

# Project Report

Pramoda Jayasinghe

April 23, 2021

## 1 Introduction

Statistical models are used in many practical applications to quantitatively understand an underlying process and to predict the dynamics of the process under new data. The parameters in such models are usually unknown and need to be estimated. Estimating these parameters provides insight into the way that the process of interest behaves (Ballnus et al., 2017).

The parameter estimation is usually done using either a frequentist or a Bayesian approach. Frequentist methods usually employ an optimization of a loss function or a likelihood function. The uncertainty of the estimates is usually obtained theoretically using asymptotic distributions or using techniques such as bootstrapping. Bayesian methods often involve sampling parameters from a defined posterior distribution, using a Markov chain Monte Carlo (MCMC) method. Different tools used in either the frequentist setting or the Bayesian setting have their advantages and disadvantages.

The literature exists for many comparisons and benchmarks of frequentist optimization tools (Villaverde et al., 2015) but the same cannot be said for benchmarking MCMC methods in the Bayesian setting (Ballnus et al., 2017). Considering the popularity of MCMC methods, especially over the last few decades, it is important to explore the performance of these methods with different Bayesian problems.

This project aims to examine the “performance” of 2 main MCMC algorithms, namely Metropolis-Hastings and Parallel Tempering. To achieve this aim, there are 2 objectives defined for this project.

- Implement MH and PT algorithms in R.
- Evaluate the “performance” of these algorithms under varying difficulties of problems.

## 2 Methods

### 2.1 MCMC Algorithms

The two algorithms considered in this study are *Metropolis-Hastings* (MH) and *Parallel Tempering* (PT). The MH algorithm (Algorithm 1) was selected as a simple algorithm that is easy to implement with comparatively lower computational costs. An advantage of this algorithm is that it can use a function proportional to the posterior as calculating the necessary normalization factor is often extremely difficult in practice (Robert and Casella, 2010).

Single-chain algorithms such as MH might have trouble working with different posterior modes, especially if they are separated by areas of low probability density. The PT algorithm samples from multiple versions of the posterior with different *annealing parameters*; ( $\beta_0 = 0, \dots, \beta_N = 1$ ) in parallel (Algorithm 2). The “landscape” corresponding to  $\beta_0 = 0$  is easy to explore since it would be sampling from the prior. On the other hand, the landscape of interest corresponds to  $\beta_N = 1$ , which is the posterior.

For this project, these algorithms are implemented in R. Please see Appendix A for further details.

### 2.2 Effective sample size (ESS)

Given the number of iterations in a Markov chain, ESS measures the size of an independent and identically distributed (i.i.d.) sample with the same standard error (Gong and Flegal, 2016). Therefore having a higher ESS, compared to the number of MCMC samples is a desirable property of the algorithm. To estimate the ESS, `mcmcse::ess()` function was used which employs a batch means estimator. The estimate is given by

$$\widehat{ESS} = n \frac{\lambda^2}{\sigma^2},$$

where  $\lambda^2$  is the sample variance and  $\sigma^2$  is an estimate of the variance in the CLT (Flegal and Jones, 2010).

---

**Algorithm 1:** Metropolis-Hastings

---

**Input:** Iterations:  $N_{iter}$ , Unnormalized density:  $\gamma(x)$ , Proposal:  $q(x | y)$ , Initial state:  $x^{(0)}$

**Result:** Approximated posterior sample

**for**  $i \leftarrow 0$  **to**  $N_{iter}$  **do**

1. Generate  $x^* \sim q(y | x^{(0)})$
2. Compute the MH ratio

$$r = \frac{\gamma(x^*)}{\gamma(x^{(i)})} \frac{q(x^{(i)} | x^*)}{q(x^* | x^{(i)})}$$

3. Simulate an acceptance variable

$$a \sim \text{Bern}(\min(1, r))$$

- 4.

**if**  $a = 1$  **then**

  |  $x^{(i+1)} = x^*$

**else**

  |  $x^{(i+1)} = x^{(i)}$

**end**

**end**

---

---

**Algorithm 2:** Parallel tempering

---

**Input:** Iterations:  $N_{iter}$ , Initial state:  $x^{(0)}$ , Annealing parameters:  $(\beta_0 = 0, \dots, \beta_N = 1)$ , Landscapes:

$$\pi_j(x) := \pi_{\beta_j}(x)$$

**Result:** Approximated posterior sample

**for**  $i \leftarrow 0$  **to**  $N_{iter}$  **do**

**for**  $j \leftarrow 0$  **to**  $N$  **do**

  | Generate  $x_j^*$  using MH with landscape  $\pi_j(x)$

**end**

Swap between neighbouring chains based on the Metropolis ratio

$$\frac{\pi_j(x_{j+1}^{(i)})\pi_{j+1}(x_j^{(i)})}{\pi_j(x_j^{(i)})\pi_{j+1}(x_{j+1}^{(i)})}$$

**end**

---

### 3 Simulation studies

The performance of MH and PT algorithms is evaluated under two examples.

- Beta-Binomial problem with varying prior parameters and a varying number of iterations.
- Mixture distribution problem with data generated from varying mixture parameters.

The “performance” of the algorithm is evaluated under the running time, effective sample size and effective sample size per second of running time.

All these algorithms were run as a serial program on an Intel Core i7-8750H running Windows 10.

#### 3.1 Beta-Binomial

Consider a Binomial random variable from  $\text{Bin}(n, p)$  and a prior on  $p \sim \text{Beta}(\alpha, \beta)$ . If the data shows  $k$  successes, then the posterior can be expressed as  $\text{Beta}(\alpha + k, \beta + (n - k))$ .

This example aims to change the prior;  $(\alpha, \beta)$  and examine the performance characteristics mentioned above. For this example, a special case  $\alpha = \beta$  was selected while increasing these would make the prior more peaked eventually making the sampling process harder.

Throughout the simulation,  $n$  was selected to be 3 and  $k$  was set to 0. These values were selected to keep the problem simple and to make the likelihood skew to the right. 50 values were used for  $\alpha = \beta$  in  $(1, 3, 5, \dots, 99)$  to ensure that the prior would take forms from a Uniform distribution to a highly peaked distribution. Generated MH samples, theoretical posteriors and the prior for hyper-parameters  $\alpha = \beta = 1$  and  $99$  are shown in Figure 1.

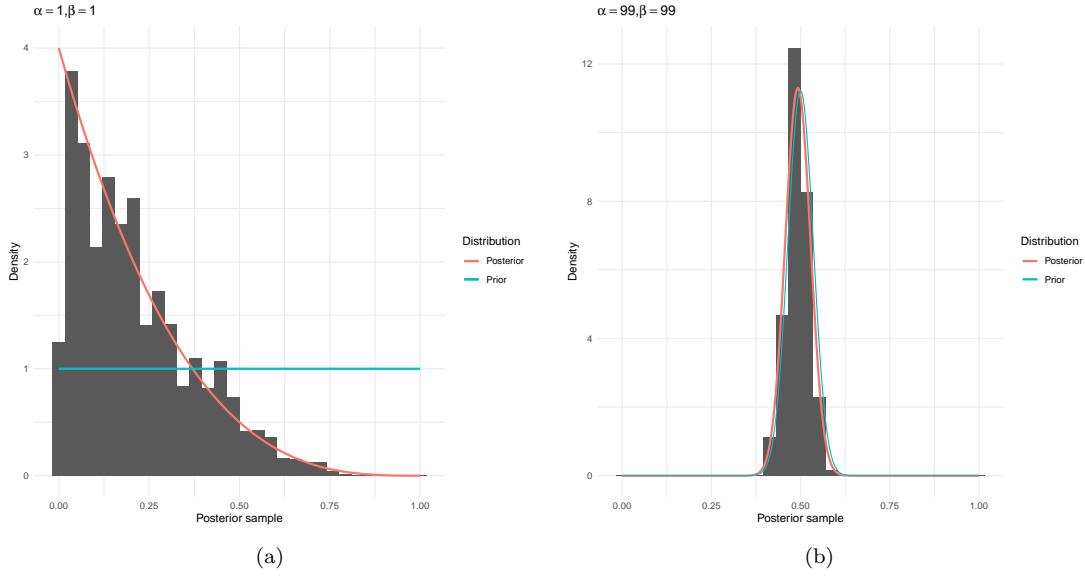


Figure 1: The generated MH samples, theoretical posterior (red) and the prior (green) used for (a)  $\alpha = \beta = 1$  and (b)  $\alpha = \beta = 99$ .

The MH algorithm was used with a  $\mathcal{N}(\cdot, 1)$  proposal function. The use of a symmetric proposal simplifies the MH ratio in Algorithm 1 into a simple Metropolis ratio. The initial value for the algorithm was set to 0.5, but it was observed that the starting value did not have a significant impact on the results in the study. 10,000 samples were generated in the simulation with a 50% burn-in proportion.

The PT algorithm used the same properties mentioned for the MH algorithm, with the only addition being the use of 5 chains. The number of chains was decided on the computation time for the simulation.

The computation times taken to generate the posterior samples have not changed with the increase of the hyper-parameters in either MH or PT algorithms (Figure 6). This is expected as the sampling process is the same regardless of the value of the hyper-parameters.

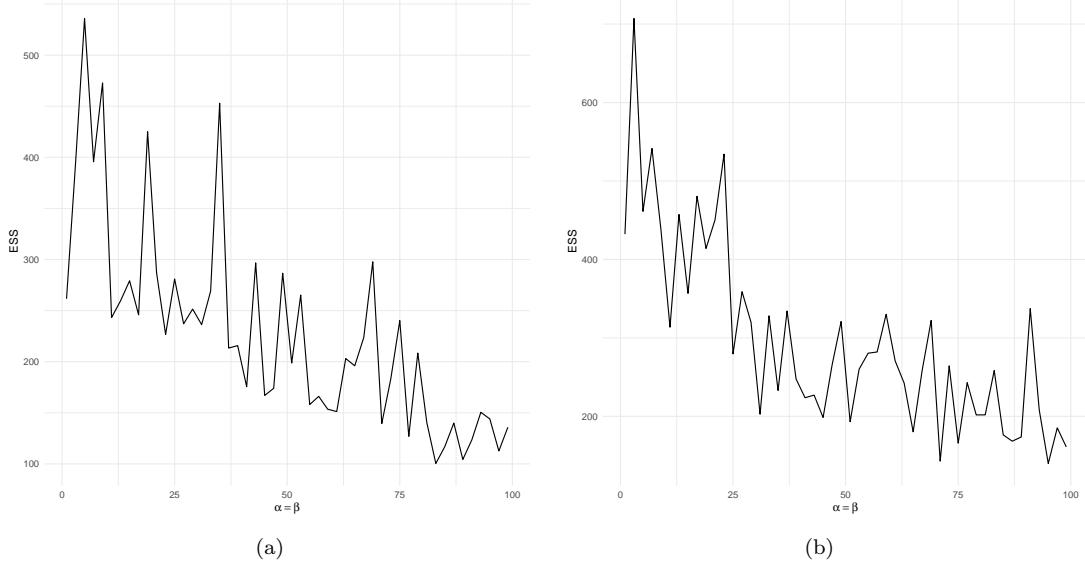


Figure 2: ESS vs. the value of the hyper-parameter for (a) MH and (b) PT algorithms.

Figure 2 shows that PT generally has better performance for any value of the hyper-parameters. But 3 shows that when the computation time is considered, MH has about an 10 times performance advantage over PT.

A simulation was also done to explore the relationship of the ESS with the number of MCMC samples for the two algorithms. The running time increased exponentially with the number of samples, especially for PT (Figure 7). This is understandable as it will have to generate samples for 5 chains separately. This result is consistent with

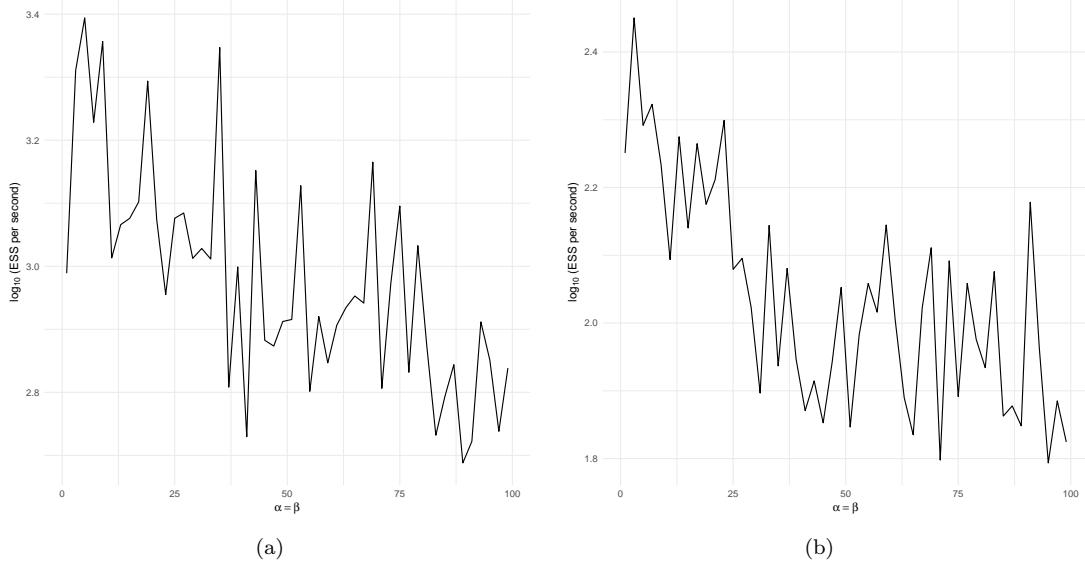


Figure 3: log-transformed ESS per second of computation time vs. the value of the hyper-parameter for (a) MH and (b) PT algorithms.

the observations by Ballnus et al. (2017).

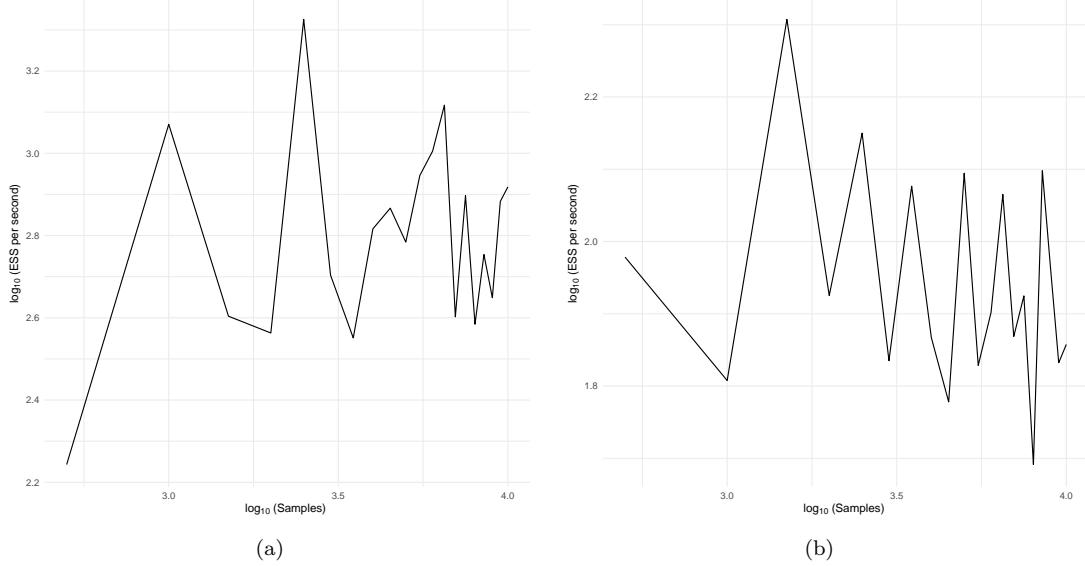


Figure 4: log-transformed ESS per second of computation time vs. log-transformed number of MCMC samples for (a) MH and (b) PT algorithms.

In both algorithms, the time-adjusted ESS does not change when the number of MCMC samples is increased. This result is a bit suspicious as one would expect the ESS to increase with the number of samples. One explanation is that the increased time incurred by increasing the samples is in equilibrium with the ESS.

### 3.2 Mixture distribution

In this example, a mixture distribution is considered as

$$f(x; \mu_1, \mu_2, p) = p\mathcal{N}(\mu_1, 2^2) + (1-p)\mathcal{N}(\mu_2, 2^2). \quad (1)$$

Prior distributions were set as below,

$$\begin{aligned} p &\sim \text{Beta}(3, 2) \\ \mu_1 &\sim \mathcal{N}(1, 2^2) \\ \mu_2 &\sim \mathcal{N}(11, 2^2). \end{aligned}$$

Data for this example was generated from (1) with  $p = 0.3$  and varying values of  $\mu_1$  and  $\mu_2$  (Figure 8). The values of  $\mu_1$  and  $\mu_2$  were selected such that the distance between the two means increases;  $\mu_1 \in (-5, \dots, 2)$  and  $\mu_2 \in (10, \dots, 3)$ .

Only the MH algorithm was used in this example. The properties of the algorithm were set similar to the Beta-Binomial example, with the only exception being using a  $\mathcal{N}(\cdot, 0.5^2)$  proposal. This was done to limit the jump of the proposal from one mode to another.

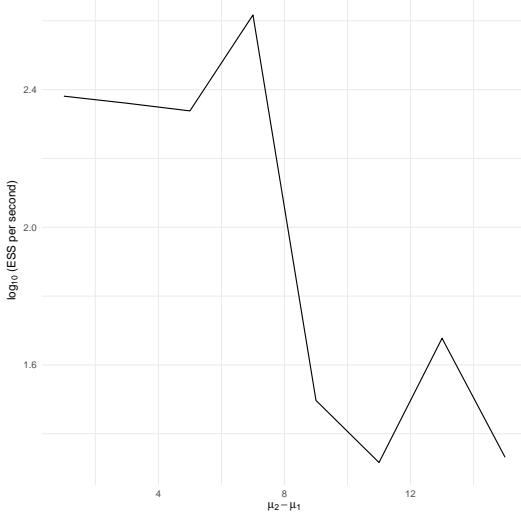


Figure 5: log-transformed ESS per second of computation time vs.  $\mu_2 - \mu_1$  for the mixture distribution example.

Figure 5 shows that similar to the Beta-Binomial example, the ESS decreases when the likelihood has mixture means that are further apart.

## 4 Discussion

One objective for this project was to implement MH and PT algorithms in R. The way the PT algorithm is implemented, it is only able to handle a chain in  $\mathbb{R}$ . This is the reason that PT is not used in the mixture distribution example. The implementation should be expanded to handle chains of higher dimensions.

The second objective was to evaluate the performance of MH and PT algorithms. It was observed that MH was about 10 times faster than the PT algorithm. This is understandable due to the extra sample generation in a multi-chain algorithm such as PT. The speed of the PT algorithm can be greatly improved by using a parallelization library such as `Rmpi`.

It was also seen that PT performed much better than MH when the ESS was considered. This result is consistent with the observations by Ballnus et al. (2017) and Valderrama-Bahamón and Fröhlich (2019). Although this was the case, when ESS was adjusted for the running time, MH performed about 10 times better than PT.

The time adjusted-ESS showed that increasing the number of MCMC samples does not necessarily increase the performance of either algorithm. Which is an interesting property.

The MC standard errors were not considered in this study. To properly evaluate the performance of these algorithms the statistical efficiencies should be also accounted for.

## References

- Ballnus, B., S. Hug, K. Hatz, L. Görlitz, J. Hasenauer, and F. J. Theis (2017, dec). Comprehensive benchmarking of Markov chain Monte Carlo methods for dynamical systems. *BMC Systems Biology* 11(1), 63. 1, 4, 5

- Flegal, J. M. and G. L. Jones (2010). Batch Means and Spectral Variance Estimators in Markov Chain Monte Carlo. *The Annals of Statistics* 38(2), 1034–1070. 1
- Gong, L. and J. M. Flegal (2016, jul). A Practical Sequential Stopping Rule for High-Dimensional Markov Chain Monte Carlo. *Journal of Computational and Graphical Statistics* 25(3), 684–700. 1
- Robert, C. and G. Casella (2010). *Introducing Monte Carlo Methods with R*. New York, NY: Springer New York. 1
- Valderrama-Bahamóndez, G. I. and H. Fröhlich (2019, nov). MCMC Techniques for Parameter Estimation of ODE Based Models in Systems Biology. *Frontiers in Applied Mathematics and Statistics* 5. 5
- Villaverde, A. F., D. Henriques, K. Smallbone, S. Bongard, J. Schmid, D. Cicin-Sain, A. Crombach, J. Saez-Rodriguez, K. Mauch, E. Balsa-Canto, P. Mendes, J. Jaeger, and J. R. Banga (2015, dec). BioPreDyn-bench: a suite of benchmark problems for dynamic modelling in systems biology. *BMC Systems Biology* 9(1), 8. 1

## A Supplementary material

The entire GitHub repository is available at [https://github.com/sachijay/mcmc\\_benchmarking](https://github.com/sachijay/mcmc_benchmarking). Please visit [https://github.com/sachijay/mcmc\\_benchmarking/tree/main/src](https://github.com/sachijay/mcmc_benchmarking/tree/main/src) to view the source code for this project (in R scripts).

## B Additional figures

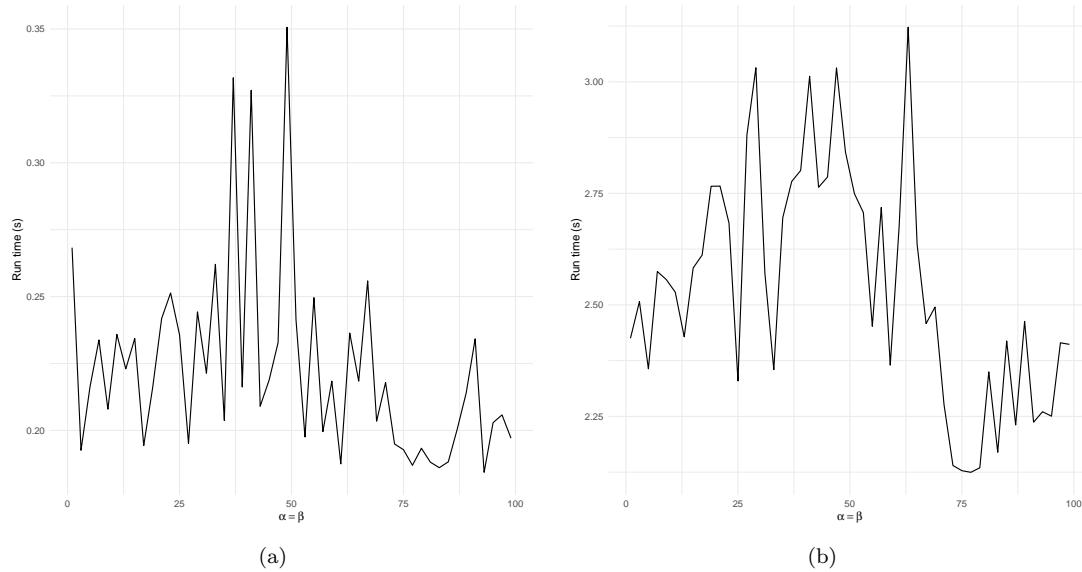


Figure 6: Running time vs. the value of the hyper-parameter for (a) MH and (b) PT algorithms.

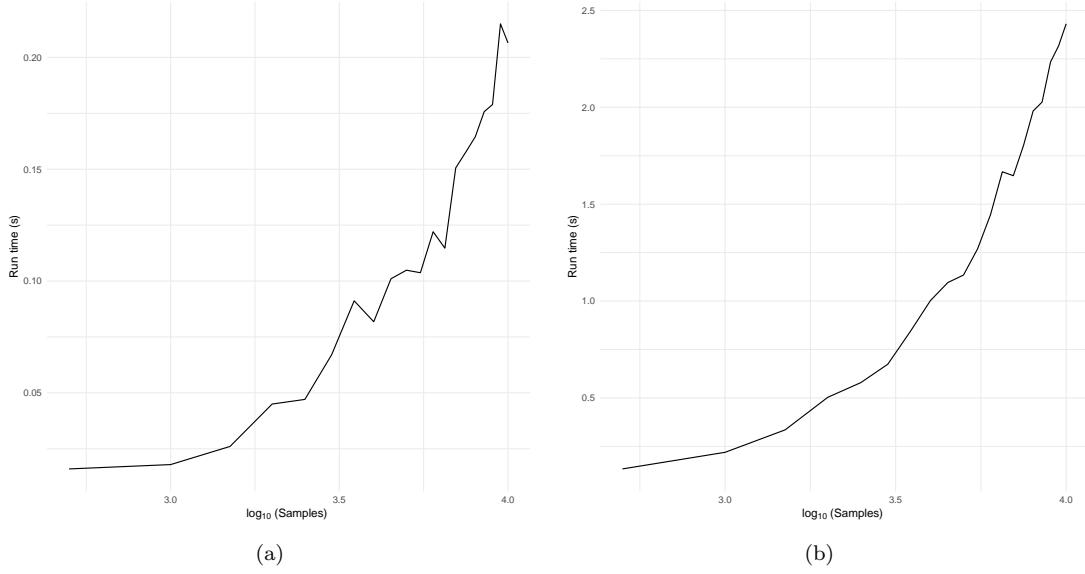


Figure 7: Running time vs. log-transformed number of MCMC samples for (a) MH and (b) PT algorithms.

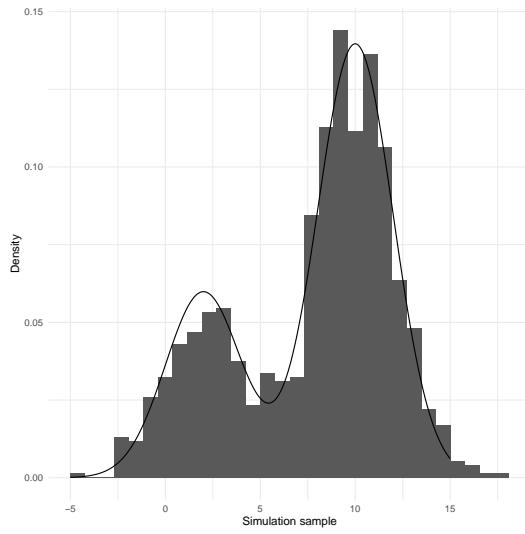


Figure 8: Generated data from a Normal mixture distribution (1) with  $\mu_1 = 2$ ,  $\mu_2 = 10$  and  $p = 0.3$

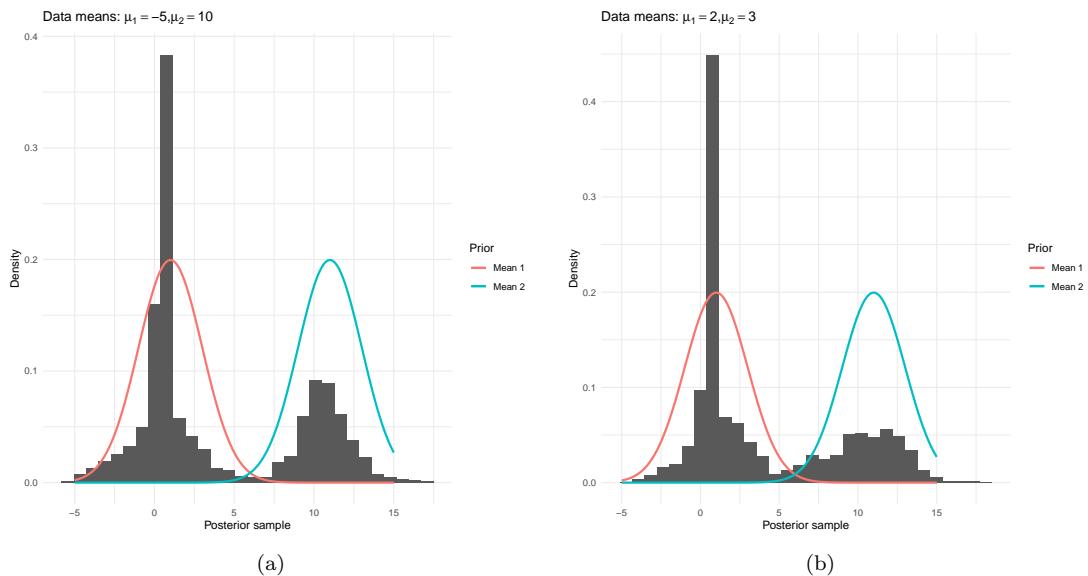


Figure 9: The generated MH samples, prior for  $\mu_1$  (red) and prior for (green) used for (a) ( $\mu_1 = -5, \mu_2 = 10$ ) and (b) ( $\mu_1 = 2, \mu_2 = 3$ ).