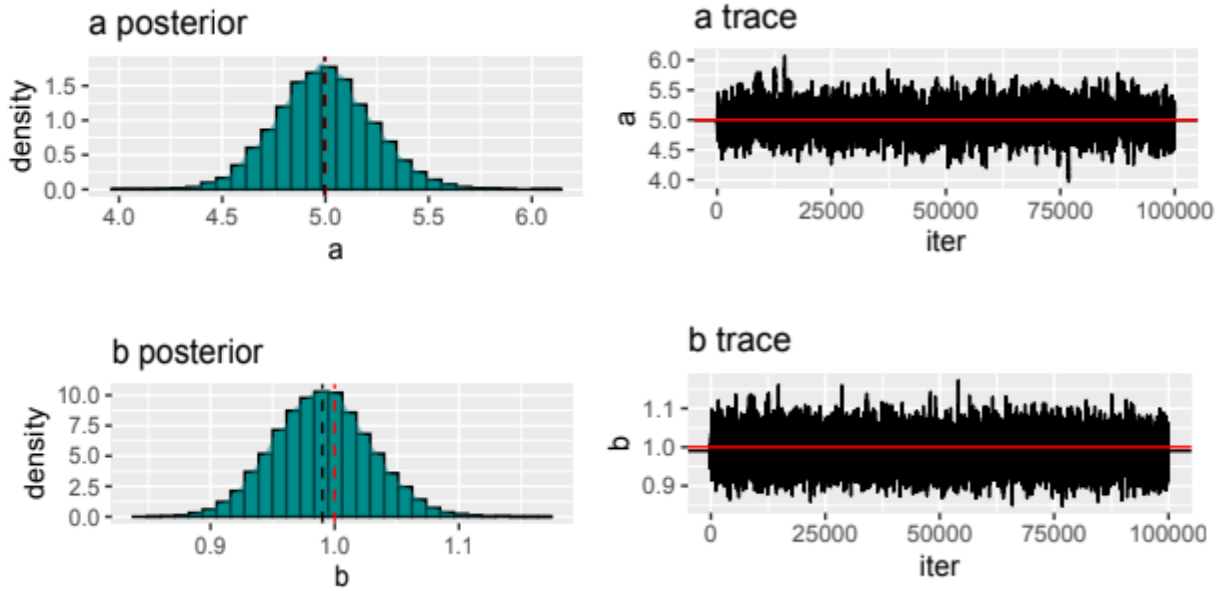

Figure 10

Figure 10 shows the prediction of the alpha and beta of Beta distribution by using chain 2.

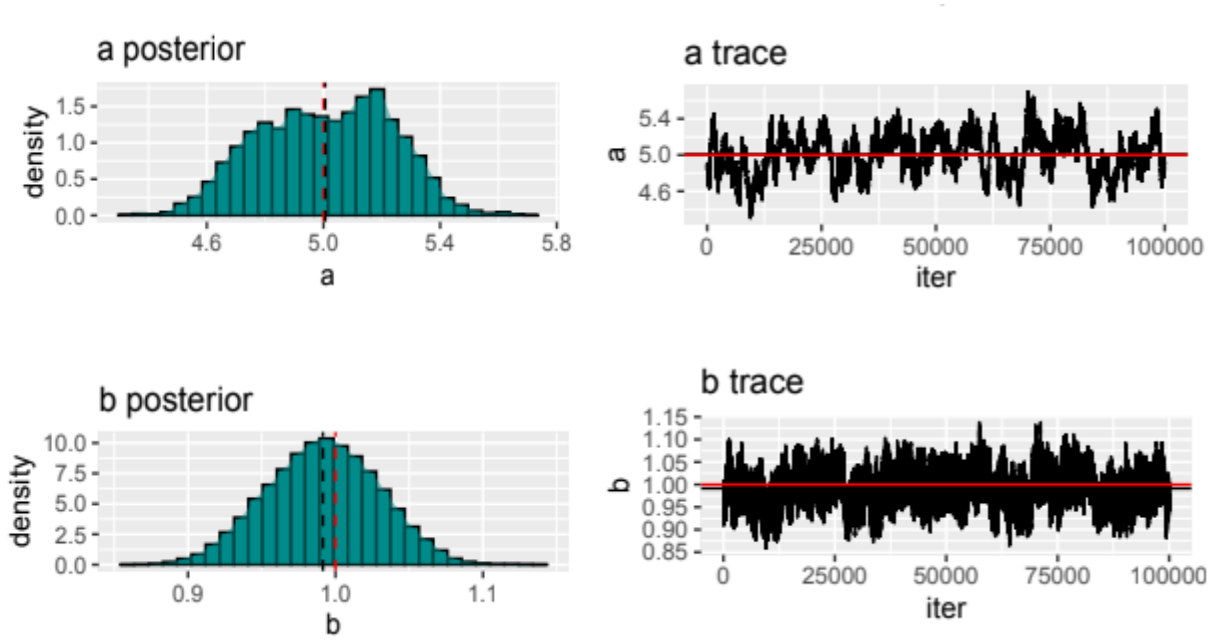

Figure 11

Figure 11 shows the prediction of the alpha and beta of Beta distribution by using chain 3

The red lines in the figure represent the value of the alpha and beta of the original distribution. All results with 3 different chains show the same prediction with 5.0 for alpha and 1.0 for beta.

```
## Potential scale reduction factors:
##
##   Point est. Upper C.I.
## a       1.01       1.02
## b       1.00       1.02
##
## Multivariate psrf
##
## 1.01
```

Figure 12

As shown in Figure 12, same as Gamma distribution, the point estimation of both alpha and beta in Beta distribution are below 1.1, so we can conclude that all chains have converged.

As a result, we find that by using Metropolis-Hastings Algorithm we can predict the parameters of the original distribution. With sample datasets and backtracking them, we can predict alpha and beta for both Gamma and Beta distribution and results are almost exactly the same as the original.

Discussion

We can see the usefulness of the Metropolis-Hastings Algorithm and how it can be used to predict parameters of the original distribution. In our case, we were able to use the normal distribution as our prior distribution and sample it to see if the candidates of the new parameters were appropriate. This approach can be used for different types of distributions in the future, including Inverse Gamma, Poisson, Binomial, Negative Binomial, and many more. Metropolis-Hastings will allow us to produce samples from distributions that can otherwise be difficult to sample from and is an extremely powerful test for sampling in Statistics. It provides a new outlet for sampling beyond the standard Markov Chain Monte Carlo method and has a wide range of useful applications that can be employed. It is a truly versatile algorithm.

References

Chib, S., & Greenberg, E. (1995). Understanding the Metropolis-Hastings Algorithm. In *The American Statistician* (4th ed., Vol. 49, pp. 327–335). story, American Statistical Assoc.

Yildirim, I. (2012). (rep.). *Bayesian Inference: Metropolis-Hastings Sampling*. Rochester, NY.

Burke, N. (2018). (rep.). *Metropolis, Metropolis-Hastings and Gibbs Sampling Algorithms* (pp. 21–26). Thunder Bay, Ontario.

Chib, S., & Greenberg, E. (2012). (rep.). Understanding the Metropolis-Hastings Algorithm.

Peng, R. D. (2021, January 26). Advanced statistical computing. 7.2 Metropolis-Hastings. Retrieved December 16, 2021, from https://bookdown.org/rdpeng/advstatcomp/metropolis-hastings.html#random-walk-metropolis-hastings

Appendix

For this project, Minwoo worked on running the algorithm in R and wrote the initial part of the method section. Mihir wrote about the shortcomings in the introduction, part of the methods section, and the discussion. Yeonchul worked on the abstract, the initial part of the introduction, and the results section.