VARIANCE REDUCTION IN MARKOV CHAIN MONTE CARLO ALGORITHMS

by

Onur Boyar

B.S., Management Information Systems, Boğaziçi University, 2018

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computational Science and Engineering
Boğaziçi University
2021

ACKNOWLEDGEMENTS

I want to thank my thesis advisor Assoc. Prof. Wolfgang Hörmann for his kind interest, technical advice, and his patience. I learned a lot from him thanks to several courses I had taken from him and the research process for this thesis. I feel privileged to be his student.

I want to thank Assoc. Prof. Bertan Badur who has had a significant impact on my life. He opened my way into academic studies and guided me on every occasion.

I want to thank my family for their support over the years.

Last but not least, I want to thank the inzva community for their support.

ABSTRACT

VARIANCE REDUCTION IN MARKOV CHAIN MONTE CARLO ALGORITHMS

In a simulation study, the output of the simulation is used to estimate target parameters. The estimated parameters typically has a variance. In Monte Carlo simulations, the standard approach to increase the precision of the estimation is to increase the sample size. However, we can increase simulation size until a certain point due to resource restrictions. This is where Variance Reduction (VR) techniques help. These techniques use simulations as an input and return another simulation with almost the same expected value and less variance value. VR techniques are widely studied in the context of Monte Carlo simulations. However, the research on these algorithms for Markov Chain Monte Carlo (MCMC) simulations is limited. In this thesis, we study the effectiveness of VR techniques for MCMC algorithms. We created an experiment environment that allows us to control the difficulty of the problems solved by MCMC algorithms. In our study, we use the Random-Walk Metropolis-Hastings algorithm with different types of proposal distributions. We first evaluate the convergence of the MCMC algorithms, select chains with good convergence properties, and then apply the VR techniques. The VR techniques of interest are *general* VR techniques, which are Antithetic Variates, Control Variates, and Latin Hypercube Sampling. We study the effectiveness of these algorithms using problems with varying difficulties. We tackle the problem of calculating the standard error of the MCMC outputs, evaluate the accuracy of the estimations, and evaluate the correctness of the standard error calculations by calculating the coverage probabilities of the confidence intervals and RMSE/MAE values of the estimations. We show that VR techniques are successful at decreasing the variance of the MCMC simulations without introducing bias.

ÖZET

MARKOV ZİNCİRİ MONTE CARLO ALGORİTMALARINDA VARYANS DÜŞÜRME

Bir simülasyon çalışmasında, simülasyonun çıktısı bir varyansa sahiptir. Daha düşük varyans, daha kesin tahminler anlamına gelir ve daha yüksek varyans, daha az kesin tahminler anlamına gelir. Monte Carlo simülasyonlarında, tahminin kesinliğini artırmak için standart yaklaşım, örneklem boyutunu artırmaktır. Ancak simülasyon boyutunu büyütmek belirli bir noktadan sonra mümkün olmayabilir. Bu sorun, farklı bir yaklaşıma olan ihtiyacı göstermektedir. Varyans Azaltma (VA) teknikleri bu noktada bize yardımcı olabilir. Bu algoritmalar simülasyonları girdi olarak kullanır, benzer aynı beklenen değeri daha az varyans değerine sahip bir şekilde elde etmemizi sağlar. VA teknikleri, Monte Carlo simülasyonları bağlamında geniş çapta çalışılmıştır. Ancak bu algoritmaların Markov Zinciri Monte Carlo (MZMC) simülasyonlarında uygulanmasına ilişkin araştırmalar sınırlıdır. Bu tezde, MZMC algoritmaları için VA tekniklerinin etkinliğini kullanılan problemlerin zorluğunu kontrol etmemizi sağlayan bir deney ortamında araştırdık. Çalışmamızda farklı teklif dağılımlarına sahip Random-Walk Metropolis-Hastings algoritmasını kullanıyoruz. İlk olarak MZMC algoritmalarının yakınsamasını değerlendiriyor, iyi yakınsama özelliklerine sahip zincirleri seçiyor ve ardından VA tekniklerini uyguluyoruz. Uygulanan VA teknikleri *genel* VA teknikleridir. Bunlar Antitetik Değişkenler, Kontrol Değişkenleri ve Latin Hiperküp Örneklemesidir. Zorlukları değişen problemler ile bu algoritmaların etkinliğini inceliyoruz. MZMC simülasyon çıktılarının standart hata değerlerinin doğruluğunu, güven aralıklarını kapsama olasılıklarını ve tahminlerin hata paylarını değerlendiriyoruz. MZMC simulasyonlarında VA tekniklerini kullanarak varyans azaltmanın mümkün olduğunu çeşitli deneyler ile gösteriyoruz.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

AR	Acceptance ratio of the chain
c	Coefficient of the control variates algorithm
K	Transition kernel
f	Density function
$F^{-1}(.)$	Inverse of the cumulative distribution function
$h(X_n)$	Resulting chain output at index n
N	Sample size
$Q(A, B)$	Transition kernel that defines transition from A to B
q	Simulation function
R	Gelman-Rubin diagnostic value
U	Uniform distribution
$Var(x)$	Variance of x
Z	Input matrix of the chain
α	Acceptance probability of Metropolis-Hastings algorithm
$\hat{\gamma}_\ell$	Autocovariance at lag ℓ
ϵ	An arbitrary random walk step
η	An estimator obtained via simulation
θ	Current parameter values of the Markov Chain
ϑ	Proposed values of Metropolis-Hastings algorithm
μ	Mean value
π	Stationary distribution
$\hat{\rho}_\ell$	Autocorrelation at lag ℓ
σ	Standard deviation value
Σ	Covariance matrix
ϕ	Proposed parameter values of the Markov Chain
χ_i^2	Chi-squared distribution
$\chi_{i,j}^2$	F distribution

$\mathbb{1}$	Indicator function

LIST OF ACRONYMS/ABBREVIATIONS

2D	Two Dimensional
4D	Four Dimensional
10D	Ten Dimensional
ACF	Autocorrelation function
AV	Antithetic Variates
CV	Control Variates
GRD	Gelman-Rubin Diagnostics
LHS	Latin Hypercube Sampling
MAE	Mean Absolute Error
MCMC	Markov Chain Monte Carlo
RMSE	Root Mean Squared Error
SE	Standard Error
VR	Variance Reduction
VRF	Variance Reduction Factor

1. INTRODUCTION

The increase in computing power has made it possible to run simulations with large sample sizes. Such large sample sizes improve the precision of the Monte Carlo simulation estimations. However, there is a limit in increasing the simulation size because the simulation will become prohibitively costly after a particular threshold. This problem reveals the need for a different approach like Variance Reduction (VR) algorithms. A Variance Reduction Algorithm is an algorithm that increases the precision of the simulation [1]. These algorithms use simulations as an input and return another simulation with almost the same expected value and less variance. The most common Variance Reduction Algorithms are Antithetic Variates, Control Variates, and Stratification algorithms [2]. Applications of these algorithms are widely studied in the context of Monte Carlo simulations. In this thesis, we research the application of these algorithms for Markov Chain Monte Carlo (MCMC) simulations.

MCMC methods have been used to handle the simulation of complex target distributions. The complexity may originate from the time requirements, the shape of the target density, or the dimension of the problem. The increase in the dimension of the simulation problem may create computational difficulties for the standard Monte Carlo algorithms. One of the alternatives to decrease the computational difficulties and increase the efficiency of the Monte Carlo simulation is using variance reduction methods. However, Monte Carlo algorithms are not suited for each simulation problem. Specific problems need to be solved using MCMC algorithms that are able to explore the area of interest, the target density, without requiring a deep prior knowledge of the target density. Like in the Monte Carlo methods, the efficiency of the MCMC algorithms can also be increased. In the case of high dimensional problems or other types of problems with increased complexity, it is unavoidable to look for methods that increase the efficiency of the simulation due to time and computing power constraints. We can increase the efficiency of the simulation by using parallelized versions of the chains if the problem

is appropriate for parallel processing, or we might choose a different MCMC algorithm that works better and more efficiently for the given problem. On the other hand, we can also increase the simulation's efficiency via variance reduction algorithms. In the literature, there are studies about implementing various variance reduction algorithms to different MCMC algorithms. Since each MCMC algorithm has a unique underlying working mechanism, variance reduction techniques mostly need to be tailored to the MCMC algorithm.

Reducing the variance of the simulation output of the MCMC algorithm might be considered a more complicated problem than applications on Monte Carlo algorithms. There are several reasons behind it. One of the reasons is that successive draws of MCMC algorithms are not independent, unlike Monte Carlo algorithms. This correlation between the successive iterations of the chain prevents obtaining independently and identically distributed (i.i.d.) samples. In addition, there are also algorithm specific challenges. For example, a set of the simulated samples might not be placed in the simulation results due to the rejection mechanism of certain MCMC algorithms, making it challenging to apply specific variance reduction algorithms. Therefore, we should take extra care when applying variance reduction algorithms to MCMC simulations.

There are previous studies on variance reduction of Monte Carlo simulations that take advantage of combining different variance reduction algorithms to achieve higher precision [3]. However, in the literature, studies apply a single variance reduction algorithm to the given problem when it comes to MCMC algorithms. Comparison of different variance reduction techniques and the study of the combination of multiple variance reduction techniques falls short compared to the Monte Carlo simulations. Therefore, variance reduction of MCMC algorithms is an attractive area to conduct research. The limited number of studies focus on two major MCMC algorithms: Gibbs Sampler and Metropolis-Hastings algorithms. These are two of the most prominent MCMC algorithms widely studied in various application areas.

This thesis uses the Random-Walk Metropolis-Hastings algorithm with different types of proposal distributions as MCMC algorithms. After running the simulation, we should follow specific steps to apply the variance reduction algorithms. The first step is to evaluate the performance of the MCMC algorithm and diagnose the convergence of the chain. This step is crucial because, in practical implementations of MCMC algorithms, it is prevalent to obtain a chain that has failed to converge to the target distribution. Therefore, evaluation of the MCMC output and diagnosis of the convergence is within the scope of this thesis. After the empirical validation of the convergence and assessing the error of the simulation, various variance reduction techniques are applied to MCMC algorithms. The individual performance of these variance reduction algorithms is compared. Furthermore, a method to combine certain variance reduction algorithms to achieve further variance reduction is also proposed. Finally, we calculated the Root Mean Squared Error, Mean Absolute Error, and coverage probabilities of naive simulation and variance reduction applied simulation estimations to test the effectiveness of variance reduction techniques.

This thesis is organized in the following fashion. Chapter 2 explains the theoretical background on MCMC algorithms and two of the most widely used MCMC algorithms. In addition, we discuss several techniques to evaluate the performance of the MCMC algorithm. This chapter builds a foundation for the upcoming chapters and experiments. In Chapter 3, we discuss variance reduction algorithms for Monte Carlo simulations. Chapter 4 includes a literature review on variance reduction for Markov Chain Monte Carlo simulations, details about our approach, and implementation details of the variance reduction techniques used in this thesis. Chapter 5 discusses details of the experimental setup and our approach to select chains with good convergence properties. Finally, in Chapter 6, the results of the variance reduction experiments are discussed.

2. THEORETICAL BACKGROUND

In this chapter, preliminary knowledge and theoretical background on the concept of Markov Chain Monte Carlo, Metropolis-Hastings algorithm, and Gibbs Sampler algorithm are presented. Implementation details, advantages and disadvantages of these algorithms are explained and the details about how to process the MCMC simulation output is given. In addition, several approaches to evaluate the performance and the convergence properties of the MCMC algorithm are discussed.

2.1. Markov Chain Monte Carlo

MCMC algorithms allow us to characterize a distribution without complete knowledge about the distribution properties. The name of such algorithms, Markov Chain Monte Carlo, combines two different approaches: Monte Carlo and Markov Chain. The Monte Carlo part means estimating the target values of the distribution by drawing a large number of random samples instead of analytical calculation. The advantage is that creating a large number of observations and calculating an estimate like the mean value of the distribution is easier than the analytical solution if the problem requires solving high dimensional integrals. On the other hand, the Markov Chain part of the MCMC algorithm means that a sequential process generates these random samples. This is due to the Markov property of the Markov chains. Markov property means that the next value depends only on the current value [4]. Markov chains are constructed by using a transition kernel, K. The transition kernel makes the transition from X_n to X_{n+1} possible. This process can be denoted as $X_{n+1} \sim K(X_n, X_{n+1})$. An intuitive example is the random walk process which can be defined as $X_{n+1} = X_n + \epsilon_n$ where ϵ_n is independent from $X_n, X_{n-1}..., X_1$.

There are several properties of an MCMC algorithm. If an MCMC algorithm has a transition kernel that allows the chain to move and explore freely over the

state space, it means that the chain is able to reach the stationary probability distribution, π or the equilibrium distribution in the long run. This property is known as *irreducibility* of the chain. If the chain can reach the stationary probability distribution it means that the successive iterations after reaching that point follow the same distribution. Another property of the MCMC algorithms is the *recurrence* property, which means that the average number of visits to a state within the state space is infinite. A stronger property known as the *Harris recurrence* ensures that the different chains that are started using different starting values will have the same limiting behavior [5]. Therefore, Harris recurrence guarantees the convergence of the chain from every starting point instead of almost every starting point. In practice, achieving the *Harris recurrence* without increasing the simulation size to a very high volume is sometimes a challenging task. There are several tests to assess the impact of the starting point on the chain performance. These tests will be explained in the following sections.

If the transition kernel of the Markov chain is symmetric, another property appears. This property is the reversibility property. It means that $K(X_{n+1}, X_n) = K(X_n, X_{n+1})$. There are also non-reversible versions of the MCMC algorithms but they are out of the scope of this study.

The most interesting consequence and the value of the interest of the convergence property of the MCMC algorithms is the average of the simulated values [6]. This average converges to the expectation $\mathbb{E}[h(X)]$ almost surely. If the Markov chain is reversible and if it can go from every state to every other state, which means the chain is *ergodic*, average of the chain values provide robust estimates of parameters of the limiting distribution [5]. Central Limit Theorem also holds for the average of the chain output $h(X)$ where the average can be calculated as

$$\frac{1}{N}\sum_{n=1}^{N} h(X_n). \tag{2.1}$$

It is not hard to use a Markov chain to solve a given problem. There are several algorithms with proof of concept. The more difficult situation is to find the optimal

parameters of the MCMC algorithms to achieve convergence to the stationary distribution with an acceptable simulation error. The chain should have a good mixing property, i.e., it should perform several jumps between different modes if the problem is multimodal, meaning that the chain can explore the high density regions [7]. It should be able to explore the high probability regions without getting stuck at a point and failing to generate enough samples from other areas. Mixing is a property that shows the stationary distribution is reached quickly from an arbitrary initial position. It takes a small number of iterations for a chain to start to explore the high probability region. It is easier to achieve it in unimodal problems. However, when the dimension of the problem increases, rapid mixing of the chain becomes harder to achieve.

There are several different MCMC algorithms. Two of the most common and most widely studied MCMC algorithms are the Metropolis-Hastings and Gibbs Sampler algorithms.

2.2. Metropolis-Hastings Algorithm

The roots of the Metropolis-Hastings algorithm go back to the Metropolis Algorithm [8]. The original paper of Metropolis algorithm [8] deals with the calculation of the properties of chemical substances. Metropolis algorithm have had a significant impact in Statistics and Simulation and it was named one of the top algorithms of the 20^{th} century.

In the Metropolis algorithm, the following method is proposed.

- Start with any initial set of starting points $\theta^0 = (\theta_1^0, \theta_2^0, .., \theta_d^0)$ and set the iteration counter $j = 1$.
- Move according to the previous positions $\theta^{j-1} = (\theta_1^{j-1}, \theta_2^{j-1}, .., \theta_d^{j-1})$. Propose a new value, ϕ, using the previous value in the chain.
- Calculate the difference in potential energy resulted by the move from θ^{j-1} to ϕ. Accept it with a certain probability. If the move is accepted, set $\theta^j = \phi$.

Otherwise, $\theta^j = \theta^{j-1}$. Therefore, if the proposed move is not good enough, the last value is repeated in the chain.

- Iterate the counter j and return to step 2 until convergence is reached.

The above method defines a Markov chain as the transitions depend only on the states at the previous positions. The transition of the chain depends on a proposed value and a subsequent step of evaluation of this proposal to accept or reject that proposal. Note that the proposed values of the chain can be selected from various distribution types since it is completely unrelated to the equilibrium distribution [5]. These values are specified using another distribution known as the "Proposal Distribution". There are different strategies to define the proposal distributions, which we discuss in Section 2.2.1.

We discussed that there are specific properties of Markov chains. One of the major properties of the transition kernels that we discussed is the reversibility property. The chain is reversible if the kernel K satisfies the following

$$\pi(\theta)K(\theta,\phi) = \pi(\phi)K(\phi,\theta) \quad \forall(\theta,\phi). \tag{2.2}$$

This equality means that the probability of moving to ϕ from θ is the same as moving to θ from ϕ. This is not a necessary condition for convergence of the chain. However, it is a sufficient condition in order to reach the equilibrium distribution, π [5].

The kernel $K(\theta,\phi)$ consists of 2 elements. An arbitrary transition kernel $Q(\theta,\phi)$ and a probability $\alpha(\theta,\phi)$ such that

$$K(\theta,\phi) = Q(\theta,\phi)\alpha(\theta,\phi) \tag{2.3}$$

if $\theta \neq \phi$. Metropolis proposed the above algorithm which has made a significant impact on the literature [8]. Hastings has contributed to the Metropolis Algorithm and proposed to define the acceptance probability in such a way that when com-

bined with the arbitrary transition kernel, it describes a reversible chain [9]. The proposed expression is known as the acceptance probability, α, and calculated as

$$\alpha(\theta, \phi) = min\left(1, \frac{\pi(\phi)Q(\phi,\theta)}{\pi(\theta)Q(\theta,\phi)}\right). \tag{2.4}$$

The ratio inside the $min()$ is known as the Hastings ratio. Algorithms based on chains with transition kernel and acceptance probability are known as Metropolis-Hastings algorithms.

The Metropolis-Hastings algorithm has the following steps.

- Initialize the iteration counter $j = 1$ and set an arbitrary initial starting value θ^0.
- Move the chain to a new value ϕ which is generated using the proposal density, $Q(\theta^{j-1}, 1)$.
- Evaluate the acceptance probability of the proposed step, $\alpha(\theta^{j-1}, \phi)$. If the proposal value is accepted, $\theta^j = \phi$. If it is not accepted, $\theta^j = \theta^{j-1}$ and the chain does not move.
- Change the iteration counter from j to $j + 1$ and return to step 2 until convergence is reached.

Step 3 is performed after the generation of an independent uniform quantity U drawn from $Uniform \sim [0, 1]$. If $u \leq \alpha$, the move is accepted and if $u \geq \alpha$ the move is not allowed. If the move is not accepted, the last value in the chain is repeated.

The chain may be stuck at the same state for many iterations. For this reason the number of the accepted proposal values should be monitored [6]. The most common approach is to calculate the acceptance ratio of the chain. It can be calculated by dividing the number of accepted proposal values and the number of iterations, j.

In the Metropolis-Hastings algorithm, one must choose the proposal distribution type and the parameters of this distribution. Depending on the problem, selecting the correct distribution and correct parameters might be a challenging task. However, the selection of the proper proposal distribution is crucial to obtain a chain with good convergence properties.

2.2.1. Types of Proposal Distributions

The efficiency of the Metropolis-Hastings algorithm increase when the specifics of the target distribution π are taken into account and the optimum proposal distribution is found. In the process of finding a good proposal distribution, it might be necessary and helpful to limit the number of choices of the proposal distribution to the specific family of the distributions to achieve high performance [6]. The selection process starts with the type of proposal distribution. Applications of the Metropolis-Hastings algorithm differ in the way that they propose the new value to move from θ_i to θ_{i+1}. The most common types are Symmetric Chains, Random-Walk Chains, and Independence Chains.

2.2.1.1. Symmetric Chains. A chain is said to be symmetric if it has a transition kernel K such that it is symmetric in its arguments, namely $K(\theta, \phi) = K(\phi, \theta)$, for every pair (ϕ, θ) of states. In the Metropolis-Hastings algorithm, the notion of symmetric chain is applied to the proposed transition. An example of a symmetric chain is the Metropolis version of the algorithm. If the proposed transition depends on (ϕ, θ) only through $|\phi - \theta|$ then $Q(\phi, \theta) = Q(\theta, \phi)$.

2.2.1.2. Random Walk Chains. Random Walk is type of a Markov Chain that defines the transition from step $j-1$ to j as $\theta^j = \theta^{(j-1)} + \epsilon_j$ where ϵ_j is a random variable with distribution independent of the chain. In general, ϵ_j is independent and identically distributed with density f_ϵ. The values of the chain are governed by the values of ϵ_j. It is a typical Markov chain where the next value depends on the current value which is obtained simply by adding a random variate to

the current value of the chain. The idea of Random Walk is implemented to Metropolis-Hastings algorithm as a proposal distribution where the next value of the chain is defined by adding the ϵ value to current value of the chain. The ϵ value can be created using different types of distributions. Some examples can be Gaussian distribution, Student's t distribution, and Uniform distribution. Since the proposed values are based around the previous values of the chain, there is no need to select a mean parameter of the distribution. We are left with the choice of the dispersion of f_ϵ, in other words, the scale parameter value of the proposal distribution. Large values for the scale parameter allow moves that are very distant from previous values but at the likely cost of minimal acceptance rates. On the other hand, small values for the scale parameter only allow moves close to the previous values but with high acceptance rates. There may be a need to run the chain for a long time to reach the equilibrium distribution if small variance values are used. The choice of that value becomes more challenging when the distribution to be sampled from is multimodal. This introduces the problem of jumping between the modes. Small proposal variance values might fail to create proposal values that jump to the other mode. This leads to errors in the estimated values. Therefore, the optimization of the proposal distribution parameters is critical. It can be optimized by observing the acceptance ratio and other properties of the chain. The challenges of the selection of this parameter are widely studied in Section 5.2.

2.2.1.3. Independence Chains. In the case of Independence Chains, the proposed transition is formulated independently from the previous position of the chain. Therefore, $Q(\theta, \phi) = f(\phi)$. In Random Walk Chains, the last accepted value is used to calculate the following proposal values. Independence Chains do not use that information and try to explore the posterior distribution by using random draws from the proposal distribution. Since it is originally a Metropolis-Hastings algorithm, acceptance probability is used to accept or reject the proposed values. It may seem that the independence from the previous state disagrees with the Markovian property of the chain. However, "Q is just a proposal that is combined with an

acceptance probability α to give the transition kernel K of the algorithm" [5]. This transition depends on the previous state, preserving the Markovian structure [5]. On the other hand, Metropolis-Hastings algorithm with independent proposal distribution will likely have a lower acceptance ratio. It does not use the information about the previously accepted values in order to guide the future proposal values. This also limits the usage of the acceptance ratio to optimize the scale parameter of the Independence Chains.

2.3. Gibbs Sampler Algorithm

Gibbs Sampler is one of the most popular MCMC algorithms. It takes a different approach than the Metropolis-Hastings algorithm in order to create samples and reach the stationary distribution. It samples directly from the conditional distributions. To explain the Gibbs Sampler, suppose that we have random variable $X = (X_1, ..., X_p)$. Let us also assume that we can simulate from corresponding univariate conditional densities $f_1, f_2, ..., f_p$ where

$$X_i|x_1, .., x_{i-1}, x_{i+1}, ..., x_p \sim f_i(x_i|x_1, ..., x_{i-1}, x_{i+1}, ..., x_p). \tag{2.5}$$

In order to achieve the transition from $X^{(t)}$ to $X^{(t+1)}$, Gibbs Sampler draws samples from each of the conditional distribution in a sequential way using the values at step t such as

$$\begin{aligned} X_1^{t+1} &\sim f_1(x_1|x_2^t, ..., x_p^t) \\ X_2^{t+1} &\sim f_2(x_2|x_1^t, ..., x_p^t) \\ &... \\ X_p^{t+1} &\sim f_p(x_p|x_1^t, ..., x_{p-1}^t) \end{aligned} \tag{2.6}$$

where f_1, f_2, ..., f_p are called the full conditionals. In Gibbs Sampler, the only densities used in order to draw random variables are these densities. In the above algorithm, at each step, Gibbs Sampler deals with one of the x_1, x_2, ..., x_p. Another type of Gibbs Sampler is the Blocked Gibbs Sampler where all of the x_i^{t+1}'s are

drawn in a single step as shown as

$$X_1^{t+1}, X_2^{t+1}, ..., X_p^{t+1} \sim f(x_1, x_2, ..., x_p | x_1^t, x_2^t, ..., x_p^t). \tag{2.7}$$

Blocked moves can be very beneficial computationally when the correlation between the components is high. Component wise movements might be slow and with the help of joint full conditionals, faster chains can be obtained. The precision of the Blocked version of Gibbs Sampler is also higher in general. In [10], it is shown that estimates of the blocking components are generally more precise than those obtained by component wise calculations.

The characteristic properties of the Gibbs Sampler can be summarized as follows.

- To make use of the full conditionals, Gibbs Sampler limits the number of distribution options used in the modeling. This is because full conditional distributions should be available. There should be prior knowledge about the target distribution to choose the distribution that meets those needs. Therefore, it might be challenging to use Gibbs Sampler in a problem without prior knowledge or domain expertise.
- In the Metropolis-Hastings algorithm, the proposed values are accepted after an evaluation step, and the resulting chain has a certain "acceptance ratio". The acceptance ratio is useful when assessing the convergence of the chain, and it is used to fine-tune the Metropolis-Hastings algorithm. Unlike Metropolis-Hastings, in Gibbs Sampler, none of the proposed values are rejected. In the Metropolis-Hastings algorithm terminology, the acceptance ratio for the Gibbs Sampler is equal to 1.

In order to have the full conditional distribution, there should be at least two variables. Therefore, in order to use the Gibbs Sampler algorithm, the problem should be at least bivariate. After deriving the full conditional distributions, the Gibbs Sampler algorithm updates the parameters of the conditional distributions in each

step to reach the stationary distribution. In the Metropolis-Hastings algorithm, the acceptance ratio is one of the values that can be used while diagnosing the convergence to the stationary distribution. As there is no acceptance ratio in the Gibbs Sampler, one must rely on other techniques to evaluate the convergence of the chain. In the next section, we cover several evaluation techniques.

2.4. Evaluating the MCMC Algorithms

One of the most challenging aspects of working with MCMC algorithms is to diagnose the convergence of the chain [11]. In MCMC algorithms, convergence to the stationary distribution is achieved when the length of the chain goes to infinity. However, this is not applicable in practice, and a chain obtained at a sufficiently large number of iterations is taken as the final output. The primary question is how to determine this "sufficiently large" value. The answer to this question is not simple, and there are theoretical and empirical strategies to answer it [6]. Theoretical methods try to measure distances and establish bounds on distribution functions generated from a chain. However, theoretical approaches have had little impact on practical work so far. Therefore, in this study, the focus is on empirical evaluation methods.

The strategy for diagnosing the convergence of the MCMC algorithms depends on the type of the algorithm. For example, the convergence of the Metropolis-Hastings algorithm depends not only on the proposal distribution but also on the initial values of the chain. Two chains with the same proposal distribution with different starting points might give totally different results. Due to this problem, the Metropolis-Hastings algorithm should run several times using other initial starting points [12]. If the chain is over sensitive to the different initial starting points, it might indicate poor performance. To fix this problem, one might need to change proposal distribution parameters or the proposal distribution itself. A different approach to diagnosing the convergence of the chain is proposed in [13]. It is recommended to run a single, very long chain and check its properties. This can be done by using visualization methods. It can be said that there are several

proposals for diagnosing the convergence of the chain. Some of these techniques are widely accepted, and some of them are not. This section will discuss several evaluations and processing techniques of the MCMC output. Effect of the initial points, strategies to decrease the impact of an arbitrary initial point, diagnosis of the convergence, quality of the obtained output will be discussed.

2.4.1. Post Processing of the MCMC Output

MCMC simulations are started using an initial point. These initial points might be irrelevant with the high probability region of the stationary distribution. Furthermore, finding a good starting point that is close to a high probability region is a challenging task, especially in higher dimensions. Therefore it is important to understand that the chain is not sampling from the stationary distribution until it converges to the stationary distribution. While the chain is converging to the stationary distribution, sampled values are coming from different distributions than the stationary distribution. Therefore, using the whole set of values in the chain output might cause errors in the estimations because of the values obtained during the initial period. The most common way to decrease the impact of the initial points is to discard these points from the chain output. This operation is known as "burn-in" [14]. The rule of thumb is to delete the first 100 or 1000 samples from the chain. However, it is not a good practice to determine the burn-in amount in advance. One strategy is to run the algorithm using different starting values, visualize the chain and observe the initial behavior of different chains. This will give an idea about what should be a good value for the burn-in period. If the first b values among n samples are removed in a *burn-in* period, the estimation becomes as follows

$$\frac{1}{b-a}\sum_{i=b+1}^{n} h(x_i). \tag{2.8}$$

In Figure 2.1, it is observed that the chain has a starting value that is far away from the high density region. Including the samples from this initial phase

to the final calculations will impact the estimated mean of the parameters and the standard error of the simulation. It will have a higher impact if the number of samples is low. Another common technique to apply as a post processing of MCMC output is *thinning* of the chain [14]. Thinning basically means subsampling the chain output in a specific way. The successive iterations of Markov chains are dependent and this brings the problem of autocorrelation, which is simply the correlation between the successive iterations in the chain. Thinning is a strategy that decreases autocorrelation by discarding some of the values from the chain. An example might be removing every 2^{nd} value from the chain so that the autocorrelation of the chain is decreased. However, even though it decreases the autocorrelation, it is not always preferable to have a smaller sample size. Decreasing the sample size by thinning will result in having a smaller effective sample size, which is an estimate of the sample size required to achieve the same level of precision if that sample was an i.i.d. random sample. Therefore, thinning is recommended only when there are memory limitations [15].

2.4.2. Number of Chains and Sample Size

In the literature, there is no widely accepted technique when it comes to running multiple chains or running one long chain. One of the most famous applications of running multiple chains is Gelman-Rubin diagnostics [12] which is a method to test the convergence of the chain by evaluating intra-chain variance and inter-chain variance values. To calculate the Gelman-Rubin diagnostic, there should be at least two chains that are initialized using different starting values. Let us denote the number of chains with different starting values with M and $m \in [1, M]$.

Each chain m is denoted as $\theta^m = (\theta_1^m, ..., \theta_n^m)$. After post processing of the chains, estimations of the target variables in each chain is calculated as follows

$$\hat{\theta}_m = \frac{1}{N_m} \sum_i^{N_m} \theta_i^m. \tag{2.9}$$

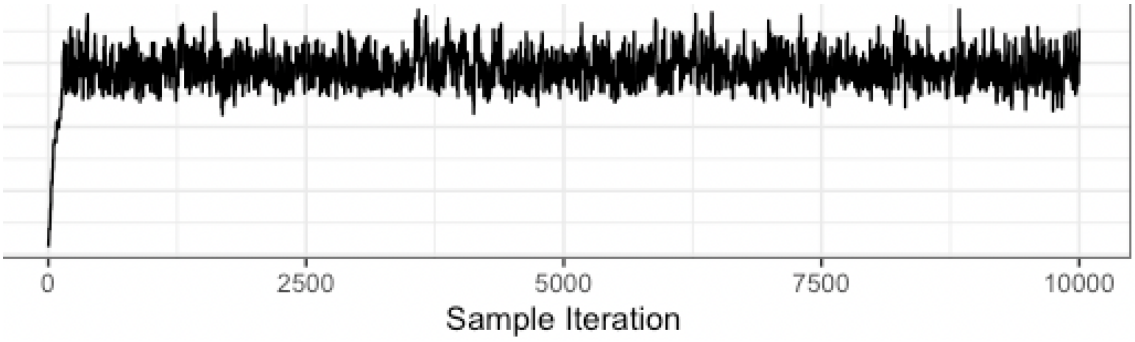

Figure 2.1. First iterations should be discarded due to poor starting point.

In the next step, the intra-chain variance is calculated as follows

$$\sigma_m^2 = \frac{1}{N_m - 1} \sum_i^{N_m} (\theta_i^m - \hat{\theta}_m)^2. \tag{2.10}$$

In addition, the mean value is estimated using all of the chains as follows

$$\hat{\theta} = \frac{1}{M} \sum_m^M \hat{\theta}_m. \tag{2.11}$$

Using the above estimation, averaged variances of the chains and the difference between the individual estimated means and the joint estimated mean is calculated as follows

$$\begin{aligned} W &= \frac{1}{M} \sum_m^M \sigma_m^2 \\ B &= \frac{N}{M-1} \sum_m^M (\hat{\theta}_m - \hat{\theta})^2. \end{aligned} \tag{2.12}$$

Finally, define $\hat{V} = \frac{N-1}{N} W + \frac{M+1}{MN} B$. This is an unbiased estimator of the true variance. However, if the chains have converged, W is also an unbiased estimate of the true variance. Gelman-Rubin diagnostic states that the ratio $R = \sqrt{\hat{V}/W} \approx 1$ if the chain is converged. In [12], it is stated that if $R > 1.2$, it can be accepted as an indication for nonconvergence.

Although it is a useful method, Gelman-Rubin diagnostic should not be the only method to evaluate the convergence of the chain. An example where

Gelman-Rubin diagnostics gives misleading results is as follows. To simulate from a multimodal distribution one of the options is to use the Random-Walk Metropolis-Hastings algorithm. When the scale parameter value of the proposal distribution is low and the distance between the modes is high, the chain may fail to jump between the modes regardless of different M starting values. In that case, $R \approx 1$ might be obtained although the mixing property of the chains are not good and the simulation error is potentially high.

Even though it is common to use multiple chains with different starting points, the other strategy is to run one long chain and use it in the convergence diagnostics [16, 17]. As the dimension of the problem increase, the effect of the initial point starts to decrease. As an example, if we are to sample from the mixture of 10 dimensional multinormal distributions, it is challenging to find a good set of different starting points. For the lower dimensions it might be practical to use multiple chains with different starting points but in higher dimensions using one long chain might be a better strategy.

2.4.3. Sample Autocorrelation

When the consecutive draws from the MCMC tend to be very close to each other, it is an indication that the chain is not moving efficiently and quickly through its space. The quantification of this property can be used as another tool to evaluate the performance of the chain [14]. It can be done by computing the sample correlations between the simulation output values x_i and lagged values $x_{i+\ell}$. Then, the *sample autocovariance* of x_i at lag ℓ for $0 \leq \ell < n$ is

$$\begin{aligned} \hat{\gamma}_\ell &= \tfrac{1}{n} \textstyle\sum_{i=1}^{n-\ell+1} (x_i - \bar{x})(x_{i+\ell} - \bar{x}) \\ \bar{x} &= \tfrac{1}{n} \textstyle\sum_{i=1}^{n} x_i \end{aligned} \tag{2.13}$$

The *sample autocorrelation (ACF)* of x_i at lag ℓ is

$$\hat{\rho}_\ell = \tfrac{\hat{\gamma}_\ell}{\hat{\gamma}_0}. \tag{2.14}$$

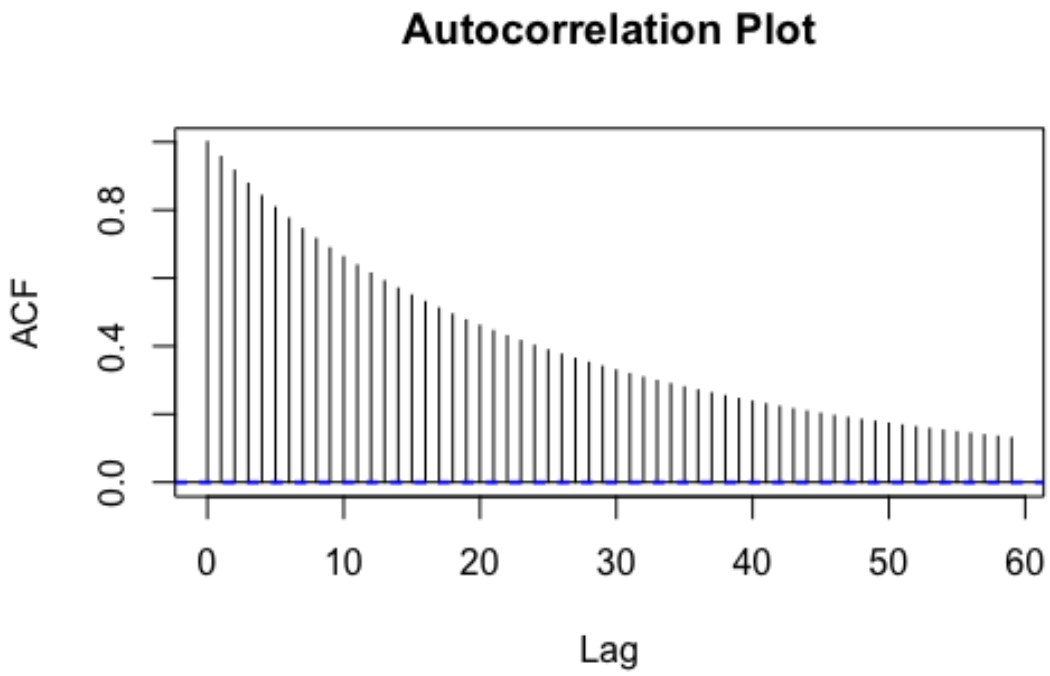

Figure 2.2. This figure shows autocorrelation functions of values sampled by Random Walk Metropolis-Hastings algorithm.

Autocorrelation value in lag ℓ can be easily analyzed by plotting $\hat{\rho}_\ell$ values for $\ell = 1, 2, 3..., n$. It can take values in the range of [1, -1]. Lag 0 autocorrelation is by definition equal to 1. The desired property of the sample autocorrelation values of the chain is to observe a decrease in autocorrelation values starting from lag 1 autocorrelation. An example plot is given in Figure 2.2, where sample autocorrelation values are not decreasing quickly. The autocorrelation plot has horizontal blue bars at $\pm 2/\sqrt{n}$. These horizontal bars indicate a region where the observations start to become statistically independent.

In Figure 2.3, it is observed that most of the values are within that region and autocorrelation values are decreased starting from lag 1 autocorrelation, they are mostly lower than the blue dashed line. Such autocorrelation values indicates that the successive iterations are not correlated with each other. When the successive iterations are dependent, it takes much more iterations to obtain such autocorrelation values as it is shown in Figure 2.2.

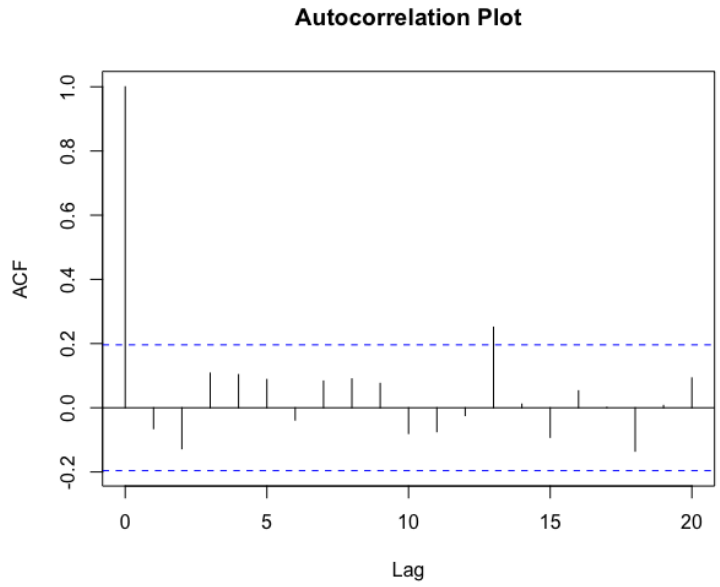

Figure 2.3. Autocorrelation plot of a sample from standard normal distribution.

2.4.4. Visualization Methods

Visualization methods [17] to evaluate the performance of the chain might be the most common model evaluation method. In this technique, the values of the target function are plotted as a trace plot. By this visualization, several properties of the chain can be observed. The most important point to check is if the chain can explore the target region successfully and quickly. In addition, this kind of visualization lets the practitioner evaluate the mixing property of the chain.

Furthermore, the sensitivity to the initial starting point can also be observed. Observing the chain's behavior until it reaches the high probability region lets the practitioner make more informed decisions and tune the model. Analyzing such a plot for multiple chains with different starting values is similar to the Gelman-Rubin diagnostics.

Besides, the chain's behavior after it reaches the high probability region can also be easily analyzed via visualization methods, and model parameters can be tuned accordingly. For instance, it is important to observe several jumps between the modes in the case of multimodal problems.

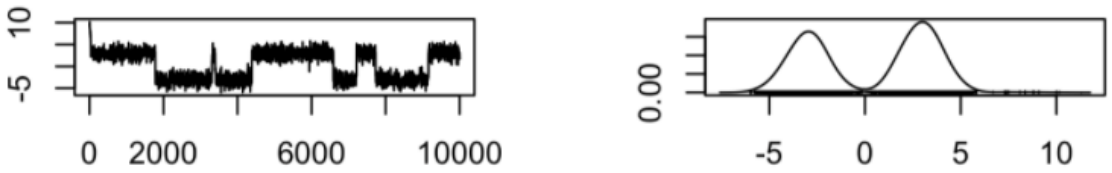

Figure 2.4. A poor mixing in a multimodal problem.

Figure 2.5. A better mixing in a multimodal problem.

In Figure 2.4, a chain with poor mixing is demonstrated as an example. There are very few jumps between the modes. Instead, it takes several samples before jumping to the other mode. In Figure 2.5, it is observed that right after the initial steps of the chain, it achieves to explore the high probability region. Therefore, it has a good mixing property.

2.4.5. Standard Error and Confidence Intervals

Estimated values should be reported along with the standard error of the estimations. Markov Chain Monte Carlo techniques give us correlated samples, which means they are not i.i.d. This makes the calculation of standard error less straightforward than calculating it for Monte Carlo estimations [18,19]. Regardless of the sample size of the simulation, there is an unknown Monte Carlo error, $\hat{\mu_n} - E[\mu]$ where n is the sample size. We cannot calculate this error directly due to the correlation of samples. However, it is possible to approximate the sampling distribution if the Markov chain central limit theorem holds. That is,

$$\sqrt{n}(\hat{\mu_n} - E[\mu]) \xrightarrow{\mathrm{D}} N(0, \sigma^2) \tag{2.15}$$

as $n \to \infty$ in dimension D. However, due to the autocorrelation in the chain, the estimated variance is not equal to the expected value of the variance [18].

Therefore, estimating the variance of the Markov Chain requires special techniques [18, 20]. There are several techniques at our disposal, such as non-overlapping batch means, overlapping batch means, spectral variance, block bootstrapping, etc. In this study, we use the non-overlapping batch means technique to calculate the standard error of the mean estimations, and we use block bootstrapping to calculate the standard error of estimated variance and estimated covariance. Let us first describe the non-overlapping batch means approach. In this technique, the chain is run for $N = ab$ iterations, a non-overlapping batches with size b are created. Then, the mean estimate of each batch is calculated as follows

$$Y_k = \frac{\sum_{i=(k-1)b+1}^{kb} g(X_i)}{b} \tag{2.16}$$

where $g(.)$ is the function that expected value is computed for Markov chain X. These estimates are averaged to yield an overall estimate as follows

$$\hat{\mu} = \frac{\sum_{i=1}^{a} Y_i}{a}. \tag{2.17}$$

In the final step, we estimate the variance as follows

$$\hat{\sigma}^2 = \frac{b}{a-1} \sum_{k=1}^{a} (Y_k - \hat{\mu})^2. \tag{2.18}$$

By this means we can calculate an estimate for standard error [20]. Using the $\hat{\sigma}^2$, we can create confidence intervals as follows. We can set $t_* = 2$ to calculate the standard error and create 95% confidence interval of the estimations [18] using the following formula

$$\hat{\mu} \pm t_* \frac{\hat{\sigma}}{\sqrt{n}}. \tag{2.19}$$

The second method we use in our study is Block Bootstrap [21, 22]. Before explaining Block Bootstrap, we will describe Bootstrapping and its usage to cal-

culate standard error of the estimations [23, 24]. Bootstrap is a computational resampling technique for finding standard errors. The idea of bootstrapping is resampling from the sample with replacement iteratively, calculating the statistic of interest in each of the samples, and calculating the standard error among these calculated statistics. The calculated standard error can be used to create confidence intervals. Bootstrapping has been widely used and studied in the context of Monte Carlo simulations. On the other hand, the study on the utilization of bootstrap methods in MCMC simulations is limited. Let us first describe the idea of the bootstrapping, specifically *Nonparametric Bootstrap*, approach and then discuss its challenges when we have data with autocorrelation.

To begin the bootstrap procedure, we need to define the number of bootstrap samples. In [24], it is shown it is possible to get robust standard error calculations when the number of bootstrap samples is between 25-200. After defining the number of bootstrap samples, B, we are ready to implement *Nonparametric Bootstrap*. The Nonparametric bootstrap procedure for estimation of the standard error of the estimate $\hat{\theta} = s(x)$ is as follows.

- Select B independent bootstrap samples $x^{*1}, x^{*2}, ..., x^{*B}$ with sample size n and drawn with replacement from x.
- Evaluate corresponding estimation for each bootstrap sample $\hat{\theta}^*(b) = s(x^{xb})$ for $b = 1, 2, ..., B$.
- The standard error is estimated by calculating the sample standard deviation of the B repetitions

$$\hat{se}_B = \left\{ \sum_{b=1}^{B} \left[\hat{\theta}^*(b) - \hat{\theta}^*(.)\right]^2 / (B-1) \right\}^{1/2} \tag{2.20}$$

where $\hat{\theta}^*(.) = \sum_{b=1}^{B} \hat{\theta}^*(b)/B$.

The bootstrap procedure explained above has a certain assumption that limits its usage for autocorrelated data, which is the i.i.d. assumption. Using Non-

parametric Bootstrap for autocorrelated data destroys the dependence between successive values. We need to change the algorithm such that it retains the correlated structure of the data. One of such approaches is *Block Bootstrap* [21, 22]. This method suggests that if we divide our sample into m nonoverlapping blocks of length k we can sample with replacement within the blocks and therefore retain the correlated structure of our sample. The accuracy of such an approach depends on the size of k. Having $k = 1$ results in resampling individual observations, which is similar to Nonparametric Bootstrap. In [25], setting $k = n^{1/3}$ is proposed as a dynamic method to choose k as sample size changes. This procedure is also known as the *Simple Block Bootstrap*. There are other versions of Block Bootstrap as well. The other methods take different approaches to create *blocks*. For example, in *Moving Block Bootstrap* [21], data is split into overlapping blocks. In our study, we use the *Simple Block Bootstrap* method.

3. VARIANCE REDUCTION FOR MONTE CARLO SIMULATIONS

Monte Carlo simulations have an error variance calculated by σ^2/n. By increasing the sample size, n, lower variance, and, therefore, better estimates can be obtained. However, as the sample size increases, the computing time also increases. Therefore, a need for an efficient way to decrease the variance without changing the expected value emerges. Methods to achieve this task are known as variance reduction techniques.

Variance reduction techniques are widely used in Monte Carlo simulations. Like it is briefly discussed in the first chapter, the aim is to increase the precision of the simulation. Application areas might be estimating the probability of rare events, high dimensional simulations, or generally simulations that take hours or more to finish. Depending on the time constraints, increasing the precision by only 10% can be good enough. The most common techniques are antithetic variates, control variates, and stratified sampling [26].

In order to evaluate the performance of the Variance Reduction algorithm, the *Variance Reduction Factor* metric is used [2] along with the error of the simulation. Variance Reduction Factor is simply the ratio between the variance of the naive simulation, which is the simulation output before applying any of the variance reduction algorithms, and the variance of the simulation output obtained from the variance reduction algorithm.

Several variance reduction techniques for Monte Carlo methods are explained in the following sections, along with implementation details. These techniques are namely Control Variates, Multiple Control Variates, Antithetic Variates, and Stratification.

3.1. Control Variates

Control variates technique has a simple yet effective idea. If one can find a problem that has some similarities with the problem at hand and if the solution of this problem is known, it can be used to decrease the variance of the simulation [2]. The Control Variates algorithm exploits information about the errors of the estimation problem with known quantities to reduce the error of the estimation problem with the unknown quantity. Another explanation of the idea of control variates is that utilizing what we already know about the system or the problem at hand. For instance, if the problem is about estimating the Asian Option Prices, we can use another type of Option for which we already know the exact result to decrease the error in the Asian Option Prices. In this case, European Option prices can be used as control variates thanks to the availability of the Black-Scholes formula [27] to calculate the expectation of the control variate.

Suppose we wish to estimate $\mu = \mathbb{E}[Y]$, where $Y = q(X)$ is the output of the simulation experiment to estimate the Asian Option prices. Following the above example, also suppose that the Z is the output of the European Option prices simulation, and we know $\mathbb{E}[Z]$ by Black Scholes formula. These values are enough to create a control variate for μ. Let us denote the following.

- $\hat{\mu} = Y$
- $\hat{\mu}_c = Y - c(Z - E(Z))$, where c is some real number.

By above definition of the $\hat{\mu}_c$, it is clear that $E[\hat{\mu}_c] = \mu$. Here the question of interest is if the variance of the $\hat{\mu}_c$ is lower than μ. The variance of the $\hat{\mu}_c$ is defined as follows

$$V(\hat{\mu}_c) = V(Y) + c^2 V(Z) - 2cCov(Y, Z). \tag{3.1}$$

In the above formula, the variance of Y and Z are fixed values. However, we can choose c such that it minimizes the $V(\hat{\mu}_c)$. The optimal value of the c, denoted by

c^*, to get lowest variance can be obtained as follows

$$c^* = \frac{Cov(Y,Z)}{Var(Z)}. \tag{3.2}$$

Substituting c^* to equation 3.1 provides

$$V(\hat{\mu}_c) = V(Y) - \frac{V(Y)V(Z)Cor(Y,Z)^2}{V(Z)} = V(Y)(1 - Cor(Y,Z)^2). \tag{3.3}$$

By equation 3.3, it can be said that for non-zero values of $Cor(Y,Z)$, variance reduction is obtained. The quantity Z is the control variate to increase the precision of the simulation $Y = q(X)$. It is worth noting that the c^* value found in equation 3.2 is also the least squares solution of the following simple linear regression problem, $Y = \alpha + c^*Z$. This way of calculating c^* is especially useful when working with multiple control variates. The key component to construct a control variate is to have the equality in $\hat{\mu}_c = Y - c(Z - E(Z))$ and have a high $Cor(Y,Z)$ value.

To evaluate the performance of the control variate, we calculate the *Variance Reduction Factor*, denoted by VRF. The Variance Reduction Factor of a single Control Variate (CV) is defined as follows

$$VRF(CV) = \frac{1}{1-Cor(Y,Z)^2}. \tag{3.4}$$

3.2. Multiple Control Variates

Using multiple control variates together to reduce the variance of the estimator is also applicable. Given that there are more than one control variates with known expected values, using all of those control variates is not a hard task [2]. Let us have m different control variates denoted as $Z_1, Z_2, ..., Z_m$. The aim is to

estimate $\mu = \mathbb{E}[Y]$. The unbiased estimator for μ using control variates is as follows

$$\hat{\mu}_c = Y + c_1(Z_1 - \mathbb{E}[Z_1]) + ... + c_m(Z_m - \mathbb{E}[Z_m]). \tag{3.5}$$

It is easy to observe that $\mathbb{E}[\hat{\mu}_c] = \mu$. The main challenge is to find the optimal set of values for $c_1, ..., c_m$ so that the variance reduction is achieved. In Section 3.1, we described that finding c^* is equivalent to finding the slope coefficient of the simple linear regression problem. Similarly, in the multiple control variates algorithm, optimal values of $c_1, ..., c_m$ come from the solution following the linear regression problem

$$Y = \alpha + c_1^* Z_1 + ... + c_m^* Z_m. \tag{3.6}$$

The solution of the above regression gives c_i^* values. Using these c_i^* values will be enough to construct an unbiased estimator of μ, potentially having a lower variance.

Applying control variates as a variance reduction technique using one or multiple control variates is shown above. However, without finding a good control variate this technique will not be as efficient as possible. In order to find good control variates, there are two possible approaches. These approaches are listed as follows.

- Internal Control Variates
- External Control Variates

Internal Control Variates make use of the random variates that are already part of the simulation. In most cases, these control variates are the random numbers generated to start and run the simulation. An example can be given from Option Pricing. To simulate an Asian Option Call, an input matrix of dimension d, Z, is created using the standard normal distribution, $N(\mu = 0, \sigma = 1)$. Using these values, the Asian Option Call is simulated for a given period. The input matrix,

Z, can be used as *Internal Control Variates* in this example.

External Control Variates do not use variates that are part of the simulation. Instead, *External Control Variates* exploits another problem with a known solution to decrease the simulation variance. In the Option Pricing example, to use *External Control Variates*, a similar problem to Asian Call Option with a known expected value is needed. One alternative is to use *European Call Option* values for the same period as an *External Control Variate* because the expected value of *European Call Option* is available thanks to the Black-Scholes formula. Thus, by using the simulated values of the *European Call Option* and the expected value of it, an external control variate can be created [2, 28]. There are advantages and disadvantages of including External Control Variates in simulation practice. The main disadvantage is that it is hard to find such a control variate, which requires domain knowledge. In addition, it may increase simulation time due to additional computations. On the other hand, external control variates are very effective and can provide high variance reduction.

3.3. Antithetic Variates

Suppose that Y is the simulation output and the goal is to estimate $\mu = \mathbb{E}[Y]$. In order to find $\mathbb{E}[Y]$ with lower variance, two different samples Y_1 and Y_2 can be used and μ can be estimated using the average of Y_1 and Y_2, $Y = (Y_1 + Y_2)/2$. In order to see the potential reduction in the variance, let us write the variance of the Y as follows

$$V(Y) = \frac{V(Y_1)+V(Y_2)+2Cov(Y_1,Y_2)}{4}. \tag{3.7}$$

The variance of the Y depends on the variance of Y_1, Y_2 and the *covariance* of Y_1, Y_2. Obviously, if $Cov(Y_1, Y_2) < 0$, variance reduction is achieved. It is not straightforward to achieve that since Y_1 and Y_2 must come from the same distribution. The way to obtain such negative covariance is as follows. Y_i's are outputs of certain simulations. The simulation function has certain input values, and the

negative correlation between simulation outputs can be achieved if appropriate input values can be passed to the simulation function for each Y_i. In other words, although the simulation output, Y_i, cannot be controlled, it is possible to control the input values of the simulation and how to create this input value. Let us denote the simulation input with X and the simulation function with $q(.)$ so that $Y_i = q(X_i)$. In order to achieve negative correlation of Y_1 and Y_2, a sensible strategy is to create negatively correlated input values X_1 and X_2. There are some options to create negatively correlated X_i values.

- If X $\sim$ Uniform[0,1], X_1 = 1 - X_2.
- If X is coming from a symmetric distribution around 0, X_2 = -X_1.

Using such input values may allow us to obtain negatively correlated simulation outputs. The input values can also be multidimensional. In that case above operations can be applied to each or some of the input values. When it comes to evaluating the performance of the Antithetic Variates algorithm in terms of the variance reduction, it is essential to note that two different simulations are used in the Antithetic Variates algorithm [2,26]. If the naive simulation has sample size n, negatively correlated chains should be of size $n/2$ each to make a fair comparison.

3.4. Stratification

In stratification or stratified sampling, the idea is to split the domain into separate regions [16]. Then, simulation samples are taken from each separated region, and the results are combined to estimate the expected value. The idea is that if each region gets an approximately equal share of points, we can get a better answer and potentially obtain a simulation with lower variance [26].

The goal is to estimate $\mu = \int_D f(x)p(x)dx$ where D is the domain. D is partitioned into mutually exclusive and exhaustive regions, denoted by D_j, for $j = 1, ..., J$. These regions are within the D and they are known as strata and denoted by w_j where $w_j = \mathbb{P}(X \in D_j)$ assuming $w_j > 0$ [26]. Next let $p_j(x) =$

$w_j^{-1} p(x) \mathbb{1}_{x \in D_j}$, which is the conditional density of X given that $X \in D_j$.

The sizes w_j of the strata and how to sample $X \sim p_j$ for $j = 1, ..., J$ should be known in order to use stratified sampling [26]. Let $X_{ij} \sim p_j$ for $i = 1, ..., n_j$ and $j = 1, ..., J$ be sampled independently. The stratified sampling estimate of μ is,

$$\hat{\mu}_{strat} = \sum_{j=1}^{J} \frac{w_j}{n_j} \sum_{i=1}^{n_j} f(X_{ij}). \tag{3.8}$$

In order to show that the stratified sampling is unbiased, the following

$$\begin{aligned} &\mathbb{E}[\hat{\mu}_{strat}] = \sum_{j=1}^{J} w_j \mathbb{E}\left(\frac{1}{n_j} \sum_{i=1}^{n_j} f(X_{ij}) \right) = \sum_{j=1}^{J} w_j \int_{D_j} f(x) p_j(x) dx \\ &= \sum_{j=1}^{J} \int_{D_j} f(x) p(x) dx = \int_D f(x) p(x) dx = \mu \end{aligned} \tag{3.9}$$

is given. In addition, the variance of the stratified sampling estimate is as follows

$$Var(\hat{\mu}_{strat}) = \sum_{j=1}^{J} w_j^2 \frac{\sigma_j^2}{n_j}. \tag{3.10}$$

The question of interest here is that if the variance of $\hat{\mu}_{strat}$ is lower than the variance of μ. Below, it is demonstrated that stratified sampling with a proportional allocation does not have a higher variance than the naive simulation for the stratum sample sizes, $n_j = n w_j$. Let us suppose that all the n_j are integers. Then, for proportional allocation, the equation to estimate the mean becomes as follows

$$\hat{\mu}_{prop} = \frac{1}{n} \sum_{j=1}^{J} \sum_{i=1}^{n_j} f(X_{ij}). \tag{3.11}$$

In addition, with proportional allocation, the $Var(\hat{\mu}_{strat})$ becomes

$$\sum_{j=1}^{J} w_j^2 \frac{\sigma_j^2}{n w_j} = \frac{1}{n} \sum_{j=1}^{J} w_j \sigma_j^2. \tag{3.12}$$

By the above equation, it can be shown that the stratified sampling with proportional allocation cannot have a larger variance than ordinary Monte Carlo sampling. Let us separate the within, σ_w^2, and between, σ_B^2, stratum variances and

denote them as follows

$$\begin{aligned} \sigma_w^2 &= \textstyle\sum_{j=1}^{J} w_j \sigma_j^2 \\ \sigma_B^2 &= \textstyle\sum_{j=1}^{J} w_j (\mu_j - \mu)^2. \end{aligned} \tag{3.13}$$

Using the above equations, independently and identically distributed variance and the variance from proportional stratification can be compared as follows

$$\begin{bmatrix} Var(\hat{\mu}) \\ Var(\hat{\mu}_{prop}) \end{bmatrix} = \frac{1}{n} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \sigma_B^2 \\ \sigma_W^2 \end{bmatrix} . \tag{3.14}$$

Good stratified sampling is the one that achieves a reduction in within-stratum variance and results in $\sigma_B^2 \gg \sigma_W^2$ [26].

3.4.1. Latin Hypercube Sampling

Stratification approach described above might not be the best approach in high dimensional problems. Higher accuracy can be achieved by increasing the number of levels used in the stratification. In Latin Hypercube Sampling (LHS), each dimension is stratified into n equal strata. In Figure 3.1, we show Latin Hypercube Sampling of $n = 4$ points in $d = 2$ dimensions. We have 4 horizontal and 4 vertical strata and each strata gets only one point. For each dimension d, $X \sim U(0,1)^d$ is stratified into n equal strata [26]. In contrast to the standard stratified sampling method, only one point is sampled from each segment during the sampling process. Such an algorithm increases the performance of the stratification in higher dimensions. The formula of the Latin Hypercube Sampling is as follows

$$X_{ij} = \tfrac{\pi_j(i-1)+U_{ij}}{n}, 1 \leq i \leq n, 1 \leq j \leq d \tag{3.15}$$

where "$\pi_1, ..., \pi_d$ are uniform random permutations of $0, 1, ..., n-1$, $U_{ij} \sim U[0,1)$, and all the U_{ij} and π_j are independent" [26].

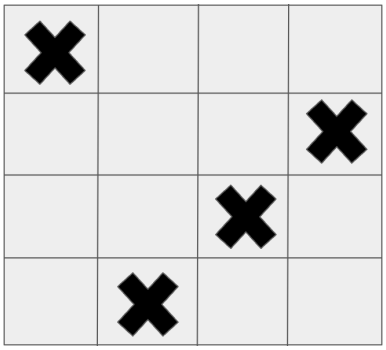

Figure 3.1. Latin Hypercube Sampling divides the region such that in each row and column there is single observation.

We can imagine the grid that is created by LHS algorithm as a chessboard. In an NxN grid, we have N rooks that are positioned in a way that none of them can capture any of the others [26]. In random sampling, new points are generated without considering the previously generated points. Therefore, one does not necessarily need to know how many sample points are needed beforehand. On the other hand, in the LHS algorithm, one must first determine how many sample points to use. Furthermore, the LHS algorithm stores each sample point's row and column information to not sample from the same row or column in the next iteration. As a result, the LHS algorithm is usually more accurate when calculating simulation statistics than is traditional Monte Carlo sampling because the entire range of the distribution is sampled more evenly and consistently. The added cost of this method is the additional memory needed to track which segments have been sampled while the simulation runs. However, this extra overhead is minor.

3.5. Combination of Different Techniques

Previous sections explain the application of Antithetic Variates, Control Variates, and Stratification techniques. Depending on the problem, high variance reduction factor values can be obtained by individual implementations of these algorithms. On the other hand, extra efficiency can be achieved by combining multiple techniques. There are studies in the literature where such a framework is applied. In [3], a framework that combines Latin Hypercube Sampling with Control Variates algorithm is proposed. In [29], the combination of control variates

stratified sampling is proposed. There are also studies where Antithetic Variates and Control Variates are combined like in [30].

3.6. Rao-Blackwell Theorem

Rao-Blackwellization is the idea of using conditional expectations to obtain an estimator with lower variance [31]. If an estimator is a good estimator, it should be close to the true value of the parameter. Let us denote the unbiased estimation of θ as $\hat{\theta}$. Since it is an unbiased estimator, the variance of $\hat{\theta}$ is equal to $\mathbb{E}[\hat{\theta}-\theta]^2$. If $\hat{\theta}$ is a biased estimator, then this value is the mean squared error (MSE) of $\hat{\theta}$. The Rao-Blackwell theorem says that an estimator with small MSE can be found if the estimator is a function of a sufficient statistics, T. Let $\theta^* = \mathbb{E}[\hat{\theta}|T]$ and $\mathbb{E}[\hat{\theta}^2] < \infty$ for all θ. Then, for all θ, the following

$$\mathbb{E}[\theta^* - \theta]^2 \leq \mathbb{E}[\hat{\theta} - \theta]^2 \tag{3.16}$$

holds. This is a strict inequality unless $\hat{\theta}$ is a function of T. It can be proven as follows

$$\begin{aligned} &\mathbb{E}[\theta^* - \theta]^2 = \mathbb{E}[\mathbb{E}[\hat{\theta} - \theta|T] - \theta] \\ &= \mathbb{E}[\mathbb{E}[\hat{\theta} - \theta|T]]^2 \leq \mathbb{E}[\mathbb{E}[(\hat{\theta} - \theta)^2|T]] \\ &= \mathbb{E}[\hat{\theta} - \theta]^2. \end{aligned} \tag{3.17}$$

It can be seen that the "Rao-Blackwellized Estimator" is improved. However, it is an unbiased improved estimator if and only if the original estimator is unbiased.

An example of Rao-Blackwellization using the Poisson process can be given as follows. Let us have a Poisson process with the average rate of the event λ. In n successive periods, $X_1, X_2, ..., X_n$ are observed events. Let us estimate the probability $e^{-\lambda}$ that in the next period, no event will be observed. A basic

estimator can be as follows. The estimator $\eta_0 = 1$ if $X_1 = 0$ and $\eta_0 = 0$ otherwise. The sum

$$S_n = \sum_{i=1}^{n} X_i = X_1 + ... + X_n \tag{3.18}$$

is the sufficient statistics for λ. Therefore, the "Rao-Blackwellized" estimator can be denoted as follows

$$\begin{aligned}
&\eta_1 = \mathbb{E}[\eta_0 | S_n = s_n] \\
&\hat{\eta_1} = \mathbb{E}[\mathbb{1}_{X_1=0} | \sum_{i=1}^{n} X_i = s_n] = P(X_1 = 0 | \sum_{i=1}^{n} X_i = s_n) \\
&= P(X_1 = 0, \sum_{i=2}^{n} X_i = s_n) P(\sum_{i=1}^{n} X_i = s_n)^{-1} \\
&= e^{-\lambda} \frac{((n-1)\lambda)^{s_n} e^{-(n-1)\lambda}}{s_n!} \left(\frac{(n\lambda)^{s_n} e^{-n\lambda}}{s_n!} \right)^{-1} \\
&= \frac{((n-1)\lambda)^{s_n} e^{-n\lambda}}{s_n!} \frac{s_n!}{(n\lambda)^{s_n} e^{-n\lambda}} = (1 - 1/n)^{s_n}.
\end{aligned} \tag{3.19}$$

The average number of events in the first n periods is $n\lambda$. If n is big, this estimator has a high probability of being close to

$$(1 - 1/n)^{n\lambda} \approx e^{-\lambda}. \tag{3.20}$$

Therefore, η_1 is an improved estimator. Moreover, basing the estimators on sufficient statistics is at least as good as the original estimator in terms of the mean squared error of the estimation.

4. VARIANCE REDUCTION FOR MARKOV CHAIN MONTE CARLO SIMULATIONS

In the literature, there are different approaches to decrease the variance of MCMC simulations. Some of the MCMC algorithms have multiple chains inside or multiple proposed values in order to achieve better precision and therefore may achieve lower variance like Multiple-try MCMC algorithm, Metropolis-Coupled MCMC or non-reversible implementations of MCMC algorithms [32, 33]. These algorithms may be considered as more complex versions of the algorithms such as Gibbs Sampler and Metropolis-Hastings algorithm. They have additional steps to run and they have more parameters to tune. The other approach is to run more chains in parallel [34,35]. If there is an available computing power to run multiple chains in parallel, one can use each of the parallelized chains at the final step and obtain the results in a faster way [36]. There is another approach to increase the precision of the chain which is to use variance reduction algorithms. These algorithms can be applied without changing the MCMC algorithm. In this study the aim is to achieve variance reduction without changing the MCMC algorithm.

The aim of reducing the variance of the MCMC algorithm is to increase the precision of the simulation. An MCMC algorithm may require a large number of iterations to achieve convergence to the stationary distribution. In theory, MCMC algorithms converge to the stationary distribution in the long run. The *long run* can be several hours or days which brings a need for more efficient implementation. The reason that MCMC simulations might require large number of iterations is that they are used to find and simulate from complex target distributions, which is not an easy task. The complexity and challenge of the problem that MCMC algorithms attempt to solve may stem from the shape of the target density, the dimension of the data that is part of the problem, the dimension of the object to be simulated, or from time restrictions. The performances of the MCMC algorithms like Metropolis-Hastings algorithm may vary depending on the

selection of the proposal distribution given the target density π. The poor selection of the proposal distribution can cause having a chain with poor convergence properties such as high autocorrelation between successive iterations in the chain, which decrease the efficiency of the chain. The existance of the autocorrelation in MCMC output implies that we need to have a larger number of simulated values to achieve the same precision with the i.i.d. simulations and it causes an increase in the computation time. The autocorrelation of the successive draws increases the importance of the initial starting point of the chain as well. By having the increased simulation size, the impact of the initial starting point can be decreased and the possibility of reaching the stationary distribution can be increased [36]. However, as the dimension of the problem increases, sampling from π becomes complicated. Simply increasing the sample size might be prohibitively costly. For this reason, an alternative approach to decrease the variance of the simulation with the same expectation is needed. This is the aim of the variance reduction algorithms for MCMC simulations. On the other hand, applying variance reduction to the MCMC output without evaluating the convergence of the chain is misleading. The original aim of the MCMC simulation is to explore the target distribution and make estimations with lowest possible errors. If the chain fails to explore the target distribution there is no meaning to apply variance reduction. Therefore, variance reduction should only be applied after the convergence of the chain is observed via multiple techniques like Gelman-Rubin diagnostics, trace plots, calculation of simulation error.

The study on variance reduction of Markov Chain Monte Carlo algorithms mostly focused on the usage of the Control Variates. There are different kinds of Control Variates proposed for Gibbs Sampler and Metropolis-Hastings sampler. Studies on other approaches like Antithetic Variates and Stratification are limited. In the previous chapter, variance reduction algorithms for Monte Carlo algorithms are given. These algorithms use the input of the simulation or another problem with a known solution in order to achieve the variance reduction. In this chapter, we first review the studies on variance reduction for MCMC algorithms in the literature in Section 4.1. In Section 4.2, we propose our methods for variance

reduction for MCMC and present their implementation details.

4.1. Literature Review

In Chapter 3, it is demonstrated that each variance reduction technique for Monte Carlo simulations has different working principles. It is also valid for MCMC simulations with an extra challenge. A given variance reduction technique might not be appropriate for a specific MCMC algorithm. There are techniques that can only be applied to Gibbs Sampler, and also there are techniques that can only be applied to Metropolis-Hastings algorithm. Therefore, in MCMC simulations, some of the variance reduction algorithms are applied in a MCMC algorithm specific way. Some of the variance reduction techniques are easy to apply while some of them are not straightforward to apply. Besides, variance reduction techniques may have extra computational costs as well. Regardless of the MCMC algorithm, one of the crucial factors is to analyze the cost of applying the variance reduction algorithm. If the variance reduction algorithm comes with a high computational cost, it might be better to avoid such an approach. Therefore, computational aspect of the variance reduction algorithm should also be considered.

4.1.1. Antithetic Variates

The idea of having negatively correlated two simulations is an effective approach to decrease the variance of the Monte Carlo algorithms as it is shown in section 3.3. This idea is applied to MCMC algorithms as well. In [37], Antithetic coupling of two Gibbs Sampler chains is proposed. In this study, authors show that it is possible to have two chains, both with stationary probability density π, and run those chains in parallel in a way that the negatively correlated chains can be obtained. We can demonstrate the proposed methodology as follows. If the two chains are denoted as X^t and Y^t and their joint probability measure can be constructed in a way that $f(X^t)$ and $f(Y^t)$ have a negative covariance, variance reduction is achieved. The coupling of the chains is done by using uniform random variate $U \sim [0, 1]$. If X^t uses a uniform U to proceed to X^{t+1}, the other chain, Y^t,

uses 1-U to proceed to Y^{t+1}. The implementation of this algorithm is simple if the problem is discrete. However, if the needed conditional distribution functions cannot be inverted analytically, the need for numerical inversion or more advanced techniques emerges which results in an increase in computing time. The estimation of the expected value using $f(X^t)$ and $f(Y^t)$ is as follows

$$\hat{\mu} = \frac{1}{T} \sum_{t=T_0+1}^{T_0+T} \frac{f(X^t)+f(Y^t)}{2}. \tag{4.1}$$

It should be noted that there are two chains in this application. In order to evaluate the performance gain, the naive Gibbs Sampler should be run twice as long as the single chain run in the antithetic coupling algorithm.

4.1.2. Rao-Blackwellization

In section 3.6, Rao-Blackwell theorem [31] is explained and an example of Rao-Blackwellized estimator for Poisson process is given. Rao-Blackwellized versions of the MCMC algorithms are widely studied in the literature.

In [38], Rao-Blackwellization of Metropolis Algorithm and Accept-Reject algorithm is given. In [39], various variance reduction techniques are discussed including Rao-Blackwellization. An example of Rao-Blackwellization for Metropolis-Hastings algorithm is demonstrated in [36]. In [36], authors denote all of the proposed values in the Metropolis-Hastings run by $\vartheta_{1,\ldots,N}$ and the values in chain output by $\theta_{1,\ldots,N}$.

The estimated mean of the chain can be calculated by using accepted ϑ_j values and number of times they are repeated within the chain as follows

$$\hat{\mu} = \frac{1}{N} \sum_{t=1}^{N} h(\vartheta_t) \sum_{i=1}^{N} \mathbb{1}_{\theta^{(i)}=\vartheta_t}. \tag{4.2}$$

The Rao-Blackwellization of the above equation is as follows

$$\frac{1}{T}\sum_{t=1}^{T} h(\vartheta_t)\mathbb{E}[\sum_{i=1}^{T} \mathbb{I}_{\theta^{(i)}=\vartheta_t}|\vartheta_1,...,\vartheta_t] \tag{4.3}$$

which is equal to

$$\frac{1}{T}\sum_{t=1}^{T} h(\vartheta_t)(\sum_{i=1}^{T} \mathbb{P}(\theta^{(i)}=\vartheta_t|\vartheta_1,...,\vartheta_t)). \tag{4.4}$$

This is straightforward since

$$\begin{aligned}&\mathbb{E}[\sum_{i=1}^{T} \mathbb{I}_{\theta^{(i)}=\vartheta_t}|\vartheta_1,...,\vartheta_t] = \sum_{i=1}^{T} \mathbb{E}[\mathbb{I}_{\theta^{(i)}=\vartheta_t}|\vartheta_1,...,\vartheta_t] \\ &= \sum_{i=1}^{T} \mathbb{P}(\theta^{(i)}=\vartheta_t|\vartheta_1,...,\vartheta_t).\end{aligned} \tag{4.5}$$

Finally,

$$\frac{1}{T}\sum_{t=1}^{T} h(\vartheta_t)\left(\sum_{i=1}^{T} \mathbb{P}(\theta^{(i)}=\vartheta_t|\vartheta_1,...,\vartheta_t)\right) \tag{4.6}$$

enjoys a smaller variance as it is shown in [36]. The computation of $\mathbb{P}(\theta^{(i)} = \vartheta_t|\vartheta_1,...,\vartheta_t)$ is based on the integration of the uniform variates used to make the choice between the proposed value and the current value of the Markov chain.

4.1.3. Post Processing the Chain Output

There are techniques which use the simulation output itself in order to reduce the variance. In the Metropolis-Hastings algorithm, the rejected proposal values do not take part in the final chain values. Using these values in order to reduce the variance of the chain is studied in various papers. The most simple yet effective usage of all of the proposal values is *"Averaging Method"*, which is discussed in [6, 36]. The proposed technique is the weighted sum of the simulation output and the proposal values. The weights are the acceptance ratios that are calculated in each step of the chain by dividing the proposed density by the density of the current value of the Markov chain. If θ_t is the current value of the Markov chain and ϑ_t

the proposed value, to be accepted (as θ_{t+1}) with probability α, the following

$$\hat{\mu} = \frac{1}{T}\sum_{t=1}^{T}\{\alpha_t h(\vartheta_t) + (1-\alpha)h(\theta_t)\} \tag{4.7}$$

should most often give an estimate with lower variance.

Another post processing based variance reduction technique is based on subsampling the Gibbs Sampler [40]. The idea is that subsampling the output of a Gibbs Sampler can increase the efficiency of marginal estimators of the chain if the subsampling method is tied to the actual realized updates made. This can be considered as a developed version of the thinning of the chain.

4.1.4. Control Variates for MCMC Algorithms

The idea of the control variates algorithm is the same for MCMC simulations as well. A variate with known expected value reduces the variance of the simulation if it is correlated with the simulation output. The main challenge of using control variates to achieve variance reduction in MCMC algorithms is that it is harder to find a control variate which is highly correlated with the simulation output.

A good example for control variates for MCMC algorithms can be found in [41]. They propose the following Control Variates to be used in the Metropolis-Hastings algorithm.

In the Metropolis-Hastings algorithm, there are accepted and rejected proposals. Rejected proposals are thrown away. The idea behind the proposed Control Variates in [41] is to make use of not only accepted proposals, but also the rejected proposals along with the acceptance ratio of the chain. In their study, they propose five different Control Variates for Metropolis-Hastings algorithm and compared their performance by conducting empirical experiments. Authors show that the following control variate, CV_1, is the most successful control variate amongst other proposed Control Variates in their study.

If we let x be the current state and y the proposal,

$$CV_1 = \frac{AR(x,y)}{1+AR(x,y)}(f(x) - f(y)) \tag{4.8}$$

where

$$AR(x,y) = \frac{\pi(y)q(x|y)}{\pi(x)q(y|x)} \tag{4.9}$$

is the acceptance ratio.

The acceptance probability is the following

$$\alpha(y|x) = min(1, AR(x,y)). \tag{4.10}$$

The remeaning proposed control variates in [41] are as follows

$$CV_2 = (\alpha(y|x) - \mathbb{1})f(y) \tag{4.11}$$

where $\mathbb{1}$ is an indicator if the proposed value is accepted or not. A similar control variate to CV_2 can be created by using $f(x)$ as follows

$$CV_3 = (\alpha(y|x) - \mathbb{1})f(x). \tag{4.12}$$

Final set of two remaining control variates are as follows

$$\begin{aligned} CV_4 &= (1-\mathbb{1})\alpha(y|x)f(x) - \mathbb{1}(1-\alpha(x|y))f(y) \\ CV_5 &= (1-\mathbb{1})\alpha(y|x)f(y) - \mathbb{1}(1-\alpha(x|y))f(y). \end{aligned} \tag{4.13}$$

The variance reduction performance of above control variates are limited. In [41], it is shown that the highest variance reduction factor value is obtained by using CV_1 alone which is 1.84. Authors did not use all five control variates together.

Each control variate is evaluated individually. In this thesis, the performance when using these control variates as Multiple Control Variates will be studied in Section 4.2.1.4 below.

A major study for variance reduction for Gibbs Sampler is proposed in [42]. In this study, it is stated that for any real-valued function G_j defined on the state space of a Markov chain X_n, the functions

$$U_j(x) := G_j(x) - E[G_j(X_{n+1})|X_n = x] \tag{4.14}$$

have zero mean with respect to the stationary distribution of the chain [43]. The goal is to estimate the posterior mean μ^i of $x^{(i)}$. Basis function can be defined as $G_j(x) = x^{(j)}$ for all components j for which $PG_j(x) = E[X_{n+1}^{(j)}|X_n = x]$ is computable in closed form. The corresponding control variates is $U_j = G_j - PG_j$.

Application of Multiple Control Variates has not been studied widely in the literature for MCMC algorithms. Multiple Control Variates for MCMC are mainly discussed in [42].

4.2. Proposed Methodology

In the previous section, we reviewed the literature on Variance Reduction for Markov Chain Monte Carlo algorithms. In this section, we describe and demonstrate our approach to implement Antithetic Variates, Control Variates and Latin Hypercube sampling algorithms to Random-Walk Metropolis-Hastings algorithm.

In the following two chapters, we demonstrate our experiments. In Chapter 5, we start with the details of our experimental setup, define the problem of detecting the convergence of Markov Chain Monte Carlo algorithms and explain our approach to evaluate the convergence of Markov Chain Monte Carlo algorithms. Chapter 6 shows our results of implementing Variance Reduction algorithms to chains with good convergence properties. We evaluate the quality of the stan-

dard error estimations of the chains and conduct experiments via the calculation of coverage probabilities of the chains. We also compare the Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) of the estimated values obtained from naive and variance reduction applied simulations. Finally, we analyze the impact of variance reduction algorithms on coverage probabilities and RMSE/MAE values of the estimations. This step is essential because it will demonstrate if variance reduction algorithms introduce bias to estimated values and provide coverage probabilities as high as the naive simulation estimations.

The remaining of this chapter and the following two chapters include Variance Reduction methods for the Metropolis-Hastings algorithm that, up to our knowledge, were not tried for MCMC algorithms yet. The main new points that up to our knowledge were not suggested in the literature are:

- We propose using proposal steps as Inner Control Variates, as shown in Section 4.2.2.1.
- We use the Multiple Control Variates algorithm and combine different types of Control Variates.
- We propose using the Antithetic Variates algorithm for the Metropolis-Hastings algorithm, as shown in Section 4.2.2.
- We test the performance of variance reduction algorithms in problems with varying difficulties in a controllable experimental setup. We experiment with different types of variance reduction algorithms and also combine multiple variance reduction algorithms to achieve better performance.
- We evaluate the quality of the standard error estimations, which is neglected by the other studies in the literature in the context of variance reduction for Markov Chain Monte Carlo algorithms. We show that standard error estimations can be biased for Markov Chain Monte Carlo algorithms.

4.2.1. Implementing Variance Reduction Techniques for Random-Walk Metropolis-Hastings Algorithm

Each MCMC algorithm takes different input parameters and performs different calculations. The difference in MCMC algorithms affects the strategy for variance reduction, which is achieved via algorithms like Antithetic Variates, Control Variates, Stratification. In this section, we discuss how to implement these techniques to the Random-Walk Metropolis-Hastings algorithm.

The Random-Walk Metropolis-Hastings algorithm uses a set of calculations and input values to move from the current state to the next state. A random walk value from the proposal distribution is used to move from step i to step $i+1$ in a chain. The density of the proposed value is compared with the most recent value at the chain, and it is accepted with a certain probability. Two input values are used during the process of moving from step i to step $i+1$.

- Random-Walk step.
- U $\sim$ Uniform[0, 1] variate that is used to accept / reject proposals.

These values are generated using different distributions than the target distribution itself. Random-Walk values are created using the proposal distribution and the acceptance probability value is created using Uniform $\sim$ [0,1] random variable. There are two strategies to create these values. They can be created in each step $1, 2, ..., N$ within the chain or they can be created at once and they can be given as an input parameters of the algorithm. Both strategies generate a correct Markov Chain as for both methods values are generated randomly from the correct distributions. However, generating them before and storing them as a matrix used as input to the MCMC simulation function has some advantages when the aim is variance reduction. In this study, aforementioned values are created at once and random walks are created using this input matrix which is denoted as Z. The acceptance probabilities for each step are also simulated at once and given as an input to the algorithm.

The proposal distribution used might be for example the Gaussian distribution, t distribution, uniform distribution etc. Let us use a Gaussian distribution as a proposal distribution with mean μ standard deviation σ to express the idea. Two different strategies can be followed when creating the proposal value. If the current value of the chain is X^t, a proposal distribution with $\mu = X^t$ and standard deviation σ can be used. The other strategy is to draw a value from a distribution with the same standard deviation but $\mu = 0$ and add the resulting value to X^t in order to create a proposal for X^{t+1}. In either way, the mean value of the proposal is the current value of the chain.

Given the σ value, proposal steps are created as a matrix of dimension N x D. If the latter strategy for creating proposals is used, each column of the input matrix Z has mean zero and standard deviation σ. These values are added to the current state of the chain at each step. In this way, the proposal value using the latest value in the chain can be created. In the case of choosing t distribution as the proposal distribution, Z matrix will still have zero mean since the t distribution is centered at the origin. In addition, if the uniform distribution is chosen as the proposal distribution, arranging the upper bound a and lower bound $-a$ of the uniform distribution will also give mean zero columns in the Z matrix. This matrix will be of main interest for the following Variance Reduction Techniques which are

- Antithetic Variates
- Control Variates
- Latin Hypercube Sampling.

In addition to 2 methods from the literature that we discussed in previous sections, which are Control Variates proposed in [41] and the "Averaging Method" discussed in [36]. In this thesis, we use Antithetic Variates, Control Variates and Latin Hypercube Sampling methods as they are discussed in detail below. All together, there are 5 methods to be implemented for simulation problems with fixed dimensions like typically used for posterior distributions of Bayesian models with small or moderate dimensions.

4.2.1.1. Implementing Antithetic Variates Algorithm. Let us start with the application of Antithetic Variates. In Section 4.1.1, we demonstrated the Antithetic Variates algorithm for Gibbs Sampler as it is proposed in [37]. A similar idea can be applied to Random-Walk Metropolis-Hastings algorithm as well. Let us use normal distribution as the proposal distribution of Random-Walk Metropolis-Hastings algorithm. In each step, a random draw from the proposal distribution is sampled and added to the current position of the chain. This value is accepted or rejected depending on the acceptance probability. Let us denote the random draw from the Gaussian proposal distribution as N_p, which is a symmetric distribution centered at the origin. If another chain is run with proposal value $-N_p$, antithetic chains will be obtained and variance can be reduced by taking the average of these two chains. We can also store the proposal steps in an input matrix, Z. Another input matrix can easily be the negative signed Z, $-Z$, since the input values come from a symmetric distribution centered at the origin. If two different chains are run using these two input values, the average of the result of those two chains are expected to reduce the variance as it is explained in section 3.3 and 4.1.1. The main challenge that limits the performance of the Antithetic Variates algorithm here is due to the acceptance-rejection process of the Metropolis-Hastings algorithm. By using Z and $-Z$ it is known that at each step proposal values go in the opposite directions. However, the accepted values in two chains may not have corresponding values that go in opposite directions. This may reduce the negative correlation between the two chains because negative correlation is achieved via opposite signed proposal values. When opposite signed proposal values are not accepted in both of the chains, negative correlation cannot be achieved because chains are no longer going to opposite directions in that step. This also means that as the ratio of accepted proposal values increases, we are more likely to obtain chains with higher negative correlation.

Figure 4.1 shows the implementation of the Antithetic Variates algorithm. We run two chains with input matrices Z and $-Z$ and take the average of the output of the two chains.

Require INPUT: *Sample Size* N, *Dimension* D, *Covariance Matrix* Σ;

$Z_{1,\ldots,N} \sim Normal(\mu = 0, \Sigma)$
Y_1 = Random-Walk Metropolis Hastings with input matrix Z
Y_2 = Random-Walk Metropolis Hastings with input matrix -Z
$Y = (Y_1 + Y_2)$ / 2
RETURN Y

Figure 4.1. Algorithm 1: Antithetic Variates.

4.2.1.2. Implementing Latin Hypercube Sampling Algorithm. The creation of the Z matrix is performed simply by taking a sample from the proposal distribution. However, the Z matrix can be created in a different way as well. By introducing stratification into the process of creation of the Z matrix, an extra gain in precision might be obtained. Regardless of the type of the proposal distribution, the cumulative distribution function of the proposal distribution has an interval [0,1]. A sensible strategy to create the Z matrix is to generate, regardless of the type of the proposal distribution, $Uniform \sim [0, 1]$ random values and to transform these value with the inverse CDF of the proposal. By using stratification in the first step we can obtain variance reduction. One of the approaches to perform that is to use Latin Hypercube Sampling. By Latin Hypercube Sampling, we can create values in the interval $[0, 1]$ with expected value 0.5. The process of using Latin Hypercube Sampling to create the Z matrix can be explained as follows.

- Generate the LHS matrix with dimension D, row number N. Denote it by LHS_{NxD}.
- Find the inverse of the cumulative distribution function of the proposal distribution, $F_X^{-1}(x)$.
- Compute $Z = F_X^{-1}(LHS_{NxD})$. The computed random variables in Z has distribution $F_X(x)$.

4.2.1.3. Inner Control Variates for Metropolis-Hastings Algorithm. Another variance reduction technique that is applied to MCMC simulations is Control Variates. In Section 4.1.4, a set of Control Variates for Random-Walk Metropolis-Hastings algorithm was given which are proposed in [41]. All of the Control Variates proposed for the Random-Walk Metropolis-Hastings algorithm in that study make use of random walks and/or the acceptance probability of the chain together to create the Control Variates. However we can always use the input variables Z as Control Variates that require next to no extra computations.

In the Metropolis-Hastings algorithm, we have to define the proposal steps as it is discussed before. We defined the input matrix Z which has the proposal steps for our algorithm. The columns of the input Z matrix are created by taking a sample from a known distribution. This means that the expected values of these Z values are known and the Z values can be used as Control Variates. In a D dimensional problem, each of the input variates can be used as a Control Variate so first D Control Variates are "free". Furthermore, these input values can be used in different forms. The sum of all Z columns or any other linear combination of Z columns can be Control Variate. In addition, the squares of each column can be used as control variates since it is known that they will become a chi-squared random variable if they are coming from Gaussian proposal distribution. Let us denote one column of the Z matrix that is created using Gaussian proposal distribution as X. If the squared value of X is used as control variate, its expected value can be calculated using $\mathbb{E}[X^2] = Var(X) + \mathbb{E}[X]^2$. The same approach can be applied to t proposal distribution as well. Square of the t-random variable with ν degree of freedom follows the F distribution $\chi^2_{1,\nu}$. The expected value of $\chi^2_{1,\nu}$ is equal to $\nu/(\nu-2)$. Since it's expected value is known, it can be used as another control variate.

We can always reach the highest possible VRF that can be obtained from the input matrix by using the D columns of Z matrix and the squared Z values together as multiple control variates after finding optimal coefficients for these control variates.

Another set of control variates appears when Latin Hypercube Sampling is used in order to create the input matrix, Z. Latin Hypercube matrix of dimension D and row number N, $LHS_{N,D}$ is created in order to create the input $Z_{N,D}$ matrix. In this case the columns of the $LHS_{N,D}$ matrix have the expected value of 0.5 and they can be used as another set of control variates as well. Moreover, another variate with known expectation is the Uniform random variate U $\sim$ Uniform[0,1] that is created to accept / reject proposals. We can store this variate in each iteration and use it as a Control Variate as well.

4.2.1.4. Implementing Multiple Control Variates Algorithm. In Section 4.2.1.3, we showed several "free" Inner Control Variates. When we combine them with the Control Variates proposed in [41], we end up with having several Control Variates for our disposal. We can use each of the Control Variates using Multiple Control Variates algorithm as it is discussed in Section 3.2. The full list of Control Variates that are used in Multiple Control Variates algorithm are as follows.

- Individual columns of Z matrix, Z_i.
- Squares of the columns of Z matrix, chi-square value $\tilde{\chi}_i^2$ or the square of t value, $\chi_{1,\nu}^2$.
- Control Variates proposed in [41], CV_{T_i}.
- U $\sim$ Uniform[0,1] that is created to accept / reject proposals, U_{CV}.
- Individual columns of LHS matrix, LHS_D.

The final equation to find the optimal coefficients for the above control variates using gaussian proposal distribution is as follows

$$\begin{aligned}\hat{\mu_c} = Y + c_a(Z_1 - \mathbb{E}[Z_1]) + .. + c_f(Z_d - \mathbb{E}[Z_d]) + c_g(\tilde{\chi}_1^2 - \mathbb{E}[\tilde{\chi}_1^2]) + \\ .. + c_l(\tilde{\chi}_d^2 - \mathbb{E}[\tilde{\chi}_d^2]) + c_m(CV_{T_1} - 0) + .. + c_p(CV_{T_5} - 0) + \\ c_r(U_{CV} - 0.5) + c_t(LHS_1 - 0.5) + ... + c_y(LHS_d - 0.5).\end{aligned} \tag{4.15}$$

The optimal coefficient c_i for each Control Variate CV_i is calculated as it is shown in Section 3.2.

In the experiments, Antithetic Variates, Control Variates, Latin Hypercube Sampling are used in order to achieve the variance reduction. The Averaging Method described in section 4.1.3 is also implemented. Besides, Antithetic Variates, Control Variates and Latin Hypercube Sampling methods can be combined together to achieve better variance reduction. The methodology for using them together is described below.

4.2.1.5. A Method to Combine Multiple Variance Reduction Algorithms. As it is demonstrated above, each variance reduction algorithm uses a different approach. This makes it possible to combine these techniques in order to have a better performing variance reduction method. Both of the Latin Hypercube Sampling and Antithetic Variates algorithms has the input matrix at the center of their implementations. The former changes the way input matrix is created and the latter introduces an antithetic chain with the help of oppositely signed input matrix.

These approaches can be used together and Antithetic Variates algorithm can use input matrices created by using Latin Hypercube Sampling. After creating the Z matrix in this way we can run two different chains with using Z and $-Z$ as input parameters. If we want to combine only Antithetic Variates and Latin Hypercube Sampling we can simply take the average of the output of two different chains. On the other hand, we can also add Control Variates algorithm at this step. Instead of taking the average of the naive simulation outputs of two chains, we can first apply Control Variates to simulation outputs and than return these values to Antithetic Variates algorithm. By using such a method, we can combine different variance reduction algorithms in order to potentially achieve higher variance reduction and more precise estimations. The proposed method is summarised in Figure 4.2.

Require INPUT: *Sample Size* N, *Dimension* D, *Covariance Matrix* Σ;

LHS_{NxD} is created

$Z_{1,...,N} \sim \mathrm{F}_N^{-1}(LHS_{NxD})$

$Z_{1,...,N} \sim Normal(\mu, \Sigma)$

Y_1 = Output of Control Variates Algorithm of Random-Walk Metropolis Hastings with input matrix Z, where Z is added to previous value of the chain.

Y_2 = Output of Control Variates Algorithm of Random-Walk Metropolis Hastings with input matrix -Z, where -Z is added to previous value of the chain.

$Y = (Y_1 + Y_2) / 2$

RETURN Y

Figure 4.2. Algorithm 2: Using Antithetic Variates, Control Variates and Latin Hypercube Sampling together.

5. EXPERIMENT SETUP

The purpose of this study is to explore the performance of variance reduction algorithms in Markov Chain Monte Carlo simulations. The main variance reduction algorithms of interest are Antithetic Variates, Control Variates and Stratification, which are general variance reduction algorithms. We studied the Random Walk Metropolis-Hastings algorithm with different types of proposal distributions. These proposal distributions are uniform distribution, t distribution, and Gaussian distribution. In order to evaluate the performance of variance reduction techniques for MCMC algorithms, we created an experimental setup that has several problems. These problems are created in a way that enables us to extensively analyze the performance of MCMC algorithms and variance reduction techniques. In this study, we use examples inspired by Bayesian-Statistic applications. We use mixture of multinormal distributions and use MCMC algorithms to obtain the random vector $\mathbf{X}$, and estimate the mean, variance, and correlation of the random vector $\mathbf{X}$. In other words, we are interested in the first and second moments of the mixture distribution, which is also crucial for Bayesian applications like the posterior of the Bayesian regression model. We would like to select an example that has simple formulas for the exact results. This allows us to compare the size of the exact error among different experiments. For this reason, along with the standard error of the estimated mean, standard error values of estimated variance and estimated correlation of the random vectors are calculated and used to create confidence intervals for estimations. We aim to find variance reduction algorithms that help us to decrease the standard error of these estimations and increase the precision of the estimated values. Before implementing the variance reduction algorithms, we need to define a sensible experimental setup, create a set of problems according to the experimental setup, use MCMC algorithms for these problems, evaluate the convergence properties of the chains, and select chains with good convergence properties. In this chapter, we will describe the steps taken before applying variance reduction algorithms and the first step is defining our experimental setup where we use mixture of multinormal distributions with different set of parameters.

A mixture of multinormal distributions has several parameters. It can consist of M distributions each of dimension D. Thus each distribution has its own parameters and also a probability. We demonstrate the exact solution of the mean, variance-covariance matrix and correlation matrix of the mixture of multinormal distributions with mean vectors μ_i, diagonal variance-covariance matrices with unit variance values denoted by Σ_i and probability values α_i where $i = 1, 2, .., M$ to obtain the mixture mean, $\bar{\mu}$ and the mixture variance-covariance matrix, $\Sigma_{\mathbf{X}}$. In our experiments, we take the simple case where $M = 2$. The exact solution for the mean vector, variance-covariance matrix, and correlation values are calculated as follows

$$\mathbb{E}[\mathbf{X}] = \bar{\mu} = \sum_{i=1}^{M=2} \alpha_i \mu_i. \tag{5.1}$$

Using $\bar{\mu}$, we can find the exact solution of the variance-covariance matrix as follows

$$\Sigma_{\mathbf{X}} = \sum_{i=1}^{M=2} \alpha_i \Sigma_i + \sum_{i=1}^{M=2} \alpha_i (\mu_i - \bar{\mu})(\mu_i - \bar{\mu})^T. \tag{5.2}$$

By using the exact solution of the variance-covariance matrix, we can calculate the correlation matrix as follows

$$\text{corr}(\mathbf{X}) = \big(\text{diag}(\Sigma_{\mathbf{X}})\big)^{-\frac{1}{2}} \; \Sigma_{\mathbf{X}} \; \big(\text{diag}(\Sigma_{\mathbf{X}})\big)^{-\frac{1}{2}} \tag{5.3}$$

where $\text{diag}(\Sigma_{\mathbf{X}})$ is the matrix of the diagonal elements of $\Sigma_{\mathbf{X}}$.

In our experiments, we estimate these values by using MCMC simulations to demonstrate the abilities of MCMC algorithms and variance reduction techniques. An MCMC algorithm simulates a sample of random vectors following the given density. In a simulation we are interested to estimate a function of that random vector. In our experiments we estimate the expectation and the variance of the marginals and the correlation between the marginals. The formulas of these MCMC-estimates are presented in equations 5.4, 5.5, and 5.6. We evaluate the

density $f(x_i)$ used in these equations just once for each $i = 1, ..., N$ and use the generated sample to make estimations for different functions. For the expectation of the marginals, the calculation is simple. To estimate the mean value of the j^{th} marginal of the random vector, we use the following

$$\hat{\mu_j} = \sum_{i=1}^{N} x_{ij} f(x_i). \tag{5.4}$$

For the variance of the marginals, calculation is performed differently. We first estimate the second moment of each marginal of the random vector and use the estimated mean value in order to estimate the variance. The equation for estimating the variance of the j^{th} marginal of the random vector is

$$Var(X_j) = \sum_{i=1}^{N} x_{ij}^2 f(x_i) - \hat{\mu_j}^2. \tag{5.5}$$

For the correlations of the marginals, we start from calculating the covariance of the marginals. In order to find the covariance between the k^{th} and j^{th} marginals, we use the x_j and x_k and multiply them and also use their estimated mean values like shown in equation 5.6. After obtaining the covariance estimation, we can calculate the correlation of marginals by using the estimated variance and estimated covariance values of the marginals as follows

$$\begin{aligned} Cov(X_{jk}) &= \sum_{i=1}^{N} x_{ij} x_{ik} f(x_i) - \hat{\mu_j}\hat{\mu_k} \\ Cor(X_{jk}) &= \frac{Cov(X_{jk})}{\sqrt{Var(X_j) Var(X_k)}}. \end{aligned} \tag{5.6}$$

Estimated values like shown above come with standard error values and the typical approach to decrease the standard error is to increase the sample size. In our study, we take a different approach and use variance reduction algorithms to decrease the standard error of the estimated values. We discussed the implementation details of the variance reduction algorithms in Chapter 4 using the expected value of the first marginal as an example. We can also apply variance reduction algorithms to

reduce the standard errors of the variance estimate and the correlation estimates. However, it is essential to implement variance reduction algorithms at the correct step and in the correct way. For example, the Multiple Control Variates algorithm should be applied to each estimated value separately after obtaining the chain output. However, we take a different approach for the Latin Hypercube Sampling algorithm and use stratified proposal steps inside the Random-Walk Metropolis-Hastings algorithm. It impacts the output of the simulation and, therefore, all of the estimated values. Hence, we need to be careful about our implementation because obtaining the wrong estimation with a lower standard error is useless. When using the Antithetic Variates algorithm in our experiments, we have two chains, denoted by C_1 and C_2. We can calculate the estimations separately for each chain, then take the average of the estimations to complete the Antithetic Variates algorithm [44]. On the other hand, if we take the chains' average first and obtain the chain $C_3 = (C_1 + C_2)/2$, we can only use it to estimate the mean value because both approaches are equivalent in the case of estimating mean value. However, if we use C_3 to estimate the variance or the correlation, such an approach will return the wrong estimations.

Even though we implement the variance reduction algorithms correctly, if the MCMC algorithm fails to provide accurate estimations, we cannot gain anything by implementing variance reduction algorithms. Depending on the problem, the MCMC algorithm might fail to provide a chain with good convergence properties, leading to wrong estimations. Therefore, we must carefully tune the MCMC algorithm parameters to obtain a chain with good convergence properties. In our experimental setup, we create problems with varying complexities by using different dimensions, mean vectors, and mixture weight values for mixture components, which constitute the input parameters of the experiments. Each experiment presents different challenges to the MCMC algorithm.

The following sections in Chapter 5 and Chapter 6 explain our findings and experiments in detail. Let us start by explaining the selection process of input parameters of the experiments.

5.1. Input Parameters of Experiments

The general idea of the experimental setup is demonstrated above and it is shown that several parameters are part of it. Therefore, we need to assign sensible values to these parameters to create our experimental setup. In this section, we discuss the aim of the experiments, how should we define the input parameters of the experiments in order to serve that aim and why it is important to choose sensible input parameters. Let us start with the aim of the experiments. The aim of creating the problems used in the experiments is to analyze the performance of MCMC algorithms in a way that we can understand their behavior and capabilities in different conditions. These problems should be defined carefully and they should serve for the following purposes.

- There should be problems with varying difficulty levels in a way that effects of different parameters of the problem can be analyzed. Such problems enable us to understand the capabilities and the limits of the Random-Walk Metropolis-Hastings algorithm.
- The problems should be complex enough so that a Random-Walk Metropolis-Hastings algorithm with a single set of proposal distribution parameters will not be able to solve them all. The poor selection of the parameters of MCMC algorithms should yield a chain with poor convergence properties.

Problems with these characteristics are crucial in order to evaluate the performance of MCMC algorithms. It is possible to define the parameters of the problems so as to reflects our needs. In the experimental setup explained above, we are able to change mixture components and therefore choose the distance between the modes, mixture weights, and the dimension of the distributions. We can designate problems with the above properties if we define them systematically. To explore the effect of each of these parameters on the performance of the MCMC algorithms, we should have experiments where only one of these parameters is changed and the rest of them remains the same. In Table 5.1, problems that are created with such concern are listed. The list of the mean vectors of the mixture

components, mixture weights, and Problem ID are given. Problem ID is simply the abbreviation of the problem itself. It has the dimension of the experiment, D, the mean vectors of the mixture components, M, and weight of the first mixture component, W. For example, *D10M01W05* means that this experiment is 10 dimensional, has the mean vectors of the first and second mixture components $(0,0,0,0,0,0,0,0,0,0) - (1,1,1,1,1,1,1,1,1,1)$, has mixture weight for the first component 0.5. Note that in dimension D, the covariance matrix of each of the mixture distributions is selected as a diagonal matrix with dimension D and diagonal entries of 1. It can be seen that there are problems in various dimensions and mixture weights. We expect that finding solutions to the problems will become harder as the dimension of the problem increase, the distance between the modes increase and mixture components have unequal weights.

A natural question to be asked here is what can be a sensible strategy to solve these problems. The Random-Walk Metropolis-Hastings algorithm will be the algorithm to solve these problems because it is one of the most widely used algorithms and the simplicity and versatility of this algorithm make it one of the best options [45]. By section 2.2.1.2, we know that there are different types of proposal distributions for the Random-Walk Metropolis-Hastings algorithm and the most common choice for proposal distribution is Gaussian distribution centered around the latest value of the chain [45, 46]. In our experiments, the main proposal distribution of interest is Gaussian distribution, but t and Uniform distributions are also used as proposal distribution for research purposes. Regardless of the type of the proposal distribution, the common challenge is to decide its parameters, the *scale parameter*, which governs the step size of the proposal value. Therefore, we have to find good scale parameters to solve the problems in Table 5.1. Poor selection of these parameters leads to chains with poor convergence properties. A chain with poor convergence properties is not preferable to be used as a solution because it indicates that it takes too long for the chain to the stationary distribution. As a result, the estimations will not be close to unbiased estimations with limited sample size. Furthermore, we cannot apply variance reduction algorithms to such a chain because variance reduction techniques do not remove the bias in

the parameter estimations. Therefore, we need to define a method to find the chains with good convergence properties. Our approach for finding such chains is explained in the next section.

Table 5.1. Experiments for mixture of 2D, 4D and 10D multinormal distributions.

Problem ID	**Mixture Means**	**Mixture Weights**
D2M01W05	(0, 0)-(1, 1)	0.5-0.5
D2M01W08	(0, 0)-(1, 1)	0.8-0.2
D2M02W05	(0, 0)-(2, 2)	0.5-0.5
D2M02W08	(0, 0)-(2, 2)	0.8-0.2
D2M04W05	(0, 0)-(4, 4)	0.5-0.5
D2M04W08	(0, 0)-(4, 4)	0.8-0.2
D2M023W0504	(0, 0)-(2, 2)-(3, 3)	0.5-0.4-0.1
D2M025W0504	(0, 0)-(2, 2)-(5, 5)	0.5-0.4-0.1
D4M01W05	(0, 0, 0, 0)-(1, 1, 1, 1)	0.5-0.5
D4M01W08	(0, 0, 0, 0)-(1, 1, 1, 1)	0.8-0.2
D4M02W05	(0, 0, 0, 0)-(2, 2, 2, 2)	0.5-0.5
D4M02W08	(0, 0, 0, 0)-(2, 2, 2, 2)	0.8-0.2
D4M04W05	(0, 0, 0, 0)-(4, 4, 4, 4)	0.5-0.5
D4M04W08	(0, 0, 0, 0)-(4, 4, 4, 4)	0.8-0.2
D10M01W05	(0,0,..,0,0)-(1,1,..,1,1)	0.5-0.5
D10M01W08	(0,0,..,0,0)-(1,1,..,1,1)	0.5-0.5
D10M02W05	(0,0,..,0,0)-(2,2,..,2,2)	0.5-0.5
D10M02W08	(0,0,..,0,0)-(2,2,..,2,2)	0.5-0.5
D10M04W05	(0,0,..,0,0)-(4,4,..,4,4)	0.5-0.5
D10M04W08	(0,0,..,0,0)-(4,4,..,4,4)	0.5-0.5

5.2. Selection of Chains with Good Convergence Properties

Diagnosing the convergence of the MCMC algorithm is a challenging task because the convergence is in distribution, not to a point, which makes it harder to evaluate and validate the convergence [47]. An ideal solution to assess the conver-

gence would be to have an analytical tool to compute the convergence rate of the MCMC algorithms and use this rate to decide when the chain reaches stationary distribution. In [48], such methodology is proposed for the Metropolis algorithm, which creates polynomial time convergence bounds. However, these techniques involve laborious calculations and it is unfeasible to apply these calculations in high dimensions or when there are multiple chains. Therefore, previous studies rely mostly on practical tools to evaluate the convergence of the MCMC algorithms and determine the ones with good convergence properties [11, 49]. These practical tools apply various techniques to the output of the MCMC algorithm and provide an idea about the convergence of the MCMC algorithm. However, they cannot prove the convergence by themselves. They are preferred because "a weak diagnostic is better than no diagnostic at all" [49]. Although they are not powerful enough to prove the convergence of the algorithm, they can help us to determine the chains that are failed to converge [50]. Some of these methods are discussed in Section 2.4. Each method uses a different approach to evaluate different convergence properties of the chain. Therefore, we cannot rely on a single method. By combining multiple "weak diagnostics tools", we can have a better and more reliable idea about the convergence properties of the chain. For this reason, we combined multiple techniques together in order to evaluate the simulation output and select the chains with good convergence properties.

In this study, we use Gelman-Rubin diagnostics, evaluation of acceptance ratio and graphical methods in order to determine the chains with good convergence properties. The following properties are an indication of convergence to the stationary distribution.

- Calculation of Gelman-Rubin Diagnostics value and having chains with $R \approx 1$.
- Convergence properties that are observed via trace plots like good mixing property.
- Sensible acceptance ratios.

These are the desired properties of the chain regardless of the type of the proposal distribution. The meaning of the *sensible acceptance ratio* is having a chain with not too high or too low acceptance ratios. There are different studies that propose optimal values for acceptance ratios [51]. The proposed acceptance ratio to be aimed at in [51] is 23.6% and they proposed tuning the parameters of proposal distribution to reach 23.6% acceptance ratio as a heuristic method. There are also studies that show the pitfalls of tuning the acceptance ratio by using static values [52].

The parameter that impacts the acceptance ratio of the Metropolis-Hastings algorithm is the scale parameter of the proposal distribution. For example, if the acceptance ratio is too low, it leads to a chain that is unable to reach the stationary distribution without being too costly to run.

The root cause of having a chain with low acceptance ratios might be the size of the proposed random walk steps, which is decided by the scale parameter. In addition, large scale parameters are expected to give rise to proposals that are more prone to be rejected due to overshooting the high probability region. Therefore, seeking an optimum value for scale parameter which enables us to explore the state space with a reasonable proportion of time and with good convergence properties is critical. Note that the scale parameter of the Gaussian distribution is standard deviation, σ, for Uniform distribution it is calculated as $\sqrt{(a--a)^2/12}$ given that $U \sim Uniform[-a, a]$ and finally for t distribution it is equal to $\sqrt{\sigma_t^2 \frac{\nu}{\nu-2}}$ where ν is degrees of freedom. Each of the proposal distributions used in the experiments is symmetric with respect to 0.

There might be cases where a chain has a sensible acceptance ratio but still has poor mixing property or is over sensitive to the initial position of the chain. Therefore, we need to define an algorithm that gives us an optimal set of scale parameters by analyzing the chain's acceptance rates and the aforementioned properties to solve the problems given in Table 5.1. The process to select these parameters among candidate values is explained the next section.

Table 5.2. Candidate scale parameters for experiments.

Problem ID	**Proposal Type**	**Grid of Scale Parameters**
D2M01W05	Gaussian	(0.55, **0.7, 1**, 1.73, 2.23, 2.82, 3.16)
D2M01W08	Gaussian	(0.55, **0.7, 1**, 1.73, 2.23, 2.82, 3.16)
D2M04W05	Gaussian	(2.23, 2.82, **3.16**, 3.5)
D2M04W08	Gaussian	(2.23, 2.82, **3.16**, 3.5)
D2M023W0504	Gaussian	(0.55, 0.7, **1**, **1.73**, 2.23, 2.82, 3.16)
D2M025W0504	Gaussian	(0.55, 0.7, 1, 1.73, **2.23**, 2.82, 3.16)
D4M01W05	Gaussian	(0.55, **0.7**, **1**, 1.73, 2.23, 2.82, 3.16)
D4M01W08	Gaussian	(0.55, **0.7**, **1**, 1.73, 2.23, 2.82, 3.16)
D4M02W05	Gaussian	(0.55, **0.7, 1, 1.73**, 2.23, 2.82, 3.16)
D4M02W08	Gaussian	(0.55, **0.7**, **1**, 1.73, 2.23)
D4M04W05	Gaussian	(0.55, 0.7, 1, 1.73, **2.23**, 2.82, 3.16)
D4M04W08	Gaussian	(0.55, 0.7, 1, 1.73, **2.23**, 2.82, 3.16)
D10M01W05	Gaussian	(0.55, **0.7**, 1, 1.73, 2.23, 2.82, 3.16)
D10M01W08	Gaussian	(0.55, **0.7**, 1, 1.73, 2.23, 2.82, 3.16)
D10M02W05	Gaussian	(0.55, **0.77**, 1, 1.73, 2.23, 2.82, 3.16)
D10M02W08	Gaussian	(0.55, **0.77**, 1, 1.73, 2.23, 2.82, 3.16)
D2M01W05	Uniform	(0.46, **0.57**, 0.87, 1.15, 1.73, 2)
D2M01W08	Uniform	(0.46, **0.57**, 0.87, 1.15, 1.73, 2)
D2M04W05	Uniform	(0.46, 0.57, 0.87, **1.15**, 1.73, 2)
D4M01W05	Uniform	(0.4,**0.46, 0.57**, 0.87, 1.15)
D4M01W08	Uniform	(0.4,**0.46, 0.57**, 0.87, 1.15)
D10M01W05	Uniform	(**0.57**, 0.7, 1, 1.73)
D10M01W08	Uniform	(**0.57**, 0.7, 1, 1.73)
D10M02W05	Uniform	(0.57, 0.7, **0.87**, 1, 1.73)
D10M02W08	Uniform	(0.57, 0.7, **0.87**, 1, 1.73)
D2M01W05	t	(0.8, **0.9, 1.12, 1.3**,1.5)
D2M01W08	t	0.8, **0.9, 1.12, 1.3**,1.5)
D2M04W05	t	(1.12, 1.3, 1.5, 1.87, **1.94**)
D2M04W08	t	(1.12, 1.3, 1.37, 1.5, 1.87, **1.94**)
D4M01W05	t	(1, 1.05, 1.12, 1.3, **1,76, 1.87**, 1.94)
D4M01W08	t	(1, 1.05, 1.12, 1.3, **1.76, 1.87**, 1.94)
D10M01W05	t	(0.55, 0.7, **1.12**, 1.3, 1.87, 1.94)
D10M01W08	t	(0.55, 0.7, 1, **1.12**, 1.3, 1.87, 1.94)
D10M02W05	t	(0.55, 0.7, 1, **1.12**, 1.3, 1.87, 1.94)
D10M02W08	t	(0.55, 0.7, 1, **1.12**, 1.3, 1.87, 1.94)

5.2.1. Tuning the MCMC Algorithm Parameters

The discussion on tuning the scale parameter of the proposal distribution has been around for a long time. The importance of tuning the scale parameters is discussed in the paper that proposed the Metropolis algorithm [8]. In this paper, for Uniform proposal distribution, $Z_n \sim Uniform[-a, a]$, it is noted that "the maximum displacement α must be chosen with some care; if too large, most moves will be forbidden, and if too small, the configuration will not change enough. In either case, it will then take longer to come to equilibrium." [8] This statement explains the problem of choosing the scale parameter very well. A properly tuned scale parameter is needed to obtain a chain with good convergence properties. Unfortunately, this is not a straightforward task with a set of defined rules and procedures. In this section, we propose a method to tune the scale parameter of the proposal distribution. In our discussion, we mainly focus on adjusting the scale parameter of the Gaussian proposals, but the same principle applies to each type of proposal distribution.

There are studies that propose a way to calculate the optimal value for the standard deviation parameter of the Gaussian proposal distribution [51]. However, like for the discussion on the optimal acceptance rate, there is no one standard approach to determine the scale parameter of the proposal distribution [53]. In this study, we propose a method to select the optimal set of parameters of proposal distribution using convergence properties of the chain. We combine Gelman-Rubin diagnostics, acceptance ratio, and a graphical evaluation of the chain properties to evaluate each candidate proposal distribution parameter. Designing a fully automated process without individually evaluating the chain properties is not recommended [49]. For this reason, we propose a semi automated approach where the number of candidate values is decreased by evaluation of their Gelman-Rubin diagnostics value and acceptance ratio as step 1. In step 2, chain properties are evaluated by graphical methods individually, and the final set of chains and, therefore, proposal distribution parameters are decided. In order to automate step 1, we use Grid Search [54] where all proposal distribution parameters are used one

by one in order to choose the ones with good convergence properties. To get an intuition about the sensible candidate values for scale parameters to be used in Grid Search, we make initial experiments by hand to get to know the problem better and make informed decisions. With the help of these experiments, we can define the upper and lower bounds for the candidate values. After finding sensible values for the edges, we can add other candidate values between them and create the set of candidate parameters. Furthermore, if one of the border values performs best among others, we should extend the number of candidate values by going to direction where the best value is obtained and candidate values that are created in such concern are listed in Table 5.2. For example, in order to select the optimum proposal distribution parameters for the mixture of $(0, 0, 0, 0) - (1, 1, 1, 1)$ with equal mixture weights the following steps are taken. For the Gaussian proposal distribution, candidate values for the standard deviation parameter of the proposal distribution are decided as $\sigma = (0.55, 0.7, 1, 1.73, 2.23, 2.82, 3.16)$. Each candidate value is used, and the chain is run using different starting points. For each candidate value, the Gelman-Rubin diagnostics value and acceptance ratio of the chain is calculated. Chains with Gelman-Rubin diagnostics value close to 1 and with acceptance ratio between [0.13, 0.75] are selected in the first step. In the second step, chains with good mixing properties are selected and the ones with poor mixing property are discarded by analyzing their trace plots. For the mixture of $(0, 0, 0, 0) - (1, 1, 1, 1)$, after applying the proposed method, it is found out that the chains with proposal distribution scale parameters 0.7 and 1 achieve good convergence properties.

In Table 5.2, scale parameter values written in bold font are the proposal distribution scale parameters that give a chain with good convergence properties, which are found using the proposed method to find the optimal set of scale parameter values. In some of the problems, multiple scale parameters worked well while in others only one optimal value is found. In order to provide more insight about the selection process of these values, acceptance ratio, Gelman-Rubin diagnostics, and confidence interval of the estimated mean values are reported in Table 5.3. The first column of Table 5.3 is *Experiment ID* which has *Problem ID*, the ab-

breviation of the type of the proposal distribution, and the scale parameter used in the experiment. For example, experiment *D2M01W05GP0.7* means Gaussian proposal distribution with 0.7 as scale parameter is used for problem *D2M01W05*. In this experiment, the acceptance ratio of the chain is 74%, and Gelman-Rubin diagnostics is equal to 1. On the other hand, experiments show that when the distance between the modes becomes higher, we need bigger steps to obtain a chain with good convergence properties. An example can be the problem *D2M04W05*. For this problem, with the change in the distance between the mean vectors of the mixture components, we observed that using 0.7 as a scale parameter does not perform well. It is found out that comparatively higher scale parameters work better. Our results show that using *3.16* as the standard deviation for the Gaussian proposal distribution gives us a chain that has an acceptance ratio of $\sim 20\%$. The experiment *D2M04W05GP3.16* has a Gelman-Rubin diagnostic value equal to 1.01, which is another indication of convergence. Furthermore, as can be observed from Figure 5.1, the chain has good mixing property and is able to find both of the modes. It does not get stuck close to one mode. Another example is the experiment *D2M025W0504GP2.23*. Different from other experiments, the number of mixture components in this problem is 3. Gaussian proposal distribution with $\sigma = 2.23$ gives a chain with Gelman-Rubin diagnostics 1 and acceptance ratio 35%. 95% Confidence Interval of its expectation is [1.28, 1.316], and its plots are given in Figure 5.2. In the histogram plot of one of the marginals of the random vector, we observe that there are three humps. One of them is centered around 0, the other is close to 2, and the other is between 4 and 6. Its acceptance ratio and Gelman-Rubin diagnostics value indicate convergence as well. The change in acceptance ratio as the distance between the modes increase can be observed in Table 5.3. As the distance between the mean vectors of the mixture components increase, we need larger steps in order to achieve good mixing and jump between the modes. Such larger steps are more prone to be rejected therefore we start to see lower acceptance ratios. Increased dimension causes the chain to have lower acceptance ratio as well because in higher dimensions it is harder for the chain to create proposals that step towards to right direction for each of the parameters.

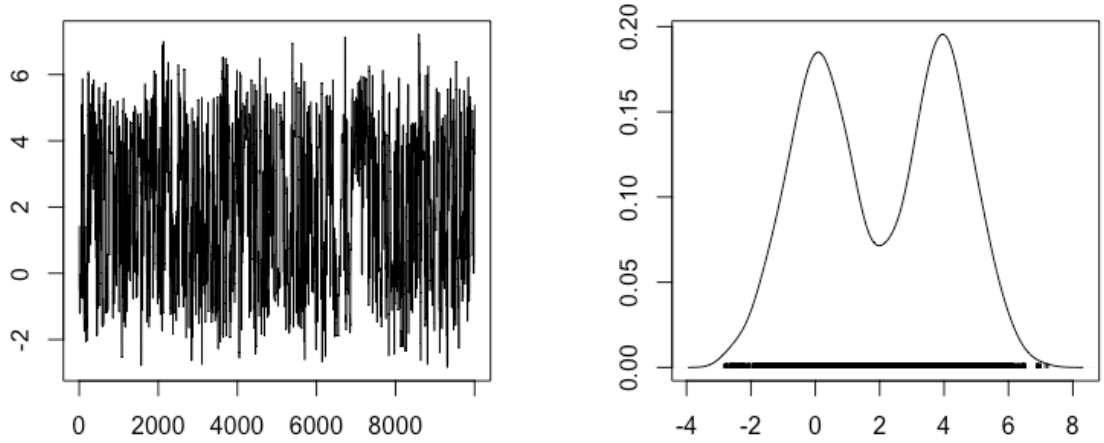

Figure 5.1. Trace plot and histogram of one of the marginals of the random vectors of experiment D2M04W05GP3.16.

Note that for each problem in Table 5.2, results from using only one of the selected scale parameters are reported in Table 5.3. In general, Gelman-Rubin diagnostics of each experiment are very close 1. The highest acceptance ratio values are obtained when the mean vectors of the mixture components are within one standard deviation from each other. In order to evaluate the quality of the estimated mean values of the marginals of the random vectors, standard error values of the estimations are calculated as it is discussed in section 2.4.5 and their 95% confidence interval is shown in Table 5.3. The width of the confidence interval is an important tool to evaluate the quality of the estimation. It also gives an idea about the difficulty of the problems. It is observed that as the dimension of the problem and distance between the modes increase, confidence intervals become wider. For the same problems, using different types of proposal distributions gives estimates with very close standard errors in most of the experiments. However, when it comes to accuracy of the point estimate, there are differences between proposal distributions. As the distance between the mean vectors of the mixture components increase, chains with Uniform proposal distributions fail to provide good convergence properties. Besides, the accuracy of the point estimates obtained by the chains with Uniform proposals is not as good as the estimates from the chains with other types of proposal distributions. Before comparing t proposal distribution with other types of proposal distributions, let us compare t proposal distributions with different degrees of freedom values within themselves. Let us

denote t proposal distribution with degrees of freedom 5, 10 and 25 as t_5, t_{10}, t_{25}. Chains with t_5 proposals have worse convergence diagnostics properties than t_{10} and t_{25}. Besides, chains with t_{25} proposals have very similar characteristics with chains with Gaussian proposals. For this reason, we decided to use chains with t_{10} proposals and tried to obtain chains with good mixing properties by tuning its scale parameter. Chains with t_{10} proposals tend to have confidence intervals with similar widths as chains with Uniform proposal distribution. However, unlike Uniform proposals, it is possible to obtain a good mixing chain using t proposals as the distance between the mean vectors of the mixture components becomes higher.

In 2 dimensional problems, Uniform and t proposals provide slightly narrower confidence intervals for the problems *D2M01W05* and *D2M01W08*. However, for *D2M04W05* and *D2M04W08*, Gaussian proposals provide narrower confidence intervals. For problem *D2M04W05*, the narrowest confidence interval is obtained when Gaussian proposal distribution is used, which is narrower than the estimation using Uniform and t proposal distributions by a factor of 2.64 and 1.52, respectively. In 4 dimensional problems, chains with Uniform proposals do not have good convergence properties for problems *D4M04W05* and *D4M04W08*. Gaussian and t proposals provide very similar confidence intervals; however, the accuracy of point estimates is better for Gaussian proposals. For 10 dimensional problems, there are slight differences between chains with different proposal distributions. Among all experiments, the highest standard error is 1e-1 which is obtained from the experiment *D4M04W05GP2.23*, and it leads to a confidence interval of width 0.4. It is followed by the standard error of the experiment *D4M04W05TP1.94* which is 9e-2. For the problem *D10M02W05*, estimations from each type of proposal distribution have standard error values around 4.5e-2, and this value is 3.7e-2 for the problem *D10M02W08*. The width of the confidence intervals is shortest when problems are two or four dimensional and the mean vectors of the mixture components are $(0,0)-(1,1)$ or $(0,0,0,0)-(1,1,1,1)$. As it is stated in the beginning of section 5.2 these convergence diagnostics tools cannot prove convergence. Their lack of power in diagnosing convergence might sometimes lead us to wrong conclusions.

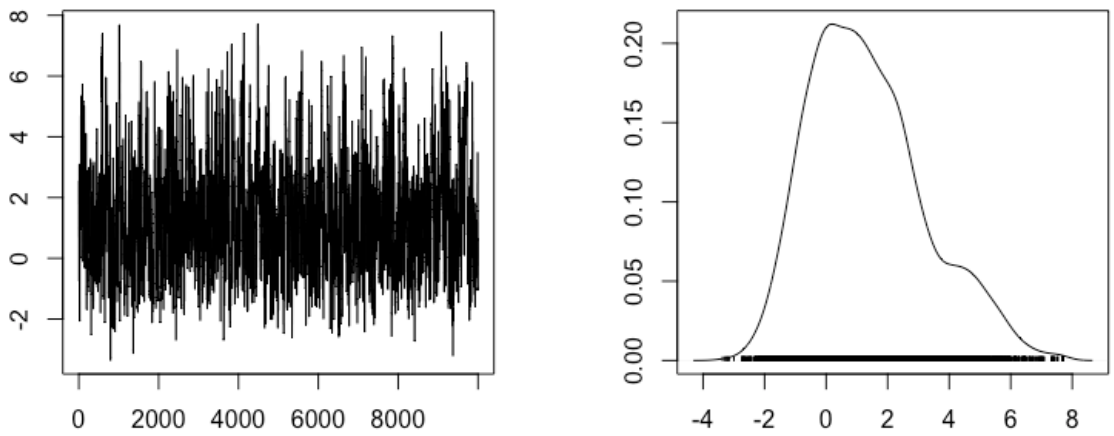

Figure 5.2. Trace plot of MCMC output that samples from mixture of three bivariate normal distributions.

In our study, since we know the answer to each problem, we can determine the cases where convergence diagnostics tools lead to wrong decisions. An example can be for the problem *D4M04W05*. When we use $Uniform \sim [-1.5, 1.5]$ as proposal distribution for this problem, we end up with a chain with good mixing property as observed from its trace plot in Figure 5.3. It does not stay too long in a similar region in successive iterations. This chain has a 40% acceptance rate, and its Gelman-Rubin diagnostics is equal to 1.03. Each convergence diagnostics tool indicates convergence. However, we know there should be two modes, and the chain has failed to find the second mode. From this perspective, the chain actually has a poor mixing property because it fails to find the other mode and cannot jump between the modes.

Another example can be for the problem *D10M02W05*. When we use t distribution with $\nu = 10$ and $\sigma_t = 0.6$, we obtained a chain with good convergence properties. It has an acceptance ratio of 20%, and the Gelman-Rubin diagnostics value is equal to 1.04. The chain has good mixing property, as can be observed from its trace plot in Figure 5.4. However, when we use our knowledge on the true solution of the problem we can identify that the chain actually has poor mixing property. It gets stuck at a single mode and it is unable to find the second mode.

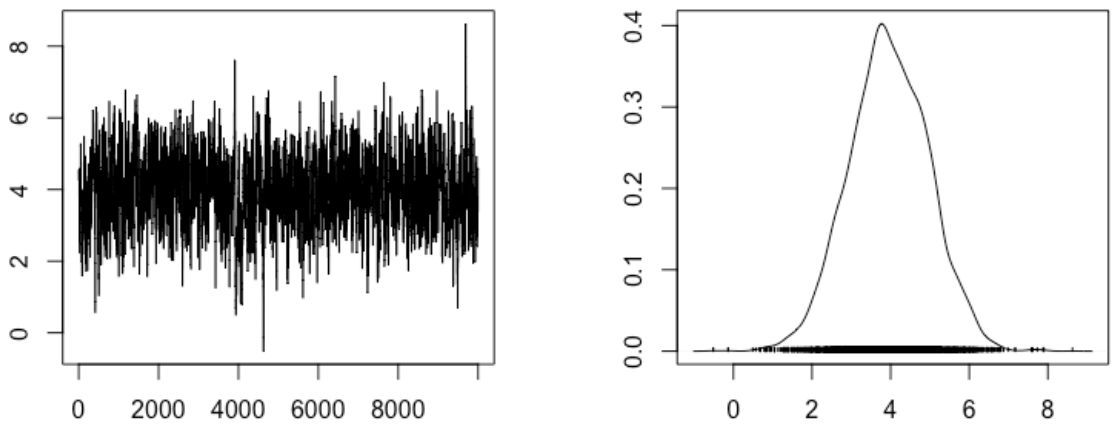

Figure 5.3. Chain with poor mixing properties for problem D4M04W05.

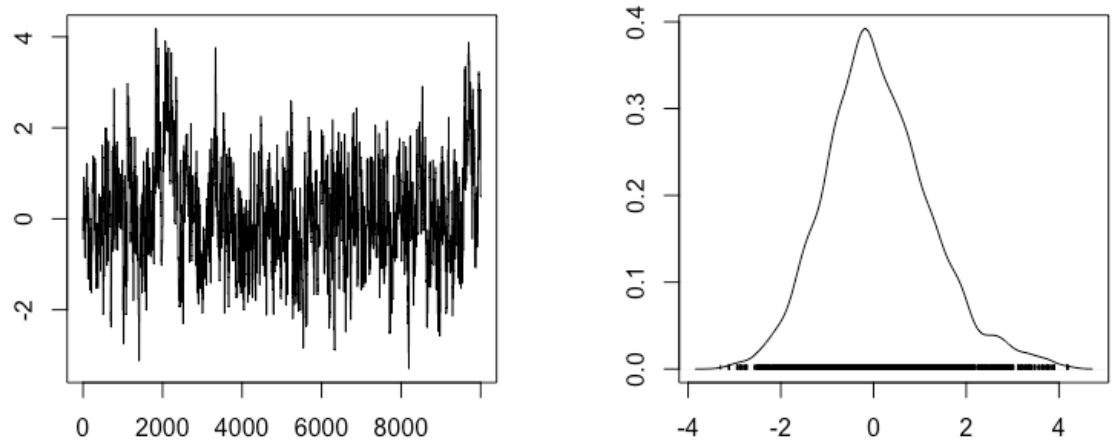

Figure 5.4. Chain with poor mixing properties for problem D10M02W05.

In this section, it is demonstrated that several techniques can be combined in order to find a chain with good convergence properties. Tuning the proposal distribution parameters and determining the convergence of the chain is a task where there is no standard and acknowledged approach. The above experiments show our approach to tune the proposal distribution parameters and diagnose the convergence of the chain. It is shown that multiple approaches, which are Gelman-Rubin diagnostics, evaluation of acceptance ratio, visualization of the chain results, and confidence interval of the expectation, can be used in order to evaluate the properties of the MCMC output and therefore conduct convergence diagnostics in order to choose the optimum proposal distribution parameters. We also discussed the scenarios where convergence diagnostics tools might lead to wrong conclusions about the convergence properties of the chain.

Having selected the chains with good convergence properties and demonstrated the tuning process of the proposal distribution parameters, the next step is to apply variance reduction algorithms in order to increase the efficiency of these simulations. In the next chapter, the efficiency gains from implementing variance reduction techniques are discussed.

Table 5.3. Details about chains with good convergence properties.

Experiment ID	**Acceptance Ratio**	**GRD**	**95% Confidence Interval**
D2M01W05GP0.7	0.74	1	[0.485, 0.515]
D2M01W08GP0.7	0.75	1	[0.187, 0.214]
D2M04W05GP3.16	0.198	1.01	[1.975, 2.025]
D2M04W08GP3.16	0.18	1.01	[0.781, 0.825]
D2M023W0504GP1	0.62	1	[1.093, 1.114]
D2M025W0504GP2.23	0.35	1	[1.28, 1.316]
D4M01W05GP0.7	0.4	1	[0.495, 0.515]
D4M01W08GP0.7	0.39	1	[0.192, 0.212]
D4M02W05GP0.7	0.185	1	[0.982, 1.038]
D4M02W08GP1.73	0.178	1	[0.383, 0.419]
D4M04W05GP2.23	0.15	1.03	[1.8, 2.2]
D4M04W08GP2.23	0.14	1.02	[0.795, 0.815]
D10M01W05GP0.7	0.298	1.02	[0.481, 0.521]
D10M01W08GP0.7	0.298	1.02	[0.189, 0.219]
D10M02W05GP0.77	0.298	1.02	[0.918, 1.098]
D10M02W08GP0.77	0.298	1.02	[0.322, 0.47]
D2M01W05UP0.57	0.5	1	[0.488, 0.512]
D2M01W08UP0.57	0.72	1	[0.189, 0.211]
D2M04W05UP1.73	0.316	1.03	[1.927, 2.067]
D4M01W05UP0.46	0.68	1	[0.482, 0.513]
D4M01W08UP0.46	0.67	1	[0.182, 0.213]
D10M01W05UP0.57	0.38	1.02	[0.48, 0.52]
D10M01W08UP0.57	0.38	1.02	[0.18, 0.214]
D10M02W05UP0.87	0.18	1.03	[0.956, 1.144]
D10M02W08UP0.87	0.18	1.01	[0.32, 0.47]
D2M01W05TP0.9	0.56	1	[0.49, 0.508]
D2M01W08TP0.9	0.62	1	[0.192, 0.208]
D2M04W05TP1.94	0.34	1	[1.97, 2.05]
D2M04W08TP1.94	0.34	1	[0.769, 0.825]
D4M01W05TP1.76	0.18	1	[0.49, 0.51]
D4M01W08TP1.76	0.183	1.02	[0.189, 0.209]
D4M04W05TP1.94	0.15	1.02	[1.84, 2.2]
D4M04W08TP1.94	0.15	1	[0.663, 0.897]
D10M01W05TP1.05	0.14	1.02	[0.482, 0.522]
D10M01W08TP1.1	0.133	1	[0.18, 0.214]
D10M02W05TP0.93	0.133	1	[0.898, 1.082]
D10M02W08TP0.93	0.133	1	[0.334, 0.466]

6. RESULTS AND EVALUATION OF METHODS

The ultimate aim of our study is to increase the efficiency of MCMC algorithms using variance reduction techniques. In Chapter 5, the process of evaluating the convergence properties of chains is demonstrated and chains with good convergence properties are selected. This was a crucial step because the prerequisite of applying variance reduction algorithms is to have chains with good convergence properties. In this chapter, variance reduction algorithms will be applied to the selected chains. Variance reduction algorithms used in this chapter are general variance reduction algorithms which are Antithetic Variates (AV), Multiple Control Variates (MCV), and Latin Hypercube Sampling (LHS). The Averaging Method explained in section 4.1.3 is also implemented. Implementation details of the Antithetic Variates algorithm are given in Section 4.2.1.1. In Multiple Control Variates algorithm, Inner Control Variates proposed in Section 4.2.1.3 and Control Variates proposed in [41] are used as it is shown in Section 4.2.1.4. Latin Hypercube Sampling method is described in Section 3.4.1 and its usage in our setting is described in Section 4.2.1.2. Furthermore, a method to combine general variance reduction algorithms is proposed in section 4.2.1.5. We implemented the proposed method and discussed the impact of using multiple variance reduction techniques on the variance of the Random-Walk Metropolis-Hastings algorithm, Root Mean Squared Error and Mean Absolute error of the estimations, and standard error of the resulting estimations, which are mean, variance, and correlation values.

In Table 5.3, the resulting confidence intervals of the estimated mean parameters are shown. The width of the confidence intervals vary according to the input parameter. With the help of variance reduction algorithms, we would like to obtain an estimation with a similar point estimate as a naive simulation but with narrower confidence intervals. For standard error estimations of the estimated mean value of the mixture distribution, we use non-overlapping batch means technique and for standard error estimations of estimated variance and estimated correlation values, we use the Block Bootstrapping method. Both methods are discussed in

section 2.4.5.

In order to evaluate the performance of variance reduction algorithms, we will use the variance reduction factor (VRF) as a metric, which is explained in Chapter 3. To reiterate, this value is calculated by dividing the standard error of the naive simulation estimation and the standard error of the variance reduction applied simulation estimation.

6.1. Results from Random-Walk Metropolis-Hastings Experiments

Let us start with the discussion of the effectiveness of variance reduction algorithms. Results of several experiments show that we can decrease the standard error of the simulation and obtain estimations with narrower confidence intervals using general variance reduction algorithms. However, the performance is highly affected by the difficulty of the problem, which we introduce by creating integration examples in different dimensions and different mean values of mixture components. For each variance reduction method, we made 14 experiments for dimension 2, 13 experiments for dimension 4, and 12 experiments for dimension 10. Table 6.1 presents the relative frequencies of successful variance reduction techniques which are defined by observed VRF values greater than 1.1, 1.5, or 3 for the estimated mean parameter. We observe that the AV algorithm is most successful in all dimensions. The VRF values for AV are always greater than 1.5 and often greater than 3. Besides, combining AV with MCV provides higher VRF values, and adding LHS to this combination rarely increases the VRF values. MCV and LHS algorithms fail to provide VRF values greater than 1.5 in each experiment. They only provide VRF values greater than 1.1 in 2 and 4 dimensional experiments. Although using MCV and LHS together provides higher VRF values than individual usage of these algorithms, resulting VRF values are still below 1.5 in each experiment. Besides, the lowest VRF values are obtained by Averaging Method, which fails to provide VRF values greater than 1.1 in each experiment. MCV algorithm uses different types of Control Variates. In Chapter 4, we demonstrated the two different groups of Control Variates used in the MCV algorithm,

which are Inner Control Variates and Control Variates proposed in [41]. When we break down the performance of these Control Variates, we observed that these Control Variates have similar contributions to resulting VRF values, and using them together in the MCV algorithm gives higher VRF values. Nevertheless, the resulting VRF values are still lower than 1.5 in each experiment.

When it comes to the performance of variance reduction algorithms for estimated variance and estimated correlation parameters, we observe comparatively smaller VRF values. For estimated mean parameter, VRF values of AV algorithm are often greater than 3 but for estimated variance and estimated correlation parameters, AV algorithm fails to provide VRF values greater than 3. VRF values of AV algorithm for estimated variance and estimated correlation parameters are very similar to each other and their values range between [1, 2.9] where 9% of VRF values are greater than 2, 64% of VRF values are greater than 1.5, and 90% of VRF values are greater than 1.1.

In Chapter 4, we discussed the possible impact of having chains with low acceptance ratios on the performance of the AV algorithm. In Chapter 5, we showed that as the dimension of the problem and distance between the mean vectors of the mixture components increase, the acceptance ratio of the chain decrease. In this chapter, VRF results of the AV algorithm show that the VRF values obtained by the AV algorithm are highly affected by the acceptance ratio of the chain. Higher acceptance ratios enable us to retain the negative correlation between the chains and it provides higher VRF values. On the other hand, as the difficulty of the problem increases, we obtain chains with lower acceptance ratios and this leads to lower VRF values. Therefore, since VRF values are highly affected by the input parameters of the experiments, rather than the VRF values themselves, the main interest is whether variance reduction algorithms can provide variance reduction consistently in each experimental setup. Our key result is that *for all types of parameters selected in our experiments, we were able to obtain small to moderate VRF values.* The *small to moderate* VRF values are mostly achieved via the AV algorithm and its collective usage with the MCV algorithm.

Although the AV algorithm and its combination with the MCV algorithm consistently provide small to moderate variance reduction across all experiments, to rely on the VRF results, we need to validate if the standard error estimations are biased or not. This is because MCMC algorithms provide correlated samples and we use methods like *Batch Means* and *Block Bootstrap* techniques to calculate standard errors of the estimations, which may give rise to biased estimations.

To evaluate the accuracy of the standard error estimations of experiments in each dimension, we calculated the coverage probabilities of the confidence intervals. We repeated the experiments 100 times and created 100 different 95% confidence intervals. By this means, we can calculate the coverage probabilities of the confidence intervals by finding how many times they include the exact solution of the problem for each parameter. We also calculated Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) for the estimated parameters. We repeated each of the experiments using chains of length 1e4, 1e5, 5e5, and 1e6. The reason for running chains with different lengths is to evaluate the impact of chain length on RMSE and MAE values of the estimations, coverage probabilities and potentially obtain improved coverage probabilities and decreased RMSE/MAE values as the length of the chain increase. Not only for naive simulation, but it is also crucial to calculate coverage probabilities and RMSE/MAE values for variance reduction applied simulations to check if variance reduction algorithms introduce bias to standard error estimates and the accuracy of the estimate itself. For this reason, we implemented an AV algorithm to these experiments and calculated the RMSE/MAE values and coverage probabilities of variance reduction applied simulations. For 2 dimensional experiments, the resulting coverage probabilities in each sample size showed that the standard error estimations of the parameters calculated using naive simulation output are unbiased. Furthermore, the coverage probabilities of each type of estimated parameter are very close or higher than 90% and lower than 96%, which indicates that the confidence intervals are not too wide or too narrow, even for chains with a sample size equal to 1e4. Coverage probabilities of the confidence intervals are not decreased after using variance reduction algorithms for estimated mean, estimated correlation, and estimated

variance parameters, and RMSE/MAE values of these parameters improve as the sample size increases. Therefore, we can obtain a chain with a similar coverage probability as the naive simulation but with a narrower confidence interval and lower RMSE/MAE values by using variance reduction algorithms in 2 dimensional experiments.

When it comes to 4 and 10 dimensional experiments, we encountered coverage probabilities that are not as high as the significance level of the confidence interval. Results of the coverage probabilities of each parameter are presented in Table 6.2. Coverage probabilities of the experiment *D4M02W05GP0.7* are very close to or higher than 90% for each estimated parameter in each sample size. Applying variance reduction algorithms does not decrease the coverage probabilities. When we evaluate RMSE and MAE values of the estimated parameters of this experiment, as they are presented in Table 6.3 and Table 6.4, we observe that the RMSE/MAE values decrease for both naive and variance reduction applied simulation estimations as the sample size increase. The AV algorithm decreases the RMSE/MAE values of each estimated parameter in each sample size level. For the experiments *D4M04W08GP2.23* and *D10M02W05GP0.77*, whose confidence intervals are the widest among all experiments and have higher RMSE/MAE values, the length of the chain has a significant impact on the coverage probabilities and RMSE/MAE values. As it is shown in the below tables, when the sample size is equal to 1e4, RMSE/MAE values are high, coverage probabilities are very low. Furthermore, coverage probabilities of variance reduction applied simulation are lower than the naive simulation, especially in the experiment *D10M02W05GP0.77*. The reason for having low coverage probabilities and high RMSE/MAE values is that the chain with size 1e4 has poor convergence properties. This problem requires longer chains to explore the target distribution. In the experiment *D10M02W05GP0.77*, RMSE and MAE values of estimations decreased by the factor of 10 when the sample size increased from 1e4 to 1e5. Such a decrease is higher than expected because if it were an i.i.d. sample, we would expect a decrease by a factor of $\sqrt{10}$. Increasing the sample size to 1e5 from 1e4 provides a substantially higher factor of decrease in RMSE/MAE values than the increases from 1.e5 to 1e6. Thus, we

understand that the RMSE and MAE values of the estimated parameters are affected by the convergence properties of the chain. Such a result and the difference between the coverage probabilities of naive and variance reduction applied simulation show that when variance reduction algorithms are applied to chains with poor convergence properties, it may give misleading results. On the other hand, as the sample size increase, coverage probabilities of naive and variance reduction applied simulations improve and become very close to each other. In each of the 2, 4, and 10 dimensional problems, our experiments demonstrate that as long as variance reduction algorithms are applied to chains with good convergence properties, a variance reduction applied simulation has a similar coverage probability as the naive simulation with lower RMSE/MAE values.

Summarizing our results on coverage probabilities of the confidence intervals and RMSE/MAE values, we can say that by using variance reduction for MCMC, we get narrower confidence intervals and estimations with higher accuracy for our examples. Thus we can conclude that the variance reductions can be helpful in practice also for MCMC algorithms.

At this point we want to emphasize again that the variance reduction algorithms can be successfully applied only to chains that have good convergence properties. We discussed the coverage probabilities of a chain of length 1e4 and its poor convergence properties above. To highlight the importance of that warning we add here an example of a chain with length 1e6 and select mixture components with mean vectors (0, 0, 0, 0) - (4, 4, 4, 4) and 0.5 as mixture weights. As a proposal distribution, Gaussian proposal distribution with scale parameter 0.1 is selected. The resulting chain is sensitive to its starting value and it fails to explore the high probability regions because its proposal distribution fails to propose large enough steps to jump between the modes. If it is started from a region where it is close to one of the modes, it fails to move away from that region and it results in a chain with an acceptance ratio higher than 90%. Since the accepted samples are from a limited region, standard error estimations become low which leads to obtaining confidence intervals that do not include the exact results of the

parameters at all, which means that the coverage probabilities of the confidence intervals are equal to 0%. Using variance reduction algorithms for such a chain should be avoided because variance reduction algorithms cannot remove the bias of the estimations.

This chapter presented our results of implementing variance reduction algorithms for MCMC simulations using the experiments defined in Chapter 5. We showed that variance reduction algorithms are applicable to MCMC problems, and that for problems of small to moderate dimensions higher precision can be obtained by applying variance reduction algorithms. Furthermore, we demonstrated that variance reduction algorithms can decrease the standard error and RMSE/MAE values of the estimations and still achieve the same coverage probabilities as the naive simulation. Before concluding this chapter, we emphasize that our observations are related to the problem type we have selected. Therefore, we do not claim these are general results for implementing variance reduction techniques for MCMC algorithms.

Table 6.1. Relative frequency of VRF values greater than threshold values of 1.1, 1.5 and 3 across 2, 4 and 10 dimensional experiments for each type of variance reduction algorithms and their combinations.

VR Method	**Dimension**	**VRF>1.1**	**VRF>1.5**	**VRF>3**
AV	2	100%	100%	43%
MCV	2	57%	0%	0%
LHS	2	35%	0%	0%
Averaging	2	0%	0%	0%
AV+MCV	2	100%	100%	71%
AV+LHS	2	100%	100%	64%
MCV+LHS	2	79%	0%	0%
AV+MCV+LHS	2	100%	100%	71%
AV	4	100%	100%	46%
MCV	4	31%	0%	0%
LHS	4	15%	0%	0%
Averaging	4	0%	0%	0%
AV+MCV	4	100%	100%	61%
AV+LHS	4	100%	100%	46%
MCV+LHS	4	38%	0%	0%
AV+MCV+LHS	4	100%	100%	61%
AV	10	100%	100%	41%
MCV	10	0%	0%	0%
LHS	10	0%	0%	0%
Averaging	10	0%	0%	0%
AV+MCV	10	100%	100%	41%
AV+LHS	10	100%	100%	33%
MCV+LHS	10	0%	0%	0%
AV+MCV+LHS	10	100%	100%	41%

Table 6.2. Coverage probabilities of the experiments. Values in squared brackets indicates the coverage probabilities after applying variance reduction.

Experiment ID	**N**	**Mean**	**Variance**	**Correlation**
D4M04W08GP2.23	1e4	10% [10%]	10% [9%]	9% [9%]
D4M04W08GP2.23	1e5	63% [63%]	64% [65%]	38% [38%]
D4M04W08GP2.23	5e5	94% [94%]	65% [65%]	57% [58%]
D4M04W08GP2.23	1e6	96% [95%]	65% [65%]	60% [59%]
D4M02W05GP0.7	1e4	91% [95%]	93% [96%]	89% [93%]
D4M02W05GP0.7	1e5	92% [96%]	94% [93%]	97% [95%]
D4M02W05GP0.7	5e5	94% [97%]	95% [94%]	94% [95%]
D4M02W05GP0.7	1e6	95% [97%]	95% [96%]	97% [94%]
D10M02W05GP0.77	1e4	27% [15%]	40% [24%]	41% [28%]
D10M02W05GP0.77	1e5	77% [85%]	83% [80%]	87% [83%]
D10M02W05GP0.77	5e5	83% [89%]	89% [90%]	95% [95%]
D10M02W05GP0.77	1e6	87% [90%]	90% [91%]	91% [91%]

Table 6.3. RMSE and MAE values for estimated mean parameter from repeated experiments. Values in squared brackets indicates the error values after applying variance reduction.

Experiment ID	**N**	**Mean RMSE**	**Mean MAE**
D4M04W08GP2.23	1e4	1.24 [0.91]	1.1 [0.83]
D4M04W08GP2.23	1e5	0.15 [0.11]	0.125 [0.09]
D4M04W08GP2.23	5e5	0.07 [0.05]	0.08 [0.04]
D4M04W08GP2.23	1e6	0.05 [0.03]	0.05 [0.03]
D4M02W05GP0.7	1e4	0.14 [0.05]	0.1 [0.05]
D4M02W05GP0.7	1e5	0.045 [0.02]	0.03 [0.01]
D4M02W05GP0.7	5e5	0.02 [0.008]	0.02 [0.007]
D4M02W05GP0.7	1e6	0.01 [0.006]	0.01 [0.005]
D10M02W05GP0.77	1e4	0.8 [0.72]	0.8 [0.7]
D10M02W05GP0.77	1e5	0.17 [0.09]	0.12 [0.07]
D10M02W05GP0.77	5e5	0.06 [0.04]	0.05 [0.03]
D10M02W05GP0.77	1e6	0.04 [0.03]	0.03 [0.02]

Table 6.4. RMSE and MAE values for estimated variance and estimated correlation parameters from repeated experiments. Values in squared brackets indicates the error values after applying variance reduction.

Experiment ID	**N**	**Variance RMSE**	**Variance MAE**	**Correlation RMSE**	**Correlation MAE**
D4M04W08GP2.23	1e4	0.64 [0.57]	0.6 [0.52]	0.86 [0.8]	0.77 [0.74]
D4M04W08GP2.23	1e5	0.08 [0.06]	0.07 [0.05]	0.03 [0.026]	0.025 [0.025]
D4M04W08GP2.23	5e5	0.03 [0.02]	0.032 [0.025]	0.016 [0.013]	0.013 [0.01]
D4M04W08GP2.23	1e6	0.02 [0.01]	0.02 [0.017]	0.012 [0.01]	0.01 [0.008]
D4M02W05GP0.7	1e4	0.05 [0.04]	0.04 [0.03]	0.05 [0.04]	0.04 [0.03]
D4M02W05GP0.7	1e5	0.015 [0.012]	0.012 [0.01]	0.014 [0.01]	0.01 [0.009]
D4M02W05GP0.7	5e5	0.006 [0.005]	0.005 [0.004]	0.007 [0.005]	0.005 [0.004]
D4M02W05GP0.7	1e6	0.005 [0.004]	0.004 [0.003]	0.005 [0.004]	0.004 [0.003]
D10M02W05GP0.77	1e4	0.27 [0.25]	0.22 [0.21]	0.48 [0.4]	0.35 [0.33]
D10M02W05GP0.77	1e5	0.03 [0.024]	0.024 [0.02]	0.034 [0.025]	0.026 [0.02]
D10M02W05GP0.77	5e5	0.01 [0.006]	0.007 [0.005]	0.009 [0.006]	0.007 [0.005]
D10M02W05GP0.77	1e6	0.006 [0.006]	0.005 [0.004]	0.006 [0.005]	0.005 [0.004]

7. CONCLUSION

In this thesis, we studied the properties of MCMC algorithms under an experimental setup, in which Bayesian statistics applications inspire the experiments. We created multiple experiments with varying difficulties using a mixture of multivariate normal distributions to explore the performance of MCMC algorithms. The problem's difficulty is calibrated by changing the dimension of the problem, the distance between the mean vectors of the mixture components, and the weight of the mixture components. In Chapter 5, we discussed the details of the input parameters of the problems and the importance of selecting the chain with good convergence properties. We used the Random-Walk Metropolis-Hastings algorithm with Gaussian, Uniform, and t proposal distributions to sample from a mixture of multinormal distributions and estimated the random vector's mean, variance, and correlation values. We chose the optimum set of scale parameters for each type of proposal distribution from a candidate set of values. This selection process is performed by using different convergence diagnostics methods: the Gelman-Rubin diagnostics, evaluation of the acceptance ratio, and visualization of the trace plot of the chain. This process is crucial because the quality of the estimated parameters relies on how good the convergence is, and variance reduction techniques should be applied only to the chains with good convergence properties.

Antithetic Variates, Control Variates, Latin Hypercube Sampling, and Averaging Method techniques are applied to the chains with good convergence properties. To evaluate the performance of the variance reduction algorithms, we calculated the decrease in the standard error of the estimations after applying variance reduction. We calculated the standard error values by using the *Batch Means* and *Block Bootstrap* techniques and calculated the variance reduction factor values. Experiments showed that small to moderate variance reduction can be achieved. VRF results of experiments show that the Antithetic Variates algorithm is most successful variance reduction algorithm in our experimental setting. The variance reduction capabilities of the Multiple Control Variates, Latin Hypercube Sam-

pling, and Averaging Method are limited. However, we showed that a slightly higher variance reduction can be achieved if the Antithetic Variates algorithm is combined with the Multiple Control Variates algorithm. Besides, we evaluated the accuracy of the standard error estimations obtained by these techniques by using repeated samples and creating multiple confidence intervals to calculate the coverage probabilities of the confidence intervals and RMSE/MAE values of the estimated parameters. Experiments showed that as the difficulty of the problem increase, the standard error estimations obtained from the *Batch Means* and *Block Bootstrap* techniques become biased and lower than they should be, and RMSE/MAE values become higher. In order to study the relation between the quality of the standard error estimate and the length of the chain, we repeated the experiments using chains of different lengths. We showed that as the size of the chain increase, the coverage probabilities of the confidence intervals improve, and RMSE/MAE values decrease. We observed that variance reduction algorithms can decrease the RMSE/MAE values and the standard errors of the estimations without decreasing the coverage probabilities. This result is very important because having an estimate with lower standard error and RMSE/MAE values than naive simulation with similar coverage probabilities shows the effectiveness of the variance reduction algorithms. Our findings should be applicable to MCMC problems for random vectors of small to moderate dimensions with densities that are not too different from the mixture of multinormal distributions. Thus, we can conclude that the variance reductions can be helpful in practice also for MCMC algorithms.

REFERENCES

1. McGeoch, C., "Analyzing Algorithms by Simulation: Variance Reduction Techniques and Simulation Speedups", *ACM Computing Surveys*, Vol. 24, 1992.

2. Hörmann, W., *Efficient Monte Carlo for Option Pricing*, IE 58D Lecture Notes, Boğaziçi University, 2017.

3. Packham, N., "Combining Latin Hypercube Sampling With Other Variance Reduction Techniques", *Wilmott*, Vol. 76, 2015.

4. van Ravenzwaaij, D., P. Cassey and S. D. Brown, "A Simple Introduction to Markov Chain Monte Carlo Sampling", *Psychonomic Bulletin & Review*, 2016.

5. Gamerman, D. and H. F. Lopes, *Markov Chain Monte Carlo: Stochastic Simulation for Bayesian Inference, Second Edition*, Chapman and Hall/CRC, Boca Raton, FL, USA, 2006.

6. Robert, C. and G. Casella, *Monte Carlo Statistical Methods, Second Edition*, Springer, New York, NY, USA, 2005.

7. Douc, R., F. Maire and J. Olsson, "On the Use of Markov Chain Monte Carlo Methods for the Sampling of Mixture Models: A Statistical Perspective", *Statistics and Computing*, Vol. 25, 2015.

8. Metropolis, N., A. W. Rosenbluth, M. N. Rosenbluth and A. H. Teller, "Equation of State Calculations by Fast Computing Machines", *The Journal of Chemical Physics*, Vol. 21, 1953.

9. Hastings, W. K., "Monte Carlo Sampling Methods Using Markov Chains and

Their Applications", *Biometrika*, Vol. 57, No. 1, pp. 97–109, 04 1970.

10. Liu, J. S., W. H. Wong and A. Kong, "Covariance Structure of the Gibbs Sampler with Applications to the Comparisons of Estimators and Augmentation Schemes", *Biometrika*, Vol. 81, 1994.

11. Roy, V., "Convergence Diagnostics for Markov Chain Monte Carlo", *Annual Review of Statistics and Its Application*, Vol. 7, 2020.

12. Gelman, A. and D. B. Rubin, "Inference From Iterative Simulation Using Multiple Sequences", *Statistical Science*, Vol. 7, No. 4, pp. 457–472, 1992.

13. Geyer, C. J., "Practical Markov Chain Monte Carlo", *Statistical Science*, Vol. 7, No. 4, pp. 473–483, 1992.

14. Riabiz, M., W. Y. Chen, J. Cockayne, P. Swietach, S. Niederer, L. Mackey and C. Oates, "Optimal Thinning of MCMC Output", *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, 2021.

15. Link, W. A. and M. J. Eaton, "On Thinning of Chains in MCMC", *Methods in Ecology and Evolution*, Vol. 3, No. 1, pp. 112–115, 2012.

16. Raftery, A. E. and S. M. Lewis, "Practical Markov Chain Monte Carlo: Comment: One Long Run with Diagnostics: Implementation Strategies for Markov Chain Monte Carlo", *Statistical Science*, Vol. 7, No. 4, pp. 493–497, 1992.

17. Ömer Yasin Birey, *Comparison Between a Unique Long MCMC Chain and Short Parallel MCMC Chains for Bayesian Inference*, Master Thesis, University Carlos III de Madrid, 2019.

18. Flegal, J. M. and G. L. Jones, *Implementing MCMC: Estimating with Confidence*, chap. 7, CRC Press, 2011.

19. Gong, L. and J. M. Flegal, "A Practical Sequential Stopping Rule for High-

Dimensional Markov Chain Monte Carlo", *Journal of Computational and Graphical Statistics*, Vol. 25, No. 3, pp. 684–700, 2016.

20. Flegal, J. M. and G. L. Jones, "Batch Means and Spectral Variance Estimators in Markov Chain Monte Carlo", *The Annals of Statistics*, Vol. 38, No. 2, pp. 1034–1070, 2010.

21. Kunsch, H. R., "The Jackknife and the Bootstrap for General Stationary Observations", *The Annals of Statistics*, Vol. 17, No. 3, pp. 1217–1241, 1989.

22. Liu, R. Y., *Moving Blocks Jackknife and Bootstrap Capture Weak Dependence*, chap. 3, pp. 225–248, John Wiley, New York, 1992.

23. DiCiccio, T. J. and B. Efron, "Bootstrap Confidence Intervals", *Statistical Science*, Vol. 11, No. 3, p. 189–228, 1996.

24. Efron, R. T., Bradley, *An Introduction to the Bootstrap*, Chapman & Hall, 1994.

25. Lahiri, S. N., *Resampling Methods for Dependent Data*, Springer, 2003.

26. Owen, A. B., *Monte Carlo Theory, Methods and Examples*, 2013.

27. Black, F. and M. Scholes, "The Pricing of Options and Corporate Liabilities", *Journal of Political Economy*, Vol. 81, No. 3, pp. 637–654, 1973.

28. Dingeç, K. D. and W. Hörmann, "A General Control Variate Method for Option Pricing Under Lévy Processes", *European Journal of Operational Research*, Vol. 221, No. 2, pp. 368–377, 2012.

29. L'Ecuyer, P. and E. Buist, "On the Interaction Between Stratification and Control Variates With Illustrations in a Call Centre Simulation", *Journal of Simulation*, Vol. 2, No. 1, pp. 29–40, 2008.

30. Tsai, S. C. and C. H. Kuo, "Screening and Selection Procedures with Control Variates and Correlation Induction Techniques", *Naval Research Logistics*, Vol. 59, No. 5, pp. 340–361, 2012.

31. Blackwell, D., "Conditional Expectation and Unbiased Sequential Estimation", *The Annals of Mathematical Statistics*, Vol. 18, No. 1, pp. 105–110, 1947.

32. Craiu, R. V. and C. Lemieux, "Acceleration of the Multiple-Try Metropolis Algorithm Using Antithetic and Stratified Sampling", *Statistics and Computing*, Vol. 17, 2007.

33. Bierkens, J., "Non-reversible Metropolis-Hastings", *Statistics and Computing*, Vol. 26, 2016.

34. Martino, L., V. Elvira, D. Luengo and F. Louzada, "Parallel Metropolis Chains with Cooperative Adaptation", *2016 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016 - Proceedings*, Vol. 2016-May, pp. 3974–3978, Institute of Electrical and Electronics Engineers Inc., United States, May 2016.

35. Calderhead, B., "A General Construction for Parallelizing Metropolis-Hastings Algorithms", *Proceedings of the National Academy of Sciences*, Vol. 111, No. 49, pp. 17408–17413, 2014.

36. Robert, C. P., V. Elvira, N. Tawn and C. Wu, "Accelerating MCMC Algorithms", *WIREs Computational Statistics*, Vol. 10, No. 5, 2018.

37. Frigessi, A., J. Gåsemyr and H. Rue, "Antithetic Coupling of Two Gibbs Sampler Chains", *The Annals of Statistics*, Vol. 28, No. 4, pp. 1128–1149, 2000.

38. Casella, G. and C. P. Robert, "Rao-Blackwellisation of Sampling Schemes",

Biometrika, Vol. 83, 1996.

39. Robert, C. P., V. Elvira, N. Tawn and C. Wu, "Accelerating MCMC Algorithms", *WIREs Computational Statistics*, Vol. 10, No. 5, 2018.

40. MacEachern, S. N. and M. Peruggia, "Subsampling the Gibbs sampler: Variance Reduction", *Statistics & Probability Letters*, Vol. 47, No. 1, pp. 91–98, 2000.

41. Hammer, H. and H. Tjelmeland, "Control Variates for the Metropolis–Hastings Algorithm", *Scandinavian Journal of Statistics*, Vol. 35, No. 3, pp. 400–414, 2008.

42. Dellaportas, P. and I. Kontoyiannis, "Control Variates for Reversible MCMC Samplers", *Journal of the Royal Statistical Society Series B*, 2010.

43. Handerson, S., *Variance Reduction via an Approximating Markov Process*, Ph.D. Thesis, Stanford University, 1997.

44. Nasroallah, A., "K-Antithetic Variates in Monte Carlo Simulation", *Afrika Statistika*, Vol. 3, No. 1, 2009.

45. Bédard, M., "Hierarchical Models and Tuning of Random Walk Metropolis Algorithms", *Journal of Probability and Statistics*, 2019.

46. Roberts, G. O. and J. S. Rosenthal, "Optimal Scaling for Various Metropolis-Hastings Algorithms", *Statistical Science*, Vol. 16, No. 4, pp. 351–367, 2001.

47. Sinharay, S., "Assessing Convergence of the Markov Chain Monte Carlo algorithms: A Review", *ETS Research Report Series*, Vol. 2003, pp. i–52, 06 2003.

48. Polson, N. G., "Convergence of Markov Chain Monte Carlo Algorithms", *Proceedings of the Fifth International Meeting on Bayesian Statistics*, pp. 483–

512, 1994.

49. Cowles, M. K. and B. P. Carlin, "Markov Chain Monte Carlo Convergence Diagnostics: A Comparative Review", *Journal of the American Statistical Association*, Vol. 91, No. 434, pp. 883–904, 1996.

50. Ford, E. B., *Lecture Notes in Convergence Diagnostics for Markov chain Monte Carlo*, 2015.

51. Roberts, G. O., A. Gelman and W. R. Gilks, "Weak Convergence and Optimal Scaling of Random Walk Metropolis Algorithms", *The Annals of Applied Probability*, Vol. 7, No. 1, pp. 110–120, 1997.

52. Bédard, M., "Optimal Acceptance Rates for Metropolis Algorithms: Moving Beyond 0.234", *Stochastic Processes and Their Applications*, Vol. 118, No. 12, pp. 2198–2222, 2008.

53. Bédard, M. and J. Rosenthal, *Optimal Scaling of Metropolis Algorithms: Is 0.234 As Robust As Believed?*, 2007.

54. Ensor, K. B. and P. W. Glynn, *Stochastic Optimization via Grid Search*, 1997.

APPENDIX A: TABLES

Table A.1. VRF values for estimated mean parameter for mixture of bivariate normal distributions.

Experiment ID	**VR Techniques**	**VRF1**	**VRF2**
D2M01W05GP0.7	AV+CV+LHS	14.29	13.18
D2M01W05GP0.7	AV+CV	12.82	13.18
D2M01W05GP0.7	AV	9.55	9.8
D2M01W08GP0.7	AV+CV	12.4	11.7
D2M01W08GP0.7	AV	2.79	2.7
D2M02W05GP0.7	AV+LHS	7.78	11
D2M02W05GP0.7	AV+CV	8.82	11
D2M02W05GP0.7	AV	7.56	9.92
D2M02W08GP0.7	AV+CV	5.66	5.66
D2M02W08GP0.7	AV	5.66	5.66
D2M02W05GP1	AV+CV+LHS	7.24	7.24
D2M02W05GP1	AV+CV	7.24	7.24
D2M02W05GP1	AV	5.8	6.25
D2M02W05GP2.23	AV	5.8	6.25
D2M04W05GP3.16	AV+CV+LHS	2.92	2.92
D2M04W05GP3.16	AV+CV	2.92	2.92
D2M04W08GP3.16	AV+CV	2.92	2.43
D2M04W08GP3.16	AV	2.43	2.22
D2M023W0504P2.23	AV+CV	5.2	5.02
D2M025W0504P2.23	AV+CV+LHS	3.8	3.8

Table A.2. Estimated parameters of bivariate Random-Walk Metropolis-Hastings experiments.

Experiment ID	**VR Algorithms**	**Mean1**	**Mean2**	**Variance 1**	**Variance 2**	**Correlation**
D2M01W05GP0.7	AV+CV+LHS	0.499	0.499	1.249	1.248	0.2
D2M01W05GP0.7	AV+CV	0.499	0.499	1.248	1.248	0.2
D2M01W05GP0.7	AV	0.499	0.499	1.247	1.249	0.2
D2M01W05GP0.7	Naive	0.499	0.499	1.247	1.248	0.2
D2M01W08GP0.7	AV+CV	0.201	0.201	1.17	1.17	0.134
D2M01W08GP0.7	AV	0.201	0.201	1.17	1.17	0.134
D2M01W08GP0.7	Naive	0.201	0.201	1.17	1.17	0.134
D2M02W05GP0.7	AV+LHS	0.998	0.998	2	2	0.5
D2M02W05GP0.7	AV+CV	0.998	0.998	2	2	0.5
D2M02W05GP0.7	AV	0.998	0.998	2	1.99	0.5
D2M02W05GP0.7	Naive	0.998	0.998	2	1.998	0.5
D2M02W08GP0.7	AV	0.398	0.397	1.64	1.64	0.38
D2M02W08GP0.7	Naive	0.398	0.397	1.64	1.637	0.38
D2M02W05GP1	AV+CV+LHS	1.002	1.001	2	2	0.5
D2M02W05GP1	AV+CV	1.002	1.001	2	2	0.5
D2M02W05GP1	AV	1.002	1.001	2	1.99	0.5
D2M02W05GP1	Naive	1.002	1.001	1.99	1.99	0.5
D2M04W05GP3.16	AV+CV+LHS	2.007	2.006	5	5	0.8
D2M04W05GP3.16	AV+CV	2.007	2.006	5	5	0.8
D2M04W05GP3.16	Naive	2.007	2.006	5	5	0.8
D2M04W08GP3.16	AV+CV	0.803	0.802	3.58	3.57	0.72
D2M04W08GP3.16	AV	0.803	0.802	3.57	3.58	0.72
D2M04W08GP3.16	Naive	0.803	0.803	3.57	3.56	0.717
D2M023W0504GP2.23	AV+CV	1.1	1.1	2.29	2.29	0.56
D2M023W0504GP2.23	Naive	1.1	1.1	2.29	2.29	0.56
D2M025W0504GP2.23	AV+CV+LHS	1.3	1.3	3.4	3.4	0.7
D2M025W0504GP2.23	Naive	1.3	1.3	3.4	3.4	0.7

Table A.3. Standard errors of estimated parameters.

Experiment ID	VR Algorithms	Mean1 SE	Mean2 SE	Variance 1 SE	Variance 2 SE	Correlation SE
D2M01W05GP0.7	AV+CV+LHS	1.8e-3	1.9e-3	6.5e-3	6.6e-3	3.6e-3
D2M01W05GP0.7	AV+CV	1.9e-3	1.9e-3	6.6e-3	6.6e-3	3.6e-3
D2M01W05GP0.7	AV	2.2e-3	2.2e-3	6e-3	6e-3	3e-3
D2M01W05GP0.7	Naive	6.8e-3	6.9e-3	7.6e-3	7.6e-3	3.5e-3
D2M01W08GP0.7	AV+CV	1.9e-3	1.9e-3	5.7e-3	5.6e-3	3.7e-3
D2M01W08GP0.7	AV	2.4e-3	2.4e-3	6.1e-3	6e-3	3.7e-3
D2M01W08GP0.7	Naive	6.7e-3	6.5e-3	7.8e-3	7.8e-3	4.3e-3
D2M02W05GP0.7	AV+CV	3.7e-3	3.6e-3	5.8e-3	5.9e-3	1.6e-3
D2M02W05GP0.7	AV	4e-3	3.8e-3	6.2e-3	6.3e-3	3.5e-3
D2M02W05GP0.7	Naive	1.1e-2	1.2e-2	7.8e-3	7.8e-3	2e-3
D2M02W08GP0.7	AV+CV	4e-3	4e-3	9.6e-3	9.4e-3	3e-3
D2M02W08GP0.7	AV	4.2e-3	4.2e-3	9.1e-3	9.2e-3	3.7e-3
D2M02W08GP0.7	Naive	1e-2	1e-2	1.3e-2	1.3e-2	4e-3
D2M02W05GP1	AV+CV	2.6e-3	2.6e-3	5.2e-3	5.4e-3	1.4e-3
D2M02W05GP1	AV	2.9e-3	2.8e-3	6e-3	6.1e-3	3e-3
D2M02W05GP1	Naive	7e-3	7e-3	7.5e-3	7.5e-3	1.8e-3
D2M04W05GP3.16	AV+CV	7.3e-3	7.3e-3	1e-2	1e-2	3.4e-3
D2M04W05GP3.16	Naive	1.25e-2	1.2e-2	1.4e-2	1.4e-2	8.8e-4
D2M04W08GP3.16	AV+CV	6.4e-3	6.4e-3	1.5e-3	1.5e-2	1.2e-3
D2M04W08GP3.16	AV	7e-3	6.7e-3	1.6e-2	1.6e-2	3.3e-3
D2M04W08GP3.16	Naive	1.1e-2	1e-2	2.4e-2	2.3e-2	2e-3
D2M023W0504GP2.23	AV+CV	2.5e-3	2.5e-3	5.6e-3	5.7e-3	1.2e-3
D2M023W0504GP2.23	Naive	5.7e-3	5.6e-3	6.9e-3	6.8e-3	1.2e-3
D2M025W0504GP2.23	AV+CV+LHS	4.6e-3	4.6e-3	1.4e-3	1.4e-3	1.3e-3
D2M025W0504GP2.23	Naive	9e-3	9e-3	2.2e-2	2.2e-2	1.6e-3

Table A.4. VRF values for estimated mean parameter for mixture of 4 dimensional normal distributions.

Experiment Number	**VR Algorithms**	**VRF1**	**VRF2**
D4M01W05GP0.7	AV+CV+LHS	7.24	5.76
D4M01W05GP0.7	AV+CV	6.66	5.76
D4M01W05GP0.7	AV	6.15	5.29
D4M01W08GP0.7	AV+CV	5	6.15
D4M01W08GP0.7	AV	5	5.2
D4M01W05GP1	AV+CV	4.7	4.7
D4M01W05GP1	AV	4	3.69
D4M01W08GP1	AV+CV+LHS	4.47	4.7
D4M02W05GP0.7	AV+CV	4.2	3.76
D4M02W05GP0.7	AV	4.2	3.17
D4M02W08GP0.7	AV+CV	3.24	3.25
D4M02W08GP1	AV+CV	3.35	3.24
D4M02W08GP1	AV	3.35	3.24
D4M04W05GP2.23	AV+CV	2.76	2.96
D4M04W05GGP2.23	AV	2.76	2.76
D4M04W08GP.2.23	AV+CV	2.19	2.43
D4M04W08GP.2.23	AV	2.19	2.28

Table A.5. Estimated parameters of 4D Random-Walk Metropolis-Hastings experiments.

Experiment ID	**VR Algorithms**	**Mean1**	**Mean2**	**Variance 1**	**Variance 2**	**Correlation**
D4M01W05GP0.7	AV+CV+LHS	0.5	0.5	1.252	1.251	0.2
D4M01W05GP0.7	AV+CV	0.5	0.5	1.252	1.252	0.2
D4M01W05GP0.7	AV	0.5	0.5	1.252	1.252	0.2
D4M01W05GP0.7	Naive	0.5	0.5	1.247	1.248	0.2
D4M01W08GP0.7	AV+CV	0.201	0.1	1.156	1.156	0.137
D4M01W08GP0.7	AV	0.202	0.202	1.155	1.156	0.137
D4M01W08GP0.7	Naive	0.202	0.202	1.148	1.147	0.134
D4M01W05GP1	AV+CV	0.498	0.497	1.256	1.255	0.2
D4M01W05GP1	AV	0.498	0.497	1.257	1.255	0.2
D4M01W05GP1	Naive	0.498	0.493	1.239	1.242	0.2
D4M01W08GP1	AV+CV+LHS	0.2	0.2	1.162	1.158	0.137
D4M01W08GP1	Naive	0.2	0.2	1.163	1.158	0.135
D4M02W05GP0.7	AV+CV	1.01	1.01	2	2	0.5
D4M02W05GP0.7	AV	1.01	1.01	2	2	0.5
D4M02W05GP0.7	Naive	1.01	1.01	2	1.999	0.5
D4M02W08GP0.7	AV+CV	0.4	0.4	1.64	1.64	0.396
D4M02W08GP0.7	Naive	0.4	0.4	1.65	1.65	0.396
D4M02W08GP1	AV+CV	0.401	0.401	1.64	1.64	0.396
D4M02W08GP1	AV	0.401	0.401	1.64	1.64	0.397
D4M02W08GP1	Naive	0.401	0.401	1.65	1.65	0.396
D4M04W05GP2.23	AV+CV	2	2	5	5	0.8
D4M04W05GP2.23	AV	2	2	5	5	0.8
D4M04W05GP2.23	Naive	2	2	5	4.97	0.8
D4M04W08GP2.23	AV+CV	0.803	0.803	3.57	3.55	0.72
D4M04W08GP2.23	AV	0.803	0.803	3.57	3.55	0.72
D4M04W08GP2.23	Naive	0.805	0.805	3.59	3.56	0.72

Table A.6. Standard errors of estimated parameters.

Experiment ID	VR Algorithms	Mean1 SE	Mean2 SE	Variance 1 SE	Variance 2 SE	Correlation SE
D4M01W05GP0.7	AV+CV+LHS	2.3e-3	2.5e-3	5.5e-3	5.6e-3	2.4e-3
D4M01W05GP0.7	AV+CV	2.4e-3	2.5e-3	5.5e-3	5.6e-3	2.3e-3
D4M01W05GP0.7	AV	2.5e-3	2.6e-3	4.6e-3	4.7e-3	2.7e-3
D4M01W05GP0.7	Naive	6.2e-3	6e-3	6.3e-3	6.3e-3	2.9e-3
D4M01W08GP0.7	AV+CV	2.5e-3	2.3e-3	5.2e-3	5.2e-3	2.9e-3
D4M01W08GP0.7	AV	2.5e-3	2.5e-3	4.7e-3	4.8e-3	3e-3
D4M01W08GP0.7	Naive	5.6e-3	5.7e-3	6e-3	6e-3	3.8e-3
D4M01W05GP1	AV+CV	2.3e-3	2.3e-3	5.2e-3	5.4e-3	2.3e-3
D4M01W05GP1	AV	2.5e-3	2.6e-3	4.4e-3	4.5e-3	2.6e-3
D4M01W05GP1	Naive	5e-3	5e-3	6.1e-3	6.1e-3	2.9e-3
D4M01W08GP1	AV+CV+LHS	2.3e-3	2.3e-3	5.3e-3	5.3e-3	2.8e-3
D4M01W08GP1	Naive	4.9e-3	5e-3	5.6e-3	5.6e-3	3.8e-3
D4M02W05GP0.7	AV+CV	6.8e-3	6.7e-3	8.3e-3	8.4e-3	2.1e-3
D4M02W05GP0.7	AV	6.8e-3	7.3e-3	8.4e-3	8.4e-3	2.1e-3
D4M02W05GP0.7	Naive	1.4e-2	1.3e-2	1e-2	1e-2	2.8e-3
D4M02W08GP0.7	AV+CV	5e-3	4.8e-3	8.5e-3	8.7e-3	2.5e-3
D4M02W08GP0.7	Naive	9e-3	8.7e-3	1.1e-2	1.1e-2	3.1e-3
D4M02W08GP1	AV+CV	6e-3	6.1e-3	8.4e-3	8.4e-3	2.9e-3
D4M02W08GP1	AV	6e-3	6.1e-3	8.2e-3	8.3e-3	2.5e-3
D4M02W08GP1	Naive	1.1e-2	1.1e-2	1e-2	1e-2	3.1e-3
D4M04W05GP2.23	AV+CV	5.9e-2	5.8e-2	1.3e-2	1.3e-2	8e-4
D4M04W05GP2.23	AV	6e-2	6e-2	1.4e-2	1.4e-2	8e-4
D4M04W05GP2.23	Naive	1e-1	1e-1	1.9e-2	1.9e-2	1e-3
D4M04W08GP2.23	AV+CV	3e-3	3.2e-3	3e-2	3e-2	2.3e-3
D4M04W08GP2.23	AV	3.3e-3	3.3e-3	3e-2	3e-2	2.7e-3
D4M04W08GP2.23	Naive	4.9e-3	5e-3	4e-2	4e-2	3.5e-3

Table A.7. VRF values for estimated mean parameter for mixture of 10 dimensional normal distributions.

Experiment ID	VR Algorithms	VRF1	VRF2
D10M01W05GP0.7	AV+CV	4	4.5
D10M01W05GP0.7	AV	4	4
D10M01W08GP0.77	AV+CV	3.35	3.69
D10M01W08GP0.77	AV	2.66	3.35
D10M02W05GP0.77	AV+CV	2.05	2.13
D10M02W05GP0.77	AV	2	2.1
D10M02W08GP0.77	AV+CV	2.2	2
D10M02W08GP0.77	AV	2.2	2

Table A.8. Estimated parameters of 10D Random-Walk Metropolis-Hastings experiments.

Experiment ID	**VR Algorithms**	**Mean1**	**Mean2**	**Variance 1**	**Variance 2**	**Correlation**
D10M01W05GP0.7	AV+CV	0.5	0.501	1.251	1.251	0.2
D10M01W05GP0.7	AV	0.501	0.501	1.251	1.251	0.2
D10M01W05GP0.7	Naive	0.501	0.501	1.247	1.47	0.2
D10M01W08GP0.77	AV+CV	0.202	0.201	1.156	1.154	0.146
D10M01W08GP0.77	AV	0.202	0.201	1.158	1.155	0.146
D10M01W08GP0.77	Naive	0.204	0.204	1.165	1.152	0.149
D10M02W05GP0.77	AV+CV	1.005	1.007	1.994	1.99	0.496
D10M02W05GP0.77	AV	1.005	1.007	1.994	1.992	0.496
D10M02W05GP0.77	Naive	1.008	1.008	1.988	1.97	0.488
D10M02W08GP0.77	AV+CV	0.396	0.396	1.62	1.62	0.384
D10M02W08GP0.77	AV	0.396	0.396	1.62	1.614	0.384
D10M02W08GP0.77	Naive	0.396	0.396	1.64	1.65	0.39

Table A.9. Standard errors of estimated parameters.

Experiment ID	**VR Algorithms**	**Mean1 SE**	**Mean2 SE**	**Variance 1 SE**	**Variance 2 SE**	**Correlation SE**
D10M01W05GP0.7	AV+CV	5e-3	4.7e-3	6.5e-3	6.5e-3	3.2e-3
D10M01W05GP0.7	AV	5e-3	5e-3	5.7e-3	5.6e-3	3.2e-3
D10M01W05GP0.7	Naive	1e-2	1e-2	9.2e-3	9.2e-3	4.2e-3
D10M01W08GP0.77	AV+CV	4.2e-3	4.1e-3	6.1e-3	6e-3	3.4e-3
D10M01W08GP0.77	AV	4.7e-3	4.3e-3	5.8e-3	5.8e-3	3.1e-3
D10M01W08GP0.77	Naive	7.7-3	7.9e-3	8e-3	7.9e-3	4.2e-3
D10M02W05GP0.77	AV+CV	3.2e-2	3.2e-2	9.5e-3	9.4e-3	2.3e-3
D10M02W05GP0.77	AV	3.3e-2	3.3e-2	9.7e-3	9.8e-3	1.9e-3
D10M02W05GP0.77	Naive	4.6e-2	4.7e-2	1.2e-2	1.2e-2	3.3e-3
D10M02W08GP0.77	AV+CV	2.5e-2	2.6e-2	1.1e-2	1.1e-2	3.9e-3
D10M02W08GP0.77	AV	2.5e-2	2.6e-2	1e-2	1e-2	4.2e-3
D10M02W08GP0.77	Naive	3.7e-2	3.6e-2	1.5e-2	1.5e-2	5.6e-3

Table A.10. VRF values for estimated mean parameter using uniform proposal distribution.

Experiment Number	**VRF Techniques**	**VRF1**	**VRF2**
D2M01W05UP0.57	AV+CV+LHS	12.5	14
D2M01W05UP0.57	AV+CV	12.5	14
D2M01W08UP0.57	AV+CV	12.25	11.6
D2M01W08UP0.57	AV	7.84	9.3
D2M04W05UP1.73	AV+CV	2.72	2.72
D2M04W05UP1.73	AV	2.72	2.72
D4M01W05UP0.46	AV+CV	11.09	9.42
D4M01W05UP0.46	AV	10.24	8.12
D4M01W08UP0.46	AV+CV	9.49	9.49
D4M01W08UP0.46	AV	7.56	7.56
D10M01W05UP0.57	AV+CV	4.33	4.49
D10M01W05UP0.57	AV	4.33	4.33
D10M01W08UP0.57	AV+CV	3.8	3.61
D10M01W08UP0.57	AV	3.8	3.31
D10M02W05UP0.87	AV+CV+LHS	2.43	2.43
D10M02W05UP0.87	AV+CV	2	2.1
D10M02W05UP0.87	AV	2	2
D10M02W08UP0.87	AV+CV	2.37	2.56
D10M02W08UP0.87	AV	2.37	2.37

Table A.11. Estimated parameters of experiments with uniform proposal distribution.

Experiment ID	**VRF Techniques**	**Mean1**	**Mean2**	**Variance 1**	**Variance 2**	**Correlation**
D2M01W05UP0.57	AV+CV+LHS	0.5	0.5	1.249	1.248	0.2
D2M01W05UP0.57	AV+CV	0.5	0.5	1.248	1.248	0.2
D2M01W05UP0.57	Naive	0.5	0.5	1.251	1.249	0.2
D2M01W08UP0.57	AV+CV	0.2	0.2	1.14	1.14	0.13
D2M01W08UP0.57	AV	0.2	0.2	1.14	1.14	0.13
D2M01W08UP0.57	Naive	0.2	0.2	1.14	1.14	0.13
D2M04W05UP1.73	AV+CV	2.03	2.03	4.99	4.99	0.8
D2M04W05UP1.73	AV	2.03	2.03	4.99	4.99	0.8
D2M04W05UP1.73	Naive	2.03	2.03	4.99	5.01	0.8
D4M01W05UP0.46	AV+CV	0.498	0.498	1.249	1.249	0.2
D4M01W05UP0.46	AV	0.498	0.498	1.248	1.247	0.2
D4M01W05UP0.46	Naive	0.498	0.498	1.25	1.24	0.2
D4M01W08UP0.46	AV+CV	0.198	0.198	1.165	1.165	0.142
D4M01W08UP0.46	AV	0.198	0.198	1.165	1.166	0.142
D4M01W08UP0.46	Naive	0.197	0.198	1.164	1.165	0.139
D10M01W05UP0.57	AV+CV	0.5	0.5	1.246	1.253	0.2
D10M01W05UP0.57	AV	0.5	0.5	1.246	1.254	0.2
D10M01W05UP0.57	Naive	0.5	0.5	1.235	1.25	0.2
D10M01W08UP0.57	AV+CV	0.197	0.196	1.164	1.163	0.141
D10M01W08UP0.57	AV	0.197	0.196	1.164	1.162	0.141
D10M01W08UP0.57	Naive	0.197	0.196	1.173	1.176	0.142
D10M02W05UP0.87	AV+CV+LHS	1.03	1.03	2.01	2.01	0.497
D10M02W05UP0.57	AV+CV	1.03	1.03	2.01	2.01	0.497
D10M02W05UP0.57	AV	1.03	1.03	2.01	2.01	0.497
D10M02W05UP0.57	Naive	1.05	1.05	2.02	2.02	0.498
D10M02W08UP0.87	AV+CV	0.397	0.396	1.644	1.643	0.388
D10M02W08UP0.87	AV	0.397	0.396	1.643	1.643	0.388
D10M02W08UP0.87	Naive	0.395	0.396	1.698	1.71	0.409

Table A.12. Standard errors of estimated parameters.

Experiment ID	VR Algorithms	Mean1 SE	Mean2 SE	Variance 1 SE	Variance 2 SE	Correlation SE
D2M01W05UP0.57	AV+CV+LHS	1.7e-3	1.6e-3	5.7e-3	5.8e-3	2.7e-3
D2M01W05UP0.57	AV+CV	1.7e-3	1.6e-3	5.8e-3	5.8e-3	2.7e-3
D2M01W05UP0.57	Naive	6e-3	6e-3	6.1e-3	6.1e-3	3.9e-3
D2M01W08UP0.57	AV+CV	1.6e-3	1.7e-3	4.8e-3	4.8e-3	3.5e-3
D2M01W08UP0.57	AV	2e-3	1.9e-3	5.4e-3	5.4e-3	2.4e-3
D2M01W08UP0.57	Naive	5.6e-3	5.8e-3	5.9e-3	5.9e-3	3.3e-3
D2M04W05UP1.73	AV+CV	2e-2	2e-2	8.2e-3	8.3e-3	5.7e-3
D2M04W05UP1.73	AV	2e-2	2e-2	7.8e-3	7.8e-3	5.7e-3
D2M04W05UP1.73	Naive	3.3e-2	3.3e-2	1.1e-2	1.1e-2	7.2e-3
D4M01W05UP0.46	AV+CV	2.4e-3	2.6e-3	7e-3	7e-3	3.9e-3
D4M01W05UP0.46	AV	2.5e-3	2.8e-3	7e-3	7e-3	3.5e-3
D4M01W05UP0.46	Naive	8e-3	8e-3	8.4e-3	8.5e-3	4.7e-3
D4M01W08UP0.46	AV+CV	2.5e-3	2.4e-3	5.9e-3	6e-3	3.9e-3
D4M01W08UP0.46	AV	2.8e-3	2.7e-3	7.9e-3	7.9e-3	4.1e-3
D4M01W08UP0.46	Naive	7.7e-3	7.4e-3	8.2e-3	8.2e-3	4.5e-3
D10M01W05UP0.57	AV+CV	4.8e-3	4.7e-3	7.1e-3	7.1e-3	3.8e-3
D10M01W05UP0.57	AV	4.8e-3	4.8e-3	6.7e-3	6.7e-3	3.1e-3
D10M01W05UP0.57	Naive	1e-2	1e-2	8.3e-3	8.3e-3	4.2e-3
D10M01W08UP0.57	AV+CV	4.3e-3	4.4e-3	6.4e-3	6.4e-3	3.6e-3
D10M01W08UP0.57	AV	4.3e-3	4.6e-3	6e-3	6e-3	3.7e-3
D10M01W08UP0.57	Naive	8.4e-3	8.4e-3	8e-3	8e-3	4.9e-3
D10M02W05UP0.87	AV+CV+LHS	3e-2	3e-2	9.4e-3	9.5e-3	2.4e-3
D10M02W05UP0.87	AV+CV	3.3e-2	3.2e-2	9.5e-3	9.6e-3	2.3e-3
D10M02W05UP0.87	AV	3.4e-2	3.4e-2	1e-2	1e-2	2.2e-3
D10M02W05UP0.87	Naive	4.7e-2	4.7e-2	1.3e-2	1.3e-2	3e-3
D10M02W08UP0.87	AV+CV	2.4e-2	2.3e-2	1.1e-2	1.1e-2	3.8e-3
D10M02W08UP0.87	AV	2.4e-2	2.4e-3	1.1e-2	1.1e-2	3.6e-3
D10M02W08UP0.87	Naive	3.7e-2	3.7e-2	1.4e-2	1.4e-2	4.5e-3

Table A.13. VRF values for estimated mean parameter using t proposal distribution.

Experiment Number	**VRF Techniques**	**VRF1**	**VRF2**
D2M01W05TP0.9	AV+CV+LHS	6.66	5.8
D2M01W05TP0.9	AV+CV	5.33	5.25
D2M01W08TP0.9	AV+CV+LHS	6.25	6.25
D2M01W08TP0.9	AV	4.82	4.82
D2M04W05TP1.94	AV+CV	3.61	3.61
D2M04W05TP1.94	AV	2.96	2.96
D2M04W08TP1.94	AV+CV	2.76	2.44
D2M04W08TP1.94	AV	2.25	2.25
D4M01W05TP1.76	AV	2.96	2.76
D4M01W08TP1.76	AV+CV	3.69	3.17
D4M01W08TP1.76	AV	2.96	2.76
D4M04W05TP1.87	AV+CV	2.25	2.25
D4M04W05TP1.87	AV	2	2
D4M04W08TP1.87	AV	2.86	2.86
D10M01W05TP1.12	AV+CV	2.86	3.17
D10M01W05TP1.12	AV	2.76	2.86
D10M01W08TP1.12	AV+CV	4	3.8
D10M01W08TP1.12	AV	3.8	3.31
D10M02W05TP0.93	AV+CV+LHS	2.5	2.72
D10M02W05TP0.93	AV+CV	2.05	2.37
D10M02W05TP0.93	AV	2.05	2.25
D10M02W08TP0.93	AV	2.05	2.05

Table A.14. Estimated parameters of experiments with t proposal distribution.

Experiment Number	**VRF Techniques**	**Mean1**	**Mean2**	**Variance 1**	**Variance 2**	**Correlation**
D2M01W05TP0.9	AV+CV+LHS	0.5	0.5	1.249	1.249	0.198
D2M01W05TP0.9	AV+CV	0.5	0.5	1.249	1.249	0.198
D2M01W05TP0.9	Naive	0.499	0.499	1.248	1.249	0.194
D2M01W08TP0.9	AV+CV+LHS	0.2	0.2	1.16	1.16	0.138
D2M01W08TP0.9	Naive	0.2	0.2	1.156	1.154	0.138
D2M04W05TP1.94	AV+CV	2.01	2.01	4.98	4.98	0.8
D2M04W05TP1.94	AV	2.01	2.01	4.98	4.98	0.8
D2M04W05TP1.94	Naive	2.01	2.01	4.98	4.98	0.8
D2M04W08TP1.94	AV+CV	0.798	0.798	3.6	3.6	0.72
D2M04W08TP1.94	AV	0.798	0.798	3.6	3.6	0.72
D2M04W08TP1.94	Naive	0.797	0.797	3.63	3.63	0.72
D4M01W05TP1.76	AV	0.5	0.5	1.248	1.249	0.197
D4M01W05TP1.76	Naive	0.5	0.5	1.253	1.254	0.198
D4M01W08TP1.76	AV+CV	0.199	0.199	1.162	1.162	0.14
D4M01W08TP1.76	AV	0.199	0.199	1.163	1.163	0.14
D4M01W08TP1.76	Naive	0.199	0.198	1.162	1.164	0.14
D4M04W05TP1.87	AV+CV	2.02	2.02	4.989	4.989	0.8
D4M04W05TP1.87	AV	2.02	2.02	4.987	4.986	0.8
D4M04W05TP1.87	Naive	2.02	2.02	5.02	5.03	0.8
D4M04W08TP1.87	AV	0.78	0.78	3.64	3.63	0.724
D4M04W08TP1.87	Naive	0.78	0.78	3.51	3.52	0.712
D10M01W05TP1.05	AV+CV	0.501	0.499	1.265	1.261	0.2
D10M01W05TP1.05	AV	0.501	0.499	1.264	1.263	0.2
D10M01W05TP1.05	Naive	0.502	0.498	1.273	1.271	0.2
D10M01W08TP1.12	AV+CV	0.197	0.199	1.163	1.164	0.141
D10M01W08TP1.12	AV	0.197	0.199	1.164	1.165	0.141
D10M01W08TP1.12	Naive	0.197	0.198	1.173	1.171	0.142
D10M02W05TP0.93	AV+CV+LHS	0.99	0.99	1.984	1.985	0.48
D10M02W05TP0.93	AV+CV	0.99	0.99	1.984	1.985	0.498
D10M02W05TP0.93	AV	0.99	0.99	1.984	1.983	0.498
D10M02W05TP0.93	Naive	0.99	0.99	1.988	1.987	0.495
D10M02W08TP0.93	AV	0.4	0.4	1.62	1.62	0.384
D10M02W08TP0.93	Naive	0.4	0.4	1.64	1.64	0.391

Table A.15. Standard errors of estimated parameters.

Experiment ID	VR Algorithms	Mean1 SE	Mean2 SE	Variance 1 SE	Variance 2 SE	Correlation SE
D2M01W05TP0.9	AV+CV	1.9e-3	1.8e-3	4-e3	4e-3	2e-3
D2M01W05TP0.9	Naive	4.4e-3	4.1e-3	5e-3	5e-3	2e-3
D2M01W08TP0.9	AV+CV+LHS	1.6e-3	1.6e-3	3.5e-3	3e-3	2e-3
D2M01W08TP0.9	Naive	4e-3	4e-3	5e-3	5e-3	2.5e-3
D2M04W05TP1.94	AV+CV	1e-2	1e-2	9.3e-3	9.4e-3	5.4e-4
D2M04W05TP1.94	AV	1.1e-2	1.1e-2	9.9e-3	1e-2	5.7e-4
D2M04W05TP1.94	Naive	1.9e-2	1.9e-2	1.3e-2	1.3e-2	7.8e-4
D2M04W08TP1.94	AV+CV	9e-3	9.6e-3	2.1e-2	2.1e-2	1.6e-3
D2M04W08TP1.94	AV	1e-2	1e-2	1.7e-2	1.7e-2	1.6e-3
D2M04W08TP1.94	Naive	1.5e-2	1.5e-2	2.7e-2	2.7e-2	2.1e-3
D4M01W05TP1.76	AV	2.9e-3	2.9e-3	4.5e-3	4.5e-3	2.4e-3
D4M01W05TP1.76	Naive	5e-3	5e-3	6.6e-3	6.6e-3	2.6e-3
D4M01W08TP1.76	AV+CV	2.6e-3	2.8e-3	4.6e-3	4.6e-3	2.6e-3
D4M01W08TP1.76	AV	2.9e-3	3e-3	5.5e-3	5.5e-3	2.4e-3
D4M01W08TP1.76	Naive	5e-3	5e-3	6e-3	6e-3	2.8e-3
D4M04W05TP1.87	AV	6.7e-2	6.3e-2	1.3e-2	1.3e-2	8.2e-4
D4M04W05TP1.87	Naive	9e-2	9e-2	1.8e-2	1.8e-2	1.2e-3
D4M04W08TP1.87	AV+CV	3.9e-2	3.9e-2	2.8e-2	2.8e-2	2.6e-3
D4M04W08TP1.87	AV	4e-2	4e-2	2.9e-2	2.9e-2	2.4e-3
D4M04W08TP1.87	Naive	5.9e-2	5.9e-2	4.3e-2	4.4e-2	3.8e-3
D10M01W05TP1.05	AV+CV	5.9e-3	5.6e-3	6.4e-3	6.3e-3	4e-3
D10M01W05TP1.05	AV	6e-3	5.9e-3	6.5e-3	6.5e-3	3.5e-3
D10M01W05TP1.05	Naive	1e-2	9e-3	8.8e-3	8.7e-3	4.6e-3
D10M01W08TP1.12	AV+CV	4.2e-3	4.3e-3	6.7e-3	6.5e-3	3.7e-3
D10M01W08TP1.12	AV	4.3e-3	4.6e-3	7e-3	7.1e-3	4.1e-3
D10M01W08TP1.12	Naive	8.4e-3	8.4e-3	8.5e-3	8e-3	5e-3
D10M02W05TP0.93	AV+CV+LHS	2.9e-2	2.9e-2	6.2e-3	6.2e-3	2.4e-3
D10M02W05TP0.93	AV+CV	3.2e-2	3.1e-2	9e-3	9e-3	2.4e-3
D10M02W05TP0.93	AV	3.2e-2	3.2e-2	1.1e-2	1e-2	2.6e-3
D10M02W05TP0.93	Naive	4.6e-2	4.8e-2	1.2e-2	1.3e-2	3.2e-3
D10M02W08TP0.93	AV	2.3e-2	2.3e-2	1.1e-2	1.1e-3	4e-3
D10M02W08TP0.93	Naive	3.3e-2	3.3e-2	1.5e-2	1.5e-2	4.7e-3

APPENDIX B: R CODES

```
library(mvtnorm) # Used for multinormal distributions.
library(mcmcse) # Used for calculation of standard errors.
library(lhs) # Used for Latin Hypercube Sampling algorithm.
library(boot) # Used for block bootstrap technique.

rw_metropolis_hastings <- function(startvalue,
                                   covariance,
                                   alpha,
                                   mean_1,
                                   mean_2,
                                   burn_in,
                                   d,
                                   Z,
                                   mat_lhs){
  ### Applies Random-Walk Metropolis Hastings algorithm.
  ### Compares current value and the proposed value using
  ### mixture_density() function.
  ### We evaluate the density of each value.
  ### Evaluating the density and creating the chain accordingly
  ### makes a weighting impact on generated samples
  ### like we showed using f(x_i) in equations 5.4, 5.5 and 5.6.
  ### After generating these values, we can use them
  ### in any q() function to obtain estimation values
  ### like estimated mean, estimated variance and
  ### estimated correlations.
  ### q() functions in our implementation are
  ### -> q_mean_estimator()
  ### -> q_variance_estimator()
  ### -> q_covariance_estimator()
```

```
#### PARAMETERS ####
#.....startvalue -> Initial starting value of the chain.
#.....covariance -> Covariance matrix.
#.....alpha -> Mixture weights.
#.....mean_1 -> Mean vector of the first mixture component.
#.....mean_2 -> Mean vector of the second mixture component.
#.....burn_in -> Burn-in ratio.
#.....d -> Dimension.
#.....Z -> Input matrix.
#.....mat_lhs -> LHS input if LHS is used.
mixture_density <- function(x){
  f_x <- alpha*dmvnorm(x, mean = mean_1, sigma = covariance) +
    (1-alpha)*dmvnorm(x, mean = mean_2, sigma = covariance)
  return(f_x)
}

mh <- function(N = 1e6, startvalue = rep(0,d)){

  N = nrow(Z)-1
  chain <- matrix(nrow=N+1, ncol=d)
  accept <- 0
  acceptance_indicator <- numeric(N+1)
  chain[1,] <- startvalue
  uniform <- numeric()
  uniform[1] <- 0
  acceptance_indicator[1] <- 0
  for(i in 1:N)
  {
    can1 = chain[i,1] + Z[i+1,1]
    can2 = chain[i,2] + Z[i+1,2]
    can3 = chain[i,3] + Z[i+1,3]
    can4 = chain[i,4] + Z[i+1,4]
```

```
      can = c(can1, can2, can3, can4)
      ratio = min(1,mixture_density(can) / mixture_density(chain[i,]))
      uniform[i+1] <- runif(1)
      if(uniform[i+1]< ratio)
      {
        chain[i+1,] <- can
        accept = accept + 1
        acceptance_indicator[i+1] <- 1
      }else{
        chain[i+1,] <- chain[i,]
        acceptance_indicator[i+1] <- 0
      }
    }
    burnin <- N*burn_in
    chain <- chain[-c(1:burnin),]
    Z <- Z[-c(1:burnin),]
    print(paste0("Acceptance rate is : ",accept/N))
    return(list(chain,Z,
                acceptance_rate = accept/N))
  }
  res <- mh(start = startvalue, N=N)
  return(res)
}

q_mean_estimator <- function(ex){
    ### Applies the equation 5.4 at page 53.
    #### Parameters ####
    #....ex -> MCMC results of the experiment.

    ch1 <- ex[[1]][,1:2]
    muhat_x1 <- mean(ch1[,1])
    muhat_x2 <- mean(ch1[,2])
```

```
    return(c(muhat_x1, muhat_x2))
}

q_variance_estimator <- function(ex){
    ### Applies the equation 5.5 at page 53.
    #### Parameters ####
    #....ex -> MCMC results of the experiment.

    ch1 <- ex[[1]][,1:2]
    second_moment_x1 <- mean(ch1[,1]^2)
    second_moment_x2 <- mean(ch1[,2]^2)
    var_x1 <- second_moment_x1 - mean(ch1[,1])^2
    var_x2 <- second_moment_x2 - mean(ch1[,2])^2
    return(c(var_x1, var_x2))
}

q_correlation_estimator <- function(ex){
    ### Applies the equation 5.6 at page 53.
    #### Parameters ####
    #....ex -> MCMC results of the experiment.

    ch1 <- ex[[1]][,1:2]
    variances <- q_variance_estimator(ex)
    covariance_12 <- mean(ch1[,1]*ch1[,2])-mean(ch1[,1])*mean(ch1[,2])
    correlation_12 <- covariance_12 / sqrt(variances[1]*variances[2])
    return(correlation_12)
}

apply_control_variates_for_mean <- function(ex){
  ### Applies control variates algorithm for mean estimation
  ### for the first marginal of the random vector.
  #### Parameters ####
```

```
  #....ex -> MCMC results of the experiment.

  chain = as.matrix(ex[[1]][,1])
  Z = as.matrix(ex[[2]])
  cv.matrix = Z

  cv_chain <- matrix(nrow = nrow(chain), ncol = ncol(chain))
  cv_sums <- 0
  lm.matrix <- as.matrix(cbind(chain[,1], cv.matrix))
  model <- lm(lm.matrix[,1]~.,
              data = data.frame(lm.matrix[,-1]))
  coeffs <- model$coefficients[-1]
  cv_sums <- 0
  for(j in 1:length(coeffs)){
      cv_sums <- cv_sums + coeffs[j]*(cv.matrix[,j]-0)
  }
  target_cv_applied <- chain - cv_sums
  cv_chain <- target_cv_applied
  return(cv_chain)
}

apply_control_variates_for_var <- function(ex){
  ### Applies control variates algorithm for variance estimation
  ### for the first marginal of the random vector.
  #### Parameters ####
  #....ex -> MCMC results of the experiment.

  chain = as.matrix(ex[[1]][,1])
  Z = as.matrix(ex[[2]])
  cv.matrix = Z

  cv_chain <- matrix(nrow = nrow(chain), ncol = ncol(chain))
```

```
  cv_sums <- 0
  lm.matrix <- as.matrix(cbind(chain[,1]^2, cv.matrix))
  model <- lm(lm.matrix[,1]~.,
              data = data.frame(lm.matrix[,-1]))
  coeffs <- model$coefficients[-1]
  cv_sums <- 0
  for(j in 1:length(coeffs)){
      cv_sums <- cv_sums + coeffs[j]*(cv.matrix[,j]-0)
  }
  target_cv_applied <- chain^2 - cv_sums
  cv_chain <- target_cv_applied
  return(cv_chain)
}

apply_control_variates_for_cov <- function(ex){
  ### Applies control variates algorithm for covariance estimation
  ### for first two marginals of the random vector.
  #### Parameters ####
  #....ex -> MCMC results of the experiment.

  chain = as.matrix(ex[[1]][,1:2])
  Z = as.matrix(ex[[2]])
  cv.matrix = Z

  cv_chain <- matrix(nrow = nrow(chain), ncol = ncol(chain))
  cv_sums <- 0
  lm.matrix <- as.matrix(cbind(chain[,1]*chain[,2], cv.matrix))
  model <- lm(lm.matrix[,1]~.,
              data = data.frame(lm.matrix[,-1]))
  coeffs <- model$coefficients[-1]
  cv_sums <- 0
  for(j in 1:length(coeffs)){
```

```
    cv_sums <- cv_sums + coeffs[j]*(cv.matrix[,j]-0)
  }
  target_cv_applied <- chain[,1]*chain[,2] - cv_sums
  cv_chain <- target_cv_applied
  return(cv_chain)
}

se_calculator <- function(ex1, ex2){
  ### Applies non-overlapping batch means to
  ### calculate standard error of the mean
  ### estimation.
  ### Returns estimations and their standard
  ### errors.
  #### Parameters ####
  #....ex1 -> First experiment.
  #....ex2 -> Second experiment

  ch1 <- ex1[[1]]
  mcse.out0 <- mcse.mat(ch1[,1:2])
  ch2 <- ex2[[1]]
  av <- (ch1 + ch2) / 2
  mcse.out1 <- mcse.mat(av[,1:2])

  rownames(mcse.out0) <- rep("Naive", nrow(mcse.out0))
  rownames(mcse.out1) <- rep("AV", nrow(mcse.out1))

  mcse.out <- rbind(mcse.out0, mcse.out1)
  return(mcse.out)
}

varFunc <- function(x,i){var(x[i])}
```

```
t4d = list()
for(i in 1:100){
  ### repeated experiments
  ### it runs two experiments with oppposite signed input matrix
  ### in each iteration.
  input_Z = cbind(rnorm(1e6+1, sd = sqrt(5)),
                  rnorm(1e6+1, sd = sqrt(5)),
                  rnorm(1e6+1, sd = sqrt(5)),
                  rnorm(1e6+1, sd = sqrt(5)))
  ex1 = rw_metropolis_hastings(Z = input_Z,
                               startvalue = c(0,0,0,0),
                               covariance = diag(1,4,4),
                               mean_1= c(0,0,0,0),
                               mean_2 = c(4,4,4,4),
                               alpha = 0.8)
  ex2 = rw_metropolis_hastings(Z = -input_Z,
                               startvalue = c(0,0,0,0),
                               covariance = diag(1,4,4),
                               mean_1= c(0,0,0,0),
                               mean_2 = c(4,4,4,4),
                               alpha = 0.8,
                               burn_in = 0,
                               d = 4,
                               mat_lhs = NULL)
  antithetic_chains = append(ex1, ex2)
  t4d = append(t4d, antithetic_chains)

}

### Example of AV algorithm
Z = cbind(rnorm(1e6+1, sd = sqrt(5)),
          rnorm(1e6+1, sd = sqrt(5)),
```

```
          rnorm(1e6+1, sd = sqrt(5)),
          rnorm(1e6+1, sd = sqrt(5)))
ex1 = rw_metropolis_hastings(Z = Z,
                             startvalue = c(0,0,0,0),
                             covariance = diag(1,4,4),
                             mean_1= c(0,0,0,0),
                             mean_2 = c(4,4,4,4),
                             alpha = 0.8,
                             burn_in = 0,
                             mat_lhs = NULL)
ex2 = rw_metropolis_hastings(Z = -Z,
                              startvalue = c(0,0,0,0),
                              covariance = diag(1,4,4),
                              mean_1= c(0,0,0,0),
                              mean_2 = c(4,4,4,4),
                              alpha = 0.8,
                              burn_in = 0,
                              mat_lhs = NULL)

## Calculate estimates and standard error of the estimates.
av_se_estimate = se_calculator(ex1, ex2)

### Example of LHS and AV together.
mat_lhs = randomLHS(1e6+1,2, preserveDraw = F)
Z_lhs = qnorm(math_lhs, mean = 0, sd = sqrt(5))

ex3 = rw_metropolis_hastings(Z = Z_lhs,
                             startvalue = c(0,0,0,0),
                             covariance = diag(1,4,4),
                             mean_1= c(0,0,0,0),
                             mean_2 = c(4,4,4,4),
                             alpha = 0.8,
```

```
                              burn_in = 0,
                              mat_lhs = NULL)
ex4 = rw_metropolis_hastings(Z = -Z_lhs,
                              startvalue = c(0,0,0,0),
                              covariance = diag(1,4,4),
                              mean_1= c(0,0,0,0),
                              mean_2 = c(4,4,4,4),
                              alpha = 0.8,
                              burn_in = 0,
                              mat_lhs = NULL)

av_se_estimate = se_calculator(ex3, ex4)
naive_var_se_estimate=tsboot(ex3[[1]][,1],
                              varFunc, l = 93,
                              R = 100, sim = 'fixed')

### Example of applying Control Variates algorithm
##  for the first marginal of the random vector.
ex1_mean_cv = apply_control_variates_for_mean(ex1)
ex1_var_cv = apply_control_variates_for_var(ex1)
ex2_cov_cv = apply_control_variates_for_cov(ex1)
```