Bayesian Analysis via MCMC (MATH30000 Double Project)

Jingqi Zhu (10807240)

supervisor: Christiana Charalambous

May 2022

Abstract

Bayesian analysis is a statistical inference paradigm that is widely adopted across fields of scientific research. The report presents various techniques of Bayesian analysis using Markov chain Monte Carlo methods. Chapter 1 introduces a general picture of Bayesian analysis including Bayesian workflow, prior selection, checking and some history. Due to the posterior intractability and the curse of dimensionality, true posterior can be hard to evaluate and Monte Carlo method is adopted to provide parameter estimates. As Monte Carlo method is based on sampling, various classical simulation methods are developed in chapter 2, such as inverse cdf methods, transformation methods, accept-reject methods and importance sampler. However, these methods are not always applicable for an arbitrary high dimensional distribution. Therefore, Markov chain Monte Carlo is adopted to generalize accept-reject method, boost efficiency of algorithms and provide decomposition for high dimensional problems. In chapter 3, Metropolis-Hastings algorithm and Gibbs sampler are derived, discussed and illustrated in details. The normal and Gaussian mixture examples are used to compare among different algorithms, demonstrate parameter tuning and diagnostics. Finally, more techniques are briefly mentioned for broadening sights, including variable selection, posterior predictive checking, RJMCMC, HMC, SMC and ABC.

Contents

Chapter 1

Introduction

1.1 Bayesian Analysis

Bayesian analysis is a statistical inference paradigm that combines both prior knowledge and observed data based on Bayes' theorem. The general idea of Bayesian approach is to assume a prior probability distribution for parameters $\boldsymbol{\theta}$, and then update it with likelihood to obtain the posterior - the conditional probability of $\boldsymbol{\theta}$ given observed data $\boldsymbol{x}$. The mission of Bayesian analysis is to make inferences about parameters based on the posterior distribution.

The typical Bayesian workflow mainly consists of three parts: i) formalizing pre-existing information about parameters in a model using prior distribution, ii) deriving the likelihood function from observed data, iii) combining prior distribution and likelihood function into the form of posterior distribution via Bayes' theorem.

The prior distribution of parameters characterises the existing experiences about the unknown parameters $\boldsymbol{\theta}$ by using a presumed probability density function (or probability mass function) $p(\boldsymbol{\theta})$ for $\boldsymbol{\theta}$.

The likelihood $p(\boldsymbol{x}|\boldsymbol{\theta})$ is the joint probability density function (or probability mass function) of observed data. An alternative expression is $\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{x})$ to address that it is not a probability density function of $\boldsymbol{\theta}$. The likelihood can be interpreted as the plausibility of generating observed data $\boldsymbol{x}$ given parameters $\boldsymbol{\theta}$. Therefore, it is a measure of the amount of information about parameters in the model brought by data.

The posterior distribution of $\boldsymbol{\theta}$ given data x represents the updated degrees of belief regarding $\boldsymbol{\theta}$ after observing sample $\boldsymbol{x}$. Once prior and likelihood are obtained, posterior distribution is accessible via Bayes' theorem

$$p(\boldsymbol{\theta}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\boldsymbol{x})} = \frac{p(\boldsymbol{x}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{\int_{\boldsymbol{\Theta}} p(\boldsymbol{x}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}} \tag{1.1}$$

where $\boldsymbol{\Theta}$ is the parameter space of $\boldsymbol{\theta}$. The Bayes' theorem divides the uncertainty of the model into two parts: i) aleatoric uncertainty $p(\boldsymbol{x}|\boldsymbol{\theta})$, which can not be reduced by acquiring more observations, as it is the uncertainty due to randomness of data generating process; ii) epistemic uncertainty $p(\boldsymbol{\theta})$, which is due to our lack of knowledge and can be reduced by observations. The numerator

$$p(\boldsymbol{x}, \boldsymbol{\theta}) = p(\boldsymbol{x}|\boldsymbol{\theta})p(\boldsymbol{\theta}) \tag{1.2}$$

is called the full probability distribution, because this is the joint distribution of parameters and data. As in Bayesian statistics, there are no differences between observable data and underlying parameters, all of them are equally regarded as random variables. The denominator

$$p(\boldsymbol{x}) = \int_{\boldsymbol{\Theta}} p(\boldsymbol{x}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta} \tag{1.3}$$

is called the marginal likelihood of underlying model or prior predictive distribution for data. Since it is independent of $\boldsymbol{\theta}$, it does not carry information about parameters directly and can be taken as a normalising constant. Thus, the formula can be simplified to

$$p(\boldsymbol{\theta}|\boldsymbol{x}) \propto p(\boldsymbol{x}|\boldsymbol{\theta})p(\boldsymbol{\theta}) \tag{1.4}$$

i.e. given prior and likelihood, posterior is known up to a constant. However, it is still important to get the normalizing constant to obtain the exact posterior distribution. Unfortunately, this constant is analytically intractable for most of practical cases. The problem of posterior intractability has stumbled Bayesian methods for a long period of time before statisticians found MCMC techniques can circumvent the problem by sampling from posterior distribution directly.

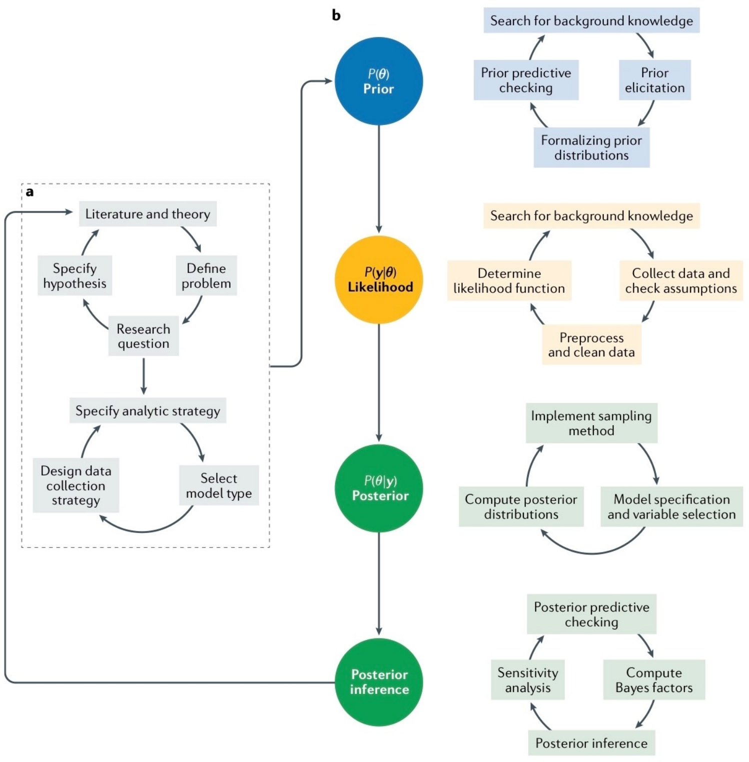

a
Literature and theory
Specify hypothesis
Define problem
Research question
Specify analytic strategy
Design data collection strategy
Select model type
b
P(θ) Prior
Search for background knowledge
Prior predictive checking
Prior elicitation
Formalizing prior distributions
P(y|θ) Likelihood
Search for background knowledge
Determine likelihood function
Collect data and check assumptions
Preprocess and clean data
P(θ|y) Posterior
Implement sampling method
Compute posterior distributions
Model specification and variable selection
Posterior inference
Posterior predictive checking
Sensitivity analysis
Compute Bayes factors
Posterior inference

Figure 1.1: General Bayesian research cycle, taken from van de Schoot et al. (2021).

In comparison with the frequentist framework, Bayesian framework considers unknown parameters as random variables in order to make probabilistic inferences about them. There is no difference between observables and parameters of a statistical model. Also, Probabilities are interpreted as uncertainties for model parameters instead of expected long-running outcomes of identical repeated experiments. The primary focus of Bayesian inference is on estimating the posterior distribution of model parameters. However, as we will discuss in section 1.3, an explicit posterior distribution is usually intractable because the normalizing constant is an integral that can not be expressed as closed form, and complexity and high-dimensionality of parameter space exacerbate the problem. Furthermore, most summary statistics for Bayesian inference can be transformed into a posterior expectation of a function of parameters, which is essentially an integration as well. Thus, the challenge of Bayesian analysis is essentially an integration or averaging problem, in contrast to frequentist statistics where the main challenge is optimization, like finding maximum likelihood estimators. (Strimmer (2021))

Inferences from obtained posterior are optimal as the conditional probability of parameter given observed data reflects one's updated knowledge that balanced available experience with newly acquired data. This procedure can be repeated by regarding the posterior as a new prior and then update it with further data. As shown in Figure 1.1, the nature of such updating process is consistent with the general scientific research cycle and human epistemological development.

1.2 Examples of Bayesian Analysis

1.2.1 Beta-Binomial Model

The Beta-Binomial model is the model used to estimate a proportion p.

- Beta prior: As the support of p is $[0, 1]$, existing knowledge of p can be formulated into a beta prior distribution $p \sim Beta(\alpha, \beta)$ with $\alpha, \beta > 0$. The probability density of $Beta(\alpha, \beta)$ is

  $$f(p|\alpha, \beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} p^{\alpha-1}(1-p)^{\beta-1},\ p \in [0, 1] \tag{1.5}$$

  where the Euler's Gamma function

  $$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt \tag{1.6}$$

  The prior mean

  $$\mu_{prior} = \frac{\alpha}{\alpha+\beta} \tag{1.7}$$

- Binomial likelihood: The observed data of size n is regarded as an outcome from n repeated Bernoulli random trials, where probability mass function is

  $$P(x_i = 1) = p, \quad P(x_i = 0) = 1 - p \tag{1.8}$$

  Thus, the likelihood $\boldsymbol{x}|p \sim Bin(n, p)$

  $$f(\boldsymbol{x}|p) = \tbinom{n}{\sum x_i} p^{\sum x_i}(1-p)^{n-\sum x_i},\ p \in [0, 1] \tag{1.9}$$

  and the maximum likelihood estimate

  $$\mu_{ML} = \frac{\sum x_i}{n} \tag{1.10}$$

- Beta posterior: The posterior

$$f(p|\boldsymbol{x}) \propto f(p)f(\boldsymbol{x}|p) \propto p^{\alpha+\sum x_i - 1}(1-p)^{\beta+n-\sum x_i - 1} \tag{1.11}$$

  Thus, $p|\boldsymbol{x} \sim Beta(\alpha + \sum x_i, \beta + n - \sum x_i)$ and the posterior mean

$$\mu_{posterior} = \frac{\alpha + \sum x_i}{\alpha + \beta + n} \tag{1.12}$$

  This form of posterior distribution suggests that the prior acts as pseudo-data in the model, where α is the pseudo-count of success and $\alpha + \beta$ is the implicit sample size. The effective sample size can be increased by the prior information in Bayesian paradigm, even though those $\alpha + \beta$ samples are actually not observed. This property can help to regularize the model and prevent overfitting.

  In addition, the posterior mean can be regarded as a linear shrinkage of maximum likelihood estimate of the mean

$$\mu_{posterior} = \lambda\mu_{prior} + (1-\lambda)\mu_{ML} \tag{1.13}$$

  where shrinkage intensity $\lambda = \frac{m}{m+n}$, m is the sample size of pseudo-data and n is the sample size of observed data. This means that Bayesian estimates are biased but asymptotically unbiased.

Example 1.2.1 (Bayes estimate is a linear shrinkage of MLE) *The observed number of successful surgeries for three hospitals are shown as follow, the question is which one is the best.*

	A	B	C
success	10	48	186
total	10	50	200

In frequentist perspective, this is viewed as repeated Bernoulli trails with three probabilities of success. Since the proportion of success for A, B and C are 1, 0.96 and 0.93 respectively, it seems that A has the highest success probability of 1. However, the total number of observed surgeries for A is too small to make this conclusion convincing.

In Bayesian statistics, people can set a non-informative prior $Beta(1,1)$ *on each probabilities of success. Combined with Bernoulli likelihood, the posterior probabilities of success are 0.92, 0.94 and 0.93, which indicates B has the greatest success probability. The amount of data reflects the amount of information about the underlining parameter. The Bayesian view of data analysis automatically incorporates sample size into the inference making process.*

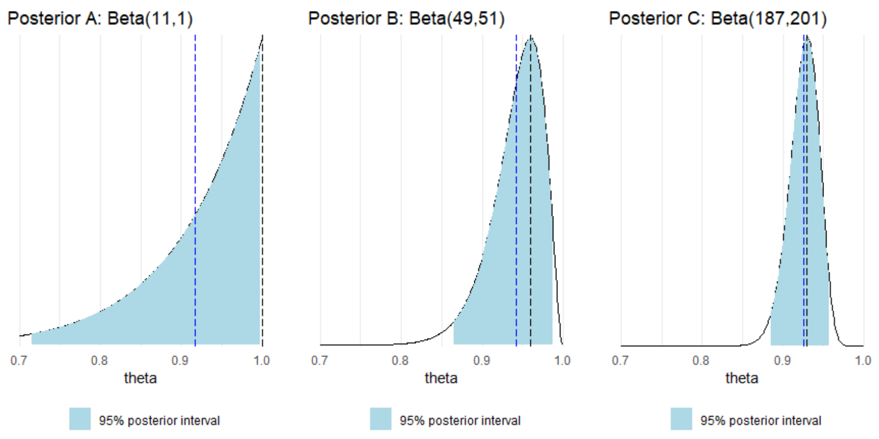


Figure 1.2: Prior performs like pseudo-data and incorporates sample size into inference via linear shrinkage: the blue dash line denotes posterior mean, the black dash line denotes MLE.

1.2.2 Normal-Normal Model

The normal-normal model is used to estimate the mean of a normal distribution with known variance.

- Normal prior: Assume the prior to be a normal distribution $\theta \sim N(\mu_0, 1/\tau_0)$

$$\begin{aligned} f(\theta) &= \sqrt{\frac{\tau_0}{2\pi}} exp(-\frac{\tau_0}{2}(\theta-\mu_0)^2) \\ &\propto exp(-\frac{\tau_0}{2}(\theta^2 - 2\mu_0\theta)) \end{aligned} \tag{1.14}$$

- Normal likelihood: Let $x_1, \dots, x_n$ be a sample from normal distribution $N(\theta, 1/\tau)$ with known precision τ. The likelihood

$$\begin{aligned} f(\boldsymbol{x}|\theta) &= \prod_{i=1}^{n} \sqrt{\frac{\tau}{2\pi}} exp(-\frac{\tau}{2}(x_i-\theta)^2) \\ &\propto exp(-\frac{\tau}{2}\sum_{i=1}^{n}(x_i-\theta)^2) \\ &= exp(-\frac{\tau}{2}(n\theta^2 - 2\theta\sum_{i=1}^{n} x_i + \sum_{i=1}^{n} x_i^2)) \\ &\propto exp(-\frac{\tau}{2}(n\theta^2 - 2n\bar{x}\theta)) \end{aligned} \tag{1.15}$$

  where $\bar{x} = \sum_{i=1}^{n} x_i/n$

- Normal posterior: The posterior distribution

$$\begin{aligned} f(\theta|\boldsymbol{x}) &\propto f(\theta)f(\boldsymbol{x}|\theta) \\ &\propto exp(-\frac{\tau_0}{2}(\theta^2 - 2\mu_0\theta))exp(-\frac{\tau}{2}(n\theta^2 - 2n\bar{x}\theta)) \\ &= exp(-\frac{\tau_0 + n\tau}{2}(\theta^2 - 2\frac{\mu_0\tau_0 + n\bar{x}\tau}{\tau_0 + n\tau}\theta)) \end{aligned} \tag{1.16}$$

the posterior

$$p|\boldsymbol{x} \sim N(\frac{\mu_0\tau_0 + n\bar{x}\tau}{\tau_0 + n\tau}, \frac{1}{\tau_0 + n\tau}) \tag{1.17}$$

i.e. the posterior mean is the weighted average of the prior mean and the sample average weighted by their precision, the posterior precision is the sum of prior precision and data precision.

1.3 Prior Elicitation, Formalizing and Checking

The selection of prior distribution is of great significance in Bayesian statistics. An inappropriate prior can have an adverse impact on posterior inferences. Priors may take various distributional forms, like uniform, normal, Poisson, etc. The parameters of prior distributions are called hyperparameters and they control the informativeness of priors. An informative prior can shift posterior away from likelihood if it does not overlap well with the data. Whereas, a diffuse prior let data speak for themselves and can result in more alignment with likelihood. A Statistician may want to use an informative prior when there are some implications suggesting restrictions on particular parameters or relations between parameters. And diffuse priors can be useful if there is a complete lack of uncertainty of certain parameters or there should be a placeholder before further informative data analysis. An improper prior usually refer to those do not integrate to one. For the sake of better understanding the impact that a prior may have on the associated posterior, prior selection always entails prior sensitivity analysis, especially when the sample size is small.

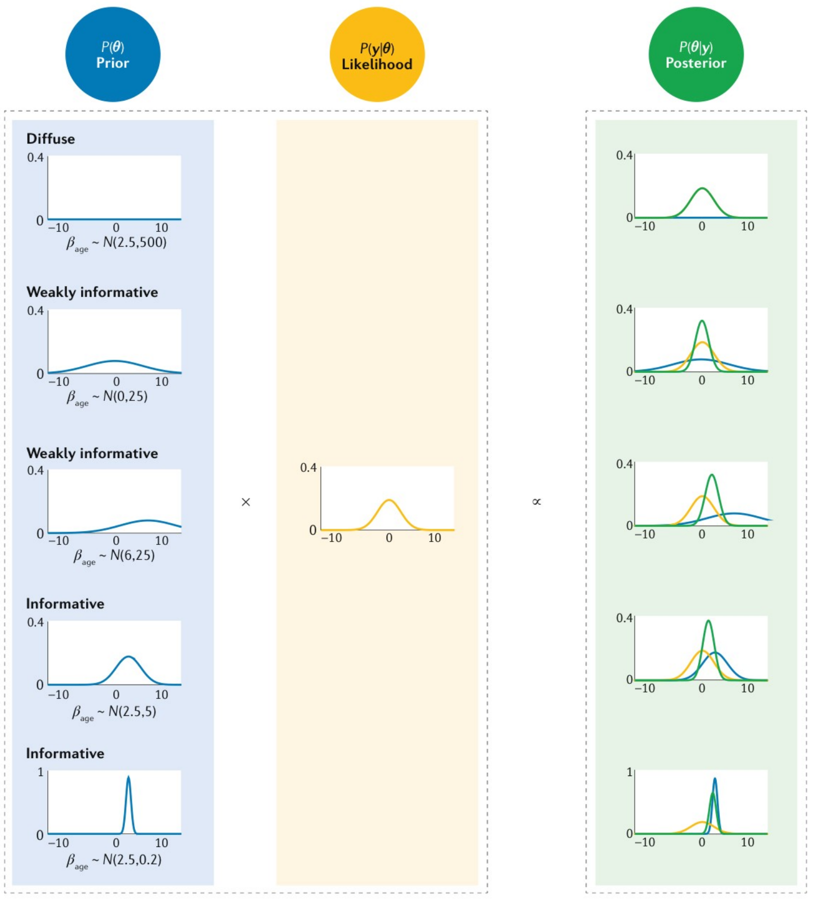


Figure 1.3: Impact of different priors on the same likelihood, taken from van de Schoot et al. (2021).

A typical family of priors is called conjugate priors where the posterior follows the same distributional form as the prior. Common examples are normal prior for normal data, beta prior for binomial data and gamma prior for exponential data. The greatest benefit of using conjugate priors is that posterior can be obtained directly from prior and likelihood without calculating the normalizing constant. So it is mathematically convenient to use conjugate priors. However, conjugacy exists almost only for exponential family distributions, and is only concerned with a particular parametric form of data distribution. For real-life modeling, it is possible that the model is too high-dimensional to determine associated conjugate prior or researchers do not want to use conjugate prior because there is not enough benefit to do so. In fact, conjugate priors used to be important as it allows efficient partial analytic marginalization which brings computational benefits to probabilistic programming such as BUGS (Bayesian inference Using Gibbs Sampling). But more recently, with the development of Hamiltonian Monte Carlo used in Stan, the computational advantage of conjugate prior diminishes. The most common case in favour of conjugate prior is that there is not enough prior information about tails, then it is useful to choose priors with tails leading to computational advantages as it will not affect much the inference. (Gelman et al. (1995)) Overall, though people still copy conjugate priors from old models, it is worthy to think what is the distribution to best describe prior information.

To construct an appropriate prior, common strategies include asking a group of experts for hyperparameter values, referring previous publications or using meta-analysis. Prior elicitation can also involve adopting data-based priors. i.e. the hyperparameters are obtained from data via, for example, maximum likelihood. Particularly, to avoid double-dipping problem - the same data set is used to derive prior and likelihood, it is recommended to implement a hierarchical modeling strategy: only allow hyperparameters of prior to be data-driven.

After formalizing prior distribution, it is vital to check whether the model can be considered to be generating the observed data. This is mainly done by prior predictive checking, which helps to improve the understanding of specified priors on possible observations. The prior predictive distribution implies the distributions of possible samples if the model is true. The prior predictive checking checks the compatibility between the prior predictive distribution with true data-generating distribution by using kernel density estimation or calculating the prior predictive p-value. Ideally, the prior predictive distribution would be analogous to the distribution of observed data. If there are two plausible proposed priors, in order to determine the more precise prior, it is advisable to apply the Bayes factor to compare these priors to determine a better one.

There are a substantial amount of more profound discussions about prior formalizing and predictive checking in Bayesian statistics. However, the effect of prior will diminish as the sample size increases, and many statisticians consider MCMC algorithms to be robust against weakly informative priors, so too much discussion for prior is inconsequential.

1.4 Approximate Inference

The posterior distribution is the core object of Bayesian analysis. Any features of the posterior distribution are legitimate for Bayesian inference, such as moments, quantiles and highest density regions. All these quantities can be regarded as posterior expectations of functions of $\boldsymbol{\theta}$ (Gilks et al. (1995)). The posterior expectation of function $f(\boldsymbol{\theta})$ is

$$E[f(\boldsymbol{\theta})|\boldsymbol{x}] = \frac{\int_{\boldsymbol{\Theta}} f(\boldsymbol{\theta})p(\boldsymbol{x}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}}{\int_{\boldsymbol{\Theta}} p(\boldsymbol{x}|\boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}} \tag{1.18}$$

For example, if the feature of interest is posterior mean, then $f(\boldsymbol{\theta}) = \boldsymbol{\theta}$; If people want to get the predictive distribution of a future observation x_0 given existing data $\boldsymbol{x}$, $p(x_0|\boldsymbol{x})$, then $f(\boldsymbol{\theta}) = P(x_0|\boldsymbol{x}, \boldsymbol{\theta})$.

Deriving this posterior expectation of $f(\boldsymbol{\theta})$ is not an easy task, as the integrals in the numerator and denominator are often analytically intractable. With the increase of dimension, the problem of posterior intractability is exacerbated, since it is almost impossible to integrate the objective function over the whole parameter space or some hyperplanes of parameter space. Therefore, the only way out is to resort to numerical integration methods.

1.4.1 Classical Methods for Integral Approximation

One of the most naive numerical method for integral approximation is Riemann sum approximation. Originated from the definition of Riemann integration, the method approximates integrals by partitioning domains and estimating the area below objective functions using the sum of corresponding rectangular areas. However, this method loses control of the error when the dimension rises. To illustrate this problem, a proof in appendix A shows that the error of Riemann sum approximation using n partitions is $O(n^{-\frac{1}{d}})$, which grows exponentially as dimension increases. There are alternative approximation methods for integration, such as Trapezium rule and Simpson's rule. Although they reshape the partitioned areas via interpolation, the idea of partitioning and summing areas is the essentially the same as Riemann sum. Therefore, all these classical numerical integration methods suffer from 'the curse of dimensionality'.

1.4.2 Monte Carlo Method

Monte Carlo method is a stochastic simulation method developed by Stanislaw Ulam and John Von Neumann after World War II. To better illustrate Monte Carlo method, we rephrase the integration problem as

$$I = \int_{\boldsymbol{\Theta}} f(\boldsymbol{\theta})\pi(\boldsymbol{\theta})d\boldsymbol{\theta} = E_{\pi}[f(\boldsymbol{\theta})] \tag{1.19}$$

where in Bayesian analysis, $\pi(\boldsymbol{\theta})$ is usually the posterior distribution of some parameter and $f(\boldsymbol{\theta})$ is a transformation from $\boldsymbol{\theta}$ to any interested quantity expressed as a function f of $\boldsymbol{\theta}$.

The general idea of Monte Carlo simulation is to use empirical average to estimate the population expectation. Assume that it is possible to obtain an independent sample $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_n$ from distribution π, calculate f point-wisely and the Monte Carlo estimator

$$\hat{I}(n) = \frac{1}{n}\sum_{i=1}^{n} f(\boldsymbol{\theta}_i) \tag{1.20}$$

This estimator is unbiased and consistent by the Law of Large Numbers. If $Var_{\pi}[f(\boldsymbol{\theta})] = \int_{\boldsymbol{\Theta}}(f(\boldsymbol{\theta}) - I)^2\pi(\boldsymbol{\theta})d\boldsymbol{\theta}$ is finite, by Central Limit Theorem

$$\sqrt{n}(\hat{I}(n) - I) \sim N(0, Var_{\pi}[f(\boldsymbol{\theta})]) \tag{1.21}$$

where

$$\hat{\sigma}_n^2 = \frac{1}{n-1}\sum_{i=1}^{n}(f(\boldsymbol{\theta}_i) - \hat{I}(n))^2 \tag{1.22}$$

can be used as a proxy for $Var_{\pi}[f(\boldsymbol{\theta})]$ in practice. Thus, Monte Carlo method is guaranteed to converge as sample size increases, the standard error of the estimator is

$$s.e.(\hat{I}_n) = \frac{\hat{\sigma}_n^2}{\sqrt{n}} \sim O(n^{-\frac{1}{2}}) \tag{1.23}$$

The fact that Monte Carlo method converges with rate $O(n^{-\frac{1}{2}})$ uniformly makes it an optimal method for high-dimensional cases, as it is theoretically faster than classical methods when $d \geq 3$.

Example 1.4.1 (Monte Carlo integration for $\int_0^1 (cos(50x) + sin(20x))^2 dx$) *Let $f(x) = \int_0^1 (cos(50x) + sin(20x))^2 dx$. Then we can rewrite the integration as $\int_0^1 f(x)\pi(x)dx = E_\pi[f(x)]$, where π is uniform distribution. Thus, Monte Carlo estimate can be obtained by generating uniform u_i, and $\hat{I}(n) = \frac{1}{n}\sum_{i=1}^{n} f(u_i)$.*

The Monte Carlo estimates and 95% confidence intervals are shown in Figure1.4, which is a demonstration that Monte Carlo method converges to the true value and the error is $O(n^{-\frac{1}{2}})$.

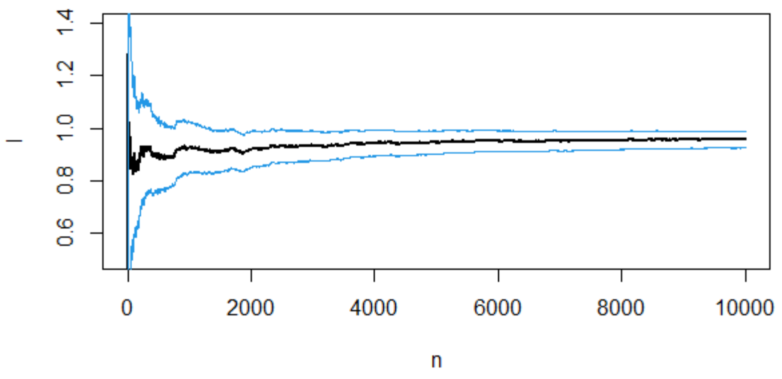


Figure 1.4: Monte Carlo estimates for $\int_0^1 (cos(50x) + sin(20x))^2 dx$ and corresponding 95% confidence intervals.

However, Monte Carlo is a sampling-based approach, to implement it for integral approximation for posterior, how to draw iid sample from the posterior is still a great obstacle. Generally, this typical sampling problem can be described as how to obtain an iid sample from an arbitrary target distribution π. In the next few chapters, we will focus on various simulation method for random variable simulation as well as why Markov Chain Monte Carlo (MCMC) technique is of central importance in sampling from the posterior.

1.4.3 Variational Method

It is important to note that Monte Carlo method is not the only feasible way towards posterior approximation. An alternative approach is variational method, also known as variational Bayes or variational inference, which originally comes from machine learning. Rather than adopting simulation-based approximation for exact posterior, variational method approximates the intractable posterior by a tractable standard probability distribution. Then it optimizes the variational parameters of the approximating distribution such that the Kullback-Leibler divergence from the real posterior is minimized. Finally, the desired posterior can be approximated by optimized standard distribution. Therefore, variational method reforms the inference problem into an optimization problem and numerical optimization algorithms like stochastic gradient descent can be applied.

Compared to Monte Carlo method (mainly MCMC), variational method tends to be faster and easier to scale to large data. However, it is not guaranteed to converge asymptotically to the target density as Monte Carlo method and is studied less rigorously than MCMC, some of its statistical properties are still less well understood. In brief, variational method is suitable for large data sets and scenarios where we want to quickly explore many models; Monte Carlo is suitable for smaller data sets where we prefer a more precise result via intensive computation.(Blei et al. (2017))

For the past few decades, Monte Carlo techniques based on MCMC have been so widely studied, extended and applied that they became a dominant paradigm in Bayesian statistics. Although there are a great number of profound discussions for variational method, those theories are beyond the scope of this report and the following chapters follow the main stream: posterior inference based on Monte Carlo method.

1.5 History of Bayesian Analysis

Bayesian analysis is named after the father of Bayesian statistics - Thomas Bayes (1701-1761) who stated probabilistic limits could be placed on an unknown event in his work, but his paper was published after his death in 1763. From late 18th century to early 19th century, Pierre-Simon Laplace (1749–1827) developed the Bayes' probability theory into 'inverse probability'/'Principle VI' and firstly apply this in medical statistics. At that time Bayesian inference only used uniform priors by Laplace's principle of insufficient reason.

However, in early 1920s, the boom of frequentist methods heavily influenced and dominated the field of statistical inference and the most famous leading statistician R.A.Fisher and Jerzy Neyman were opposed to Bayesian methods due to the subjectivity in an objective discipline. In 1925, Fisher criticized: 'The theory of inverse probability is founded upon an error, and must be wholly rejected'. Ironically, Fisher was the first person who used its commonly known name 'Bayesian' and his thought at the time was very similar to Bayes' theory. Despite criticisms, in 1931, de Finetti showed the joint distribution of exchangeable events could be represented by a mixture via Bayes' theorem, which was a strong justification for Bayesian approaches. Also, during the World War II, Alan Turing tried to use Bayesian methods to break enciphered messages of German navy. However, the heavily required mathematics and computational force limited the development of Bayesian analysis until the second half of 20th century. From 1970, there was a dramatic growth in research and applications of Bayesian methods. On the one hand, the discovery of MCMC algorithms like Metropolis-Hastings algorithm solved a great number of computational problems. On the other hand, the popularization of modern computers enabled accurate, fast and convenient computation for those computation-hungry MCMC algorithms.

In another circle of academy, a group of physicist, mainly Stan Ulam and John Von Neumann developed Monte Carlo algorithms for the study of neutron diffusion, including accept-reject sampling and importance sampling. In 1949, Stan Ulam and Nick Metropolis published the first paper on Monte Carlo simulation. Shortly afterwards, together with the Tellers and the Rosenbluths, Metropolis proposed the Metropolis algorithm (one of the oldest MCMC algorithms) in 1953 and Monte Carlo method became very popular in physic. In 1970, the Metropolis algorithm was generalized by Hastings, so the Metropolis-Hastings algorithm appeared. It is generally agreed that MCMC was first adopted to statistics by Gelfand and Smith's seminal work in 1990.(Andrieu et al. (2003))

After 1990s, more and more high-dimensional data sets from complex sciences, like genomics, came into statistical research field. As the drawbacks of frequentist approaches gradually appeared in high-dimensional cases, Bayesian statistics was boosted once more. In a survey in 2000, Metropolis algorithm was marked as one of the ten algorithms with greatest influence on the development and practice of science and engineering in the 20th century. Nowadays, applications of Bayesian analysis can be found in various fields, including ecology, genetics, social and behavioural sciences, etc.

1.6 Overview of the Report

This report concentrates on necessities of MCMC methods in the framework of Bayesian analysis. Chapter 1 introduces basic concepts and computational challenges of Bayesian analysis - sampling and integration. Chapter 2 focuses on the sampling problem and discusses various classical random variable simulation methods, including uniform generator, inverse cdf method, general transformation methods, accept-reject method and importance sampling. Section 3 delve into Markov Chain Monte Carlo methods, where building blocks of Markov Chain are reviewed, and then Metropolis-Hastings algorithm and Gibbs sampling are derived, discussed and compared thoroughly. Finally, the report concludes by a short summary and briefly looking into other procedures (variable selection), MCMC methods (RJMCMC, HMC) and non-MCMC methods (SMC, ABC) in section 5.

Chapter 2

Classical Simulation Methods

As we discussed above, the problem of how to sample from posterior distribution stumbled the implementation of Monte Carlo integration. In this chapter, we consider the generic problem of how to practically generate random variables with standard or nonstandard distribution π by computer program.

2.1 Uniform Simulation

The first task of numerical simulation is how to produce randomness. Since computers are designed to be entirely deterministic machine, it is impossible to generate real randomness in the probabilistic sense. However, it is possible to generate pseudo random numbers, which is a series of deterministic numbers behaving like random numbers in the statistical sense.

One of the oldest and most well-known methods of pseudo random generation (PRNG) is the linear congruential generator (LCG), which is very fast to implement and requires little memory. The algorithm is defined as: Given a seed X_0, for any $n \geq 1$

$$X_n = (aX_{n-1} + c) \ mod \ m \tag{2.1}$$

for some integers a, c and m. If these numbers are chosen properly, the series would have period m for any seed. Define sequence U to be $U_n = X_n/m$, then the sequence behaves like uniform random variable $U \sim \mathcal{U}_{[0,1]}$.

Example 2.1.1 (ANSI C implementation (Saucier, 2000.)) *Implement the LCG with $m = 2^{32}$, $a = 1103515245$, $c = 12345$ to simulate 10000 samples from uniform distribution.*

The plots for randomness checking are shown in figure 2.1, where the histogram performs analogous to uniform density. The plot of adjacent pairs filled the whole region without any pattern and autocorrelation function cuts off after lag 1. Thus, the generated sequence can be considered as uniform iid random variables.

A disadvantage of LCG is that it is very sensitive to the choice of a, c. For instance, any LCG with $a = 1$ and $c = 1$ obviously does not behave anything

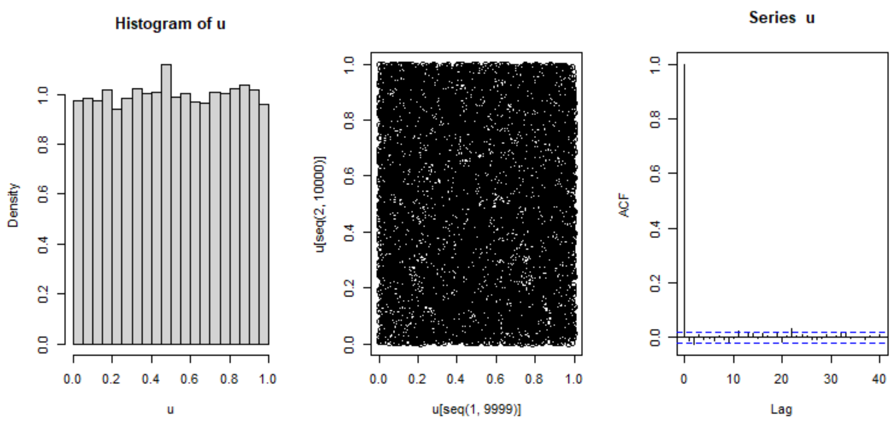


Figure 2.1: Randomness checking of LCG: histogram, plot of adjacent pairs and autocorrelation

like random at all. In practice, the most commonly used PRNG is Mersenne Twister, an algorithm based on the Mersenne prime $2^{19937}-1$, which is also the default program in R and Python.

As we will see in the following sections, most of the simulation methods under the context of Monte Carlo integration rely on the possibility of generating iid uniform random variables. Due to the good performance of modern PRNGs, uniform variable generation are often regarded to be 'perfect' when using it to obtain other random variables.

2.2 Inverse CDF Method

The cdf of an arbitrary distribution π is a function mapping the support of π to $[0,1]$. The idea of inverse cdf method is to first sample uniformly from $[0,1]$ and then map it to the supoort of π via inverse cdf. Consider random variable X with associated cdf

$$F(x) = P(X \leq x) \tag{2.2}$$

Since cdf is not strictly increasing and only right continuous, there is no general inverse for cdf. But we can define the generalized inverse

$$F^{-}(u) = \inf\{x \in R | F(x) \geq u\} \tag{2.3}$$

Let F be a cdf of distribution π, $U \sim \mathcal{U}_{[0,1]}$, $F(X) = U$, then

$$P(F^{-}(U) \leq x) = P(U \leq F(x)) = F(x) \tag{2.4}$$

i.e. $X = F^{-}(U)$ follows distribution π.

Example 2.2.1 (Simulate $X \sim Exp(1)$ via inverse cdf) *The cdf of $Exp(1)$ is $F(x) = 1 - e^{-x}$. The inverse cdf $F^{-}(u) = -log(1-u)$. Then $X = F^{-}(u) = -log(1-u) \sim Exp(1)$ can be verified by comparing the histogram and density plot with the exponential random number generator in R (Figure 2.2).*

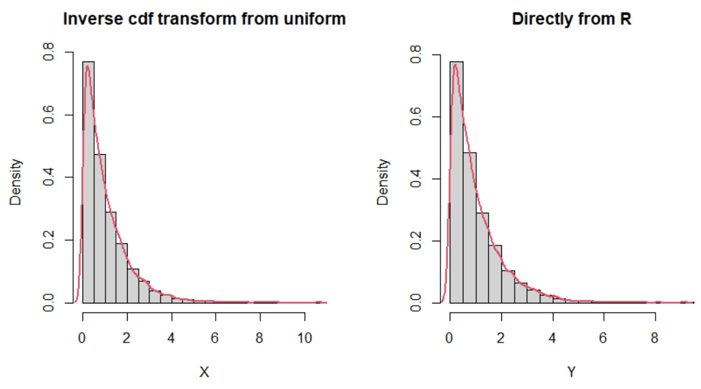


Figure 2.2: Histograms and density plots of random variables using inverse cdf and default function $runif()$.

As generating uniform variables is already a trivial task for modern computers, the inverse cdf method enables us to generate arbitrary number of random variables with distribution π as easy as drawing uniform variables. However, this method requires point-wise knowledge about the generalized inverse cdf that is hardly available in complex modeling.

2.3 General Transformation Methods

The inverse cdf method uses cdf of a distribution to generate a map from the uniform density to our target density π. It is natural to generalize the method by thinking about: i) Apart from cdf, can we choose other functions to establish a transform between uniform and π? ii) Rather than setting uniform density, can we set other standard distributions that are easy to sample from as the basis, and then transform it to π? In fact, general transformation methods are a large group of simulation methods mapping any easily sampled distribution to our target distribution π via any properly designed transformations.

2.3.1 Normal Generator

Box-Muller algorithm can be used to generate vectors of independent Gaussian random variables.

Example 2.3.1 (Box-Muller algorithm) *Let U_1, U_2 be independent and $U_1, U_2 \sim \mathcal{U}_{[0,1]}$, then*

$$X_1 = \sqrt{-2log(U_1)cos(2\pi U_2)} \sim N(0,1) \tag{2.5}$$

$$X_2 = \sqrt{-2log(U_1)sin(2\pi U_2)} \sim N(0,1) \tag{2.6}$$

and X_1, X_2 are independent.

The normality can be checked by histograms and density plots in Figure2.3. The dependency between X_1 and X_2 can be proved by comparing $Cor(X_1, X_2)$ with 0, as uncorrelated implies independent for normal variables.

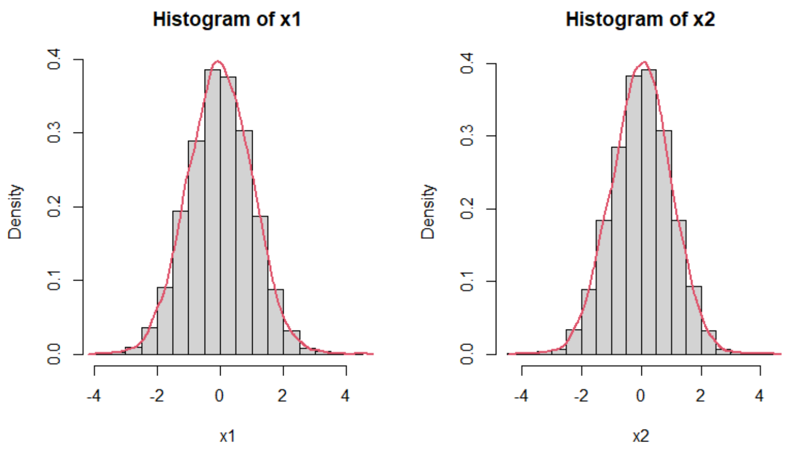


Figure 2.3: Histogram and density plots of generated normal random variables X_1, X_2 using Box-Muller algorithm.

For correlated multivariate normal distribution, we can apply coloring transformation to independent normal variables. Let $Z = (Z_1, \dots, Z_d)^T$ be a vector of d independent standard normal variables, then

$$X = \Sigma^T Z + \mu \sim N(\mu, \Sigma) \tag{2.7}$$

2.3.2 Gamma and Beta Simulation

As we discussed earlier, simple standard distributions like $Exp(1)$ can be simulated as easy as uniform distribution, we can take advantage of properties of standard probability distributions to simulate more complicated distributions.

Assume X_i are iid random variables with $X_i \sim Exp(1)$, then

$$Y = \frac{\sum_{i=1}^{\alpha} X_i}{\beta} \sim Gamma(\alpha, \beta) \tag{2.8}$$

$$Y = \frac{\sum_{i=1}^{\alpha} X_i}{\sum_{i=\alpha+1}^{\alpha+\beta} X_i} \sim Beta(\alpha, \beta) \tag{2.9}$$

Specifically, χ^2 distribution is a special case of Gamma distribution

$$Y = 2\sum_{i=1}^{n} X_i \sim \chi^2_{2n} \tag{2.10}$$

2.3.3 Composition Methods

The composition methods consider cases where a marginal distribution can be naturally represented as a mixture distribution, depending on the auxiliary space $\mathcal{Y}$ is continuous or discrete

$$\pi(x) = \int_{\mathcal{Y}} \pi(x|y)\pi(y)dy \tag{2.11}$$

or

$$\pi(x) = \sum_{i \in \mathcal{Y}} f_i(x) p_i \tag{2.12}$$

where $Y \sim \pi(y)$ and $X|y \sim \pi(x|y)$ are easy to sample. Then we can obtain sample X by first drawing Y from the mixing distribution $\pi(y)$ and then taking X from conditional distribution $\pi(x|y)$.

Example 2.3.2 (Negative binomial distribution) *(Robert et al. (2010)) For a negative binomial random variable $X \sim NB(r, p)$, it has a mixture presentation using*

$$X|y \sim Pois(y),\ Y \sim Gamma(r, \frac{1-p}{p}) \tag{2.13}$$

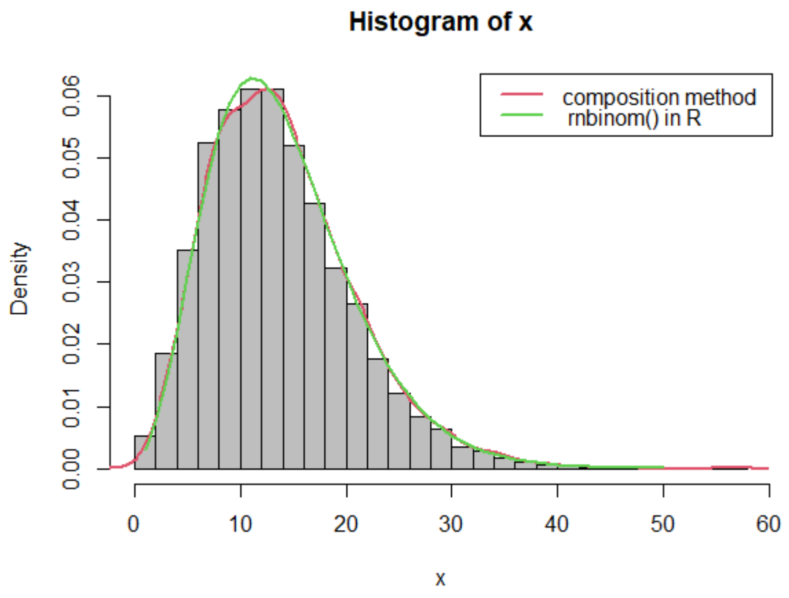


Figure 2.4: Histogram and density plot of generated variable $X \sim NB(6, 0.3)$ via composition method along with the true density plot.

In summary, general transformation methods simulate distributions relying on various relationships between probability distributions. Almost all these relations allow us to draw more complicatedly distributed samples from readily accessible distributions. Intricate links between standard probability distributions illustrate how widely available this method is among standard distributions. However, the enormous web of transformation methods is still way far from reaching all distributions, especially those non-standard distributions in real-life Bayesian analysis. Also, these methods are not computationally efficient, which can be verified by $system.time()$ in R, as it is inevitable to calculate transcendental functions such as log, cos and sin.

2.4 Accept-Reject Method

The failures of inverse cdf method and general transformation method are due to the idea of sampling directly from target distribution using its probabilistic properties. As there is no free lunch, the cost of directness is that it requires too much accurate knowledge about the target distribution, therefore can be difficult to generalize. Accept-reject method is a indirect but powerful sampling method, where we first draw from a easily sampled proposal distribution and then tailor the proposal in the shape of the target distribution π by accepting and rejecting.

For any target distribution π, find a proposal distribution q, satisfying:
i) π and q have compatible supports

$$q(x) > 0, \ \forall \ x \ s.t. \ \pi(x) > 0 \tag{2.14}$$

ii) π can be upper bounded by a multiple of q

$$\exists M < \infty, \ s.t. \ \pi(x) \le Mq(x) \tag{2.15}$$

Algorithm 1 Accept-reject algorithm

```
1. generate $Y \sim q$, $U \sim \mathcal{U}_{[0,1]}$
2. if $U \le \frac{\pi(Y)}{Mq(Y)}$
      accept $X \leftarrow Y$
   else
      reject, return to step 1
```

The distribution of accepted sample by accept-reject algorithm can be verified to be our target distribution π, as

$$\begin{aligned}
P(Y \le x | U \le \frac{\pi(Y)}{Mq(Y)}) &= \frac{P(Y \le x, U \le \frac{\pi(Y)}{Mq(Y)})}{P(U \le \frac{\pi(Y)}{Mq(Y)})} \\
&= \frac{\int_{-\infty}^{x} \int_0^{\pi(y)/(Mq(y))} duq(y)dy}{\int_{-\infty}^{\infty} \int_0^{\pi(y)/(Mq(y))} duq(y)dy} \\
&= \frac{\int_{-\infty}^{x} \frac{\pi(y)}{Mq(y)} q(y)dy}{\int_{-\infty}^{\infty} \frac{\pi(y)}{Mq(y)} q(y)dy} \\
&= \frac{\int_{-\infty}^{x} \pi(y)dy}{\int_{-\infty}^{\infty} \pi(y)dy} \\
&= P(X \le x)
\end{aligned} \tag{2.16}$$

Noticing that M cancels in equations 2.16, we do not need to worry about the exact value of normalizing constant, it is sufficient to know there exists such a constant. In fact, this happens to be just useful enough in practical Bayesian analysis, where we usually only know π and q up to a constant due to posterior intractability. i.e. $\pi = \tilde{\pi}/c_\pi$ $q = \tilde{q}/c_q$ with known $\tilde{\pi}, \tilde{q}$, then the accept probability remains the same point-wisely with constants c_π, c_q absorbed into M

$$\frac{\pi(x)}{q(x)} \le M \iff \frac{\tilde{\pi}(x)}{\tilde{q}(x)} \le \frac{c_\pi}{c_q} M = \tilde{M} \tag{2.17}$$

Unlike inverse cdf method and transformation methods, accept-reject method can be applicable in any dimension as long as the supports of target distribution and proposal are compatible and π/q is known up to a constant.

It can be proved that with an upper bound M on the density ratio $\frac{\pi}{q}$, the probability of acceptance is $1/M$. This conclusion can be used reversely to calculate the missing constant by expected acceptance rate

$$M = E[I(U < \frac{\tilde{\pi}}{\tilde{M}\tilde{q}})] \tag{2.18}$$

Therefore, although the choice of M only need to satisfy $M > \sup \frac{\pi(x)}{q(x)}$, it is advisable to choose M to be as small as possible to get a higher acceptance probability. In other words, we should proposal to be as close to target distribution as possible.

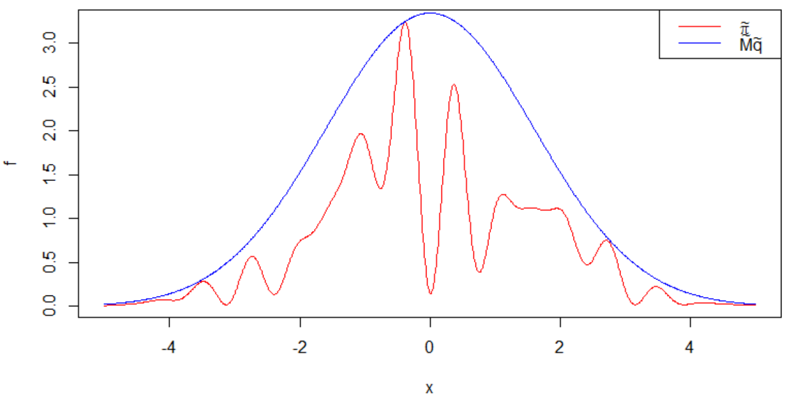


Figure 2.5: Demonstration of non-standard target distribution and proposal. The red curve is the target distribution that we know up to a constant $\tilde{\pi}$, the blue curve denotes the scaled proposal $\tilde{M}\tilde{q}$.

Example 2.4.1 (Simulate $X \sim Beta(2.7, 6.3)$ via accept-reject method) *To simulate beta density, we can start by using uniform proposal to ensure the compatibility of supports. i.e. $\tilde{q}(x) = q(x) = I_{(0,1)}(x)$, then choose M to be the maximum of beta density, s.t. $\pi(x) \leq Mq(x)$.*

For this example, uniform proposal is clearly not a good proposal as the shape of uniform distribution is nothing like the target beta distribution, thus the accept probability is very small. A good alternative proposal can come from a more non-informative distribution that is close to $Beta(2.7, 6.3)$, for example $Beta(2, 6)$, i.e. $\tilde{q}(x) = Beta(2, 6)$. In general, for target $Beta(\alpha, \beta)$, there is a proper constant M if using $Beta(\alpha', \beta')$ as proposal with $\alpha' < \alpha$ and $\beta' < \beta$. This can be interpreted as selecting a proposal that is less informative and has larger tails than target distribution. Figure 2.6 visualizes and compares the accept-reject process using uniform proposal and less informative beta proposal. It is clear from the figure that the Beta proposal is more accurate and efficient than the uniform proposal.

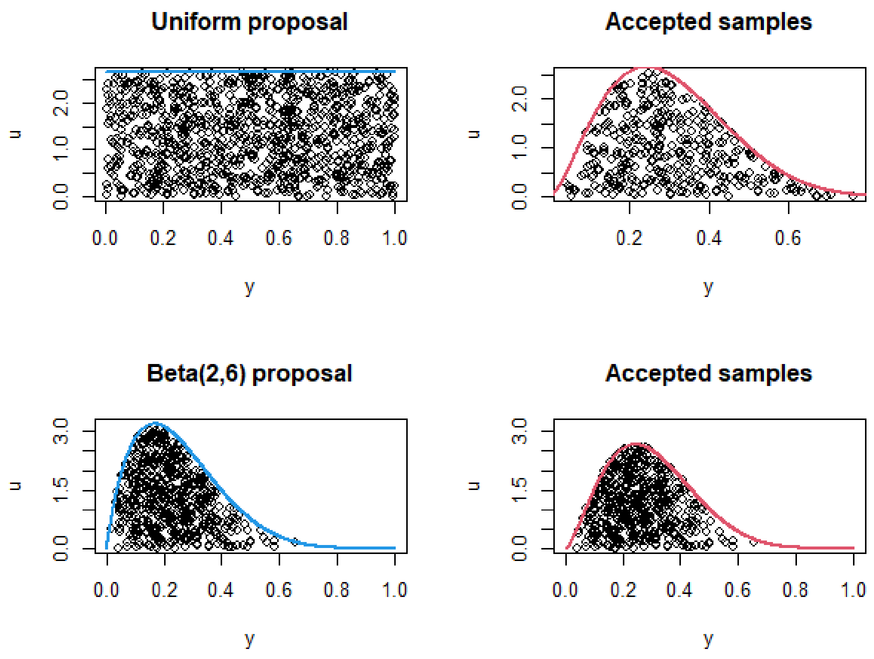


Figure 2.6: Generation of $X \sim Beta(2.7, 6.3)$ via accept-reject algorithm using uniform proposal and $Beta(2,6)$ proposal. Circles denote simulated $(Y, Mq(Y))$, the blue line denotes pdf of $Mq(Y)$ and the red curve denotes exact pdf of π - $Beta(2.7, 6.3)$.

Though the accept-reject method generally works well for target distributions even we know them up to a constant, finding a proper proposal q and small enough constant M is still not easy, especially in high-dimensional cases. Moreover, for multidimensional parameter spaces, it is common that only conditional distributions are available, such as $\pi(x_1|x_2)$ and $\pi(x_2|x_1)$, but it is hard to derive joint distribution $\pi(x_1, x_2)$ for accept-reject sampling.

2.5 Importance Sampling

Importance sampling is another indirect sampling method for estimating properties of the target distribution π by correcting a readily available distribution. The general idea of importance sampling is reweighting.

2.5.1 Standard Importance Sampling

Recall the standard Monte Carlo method, the integration is an expectation of π-distributed variables weighted by $f(x)$

$$\int_{\mathcal{X}} f(x)\pi(x)dx = E_\pi[f(X)] \tag{2.19}$$

Since sampling from π is hard, we can alternatively sample from a proposal q that is relatively easy to simulate by adding an additional importance weight.

Let q be a distribution that satisfies $q(x) > 0$ whenever $f(x)\pi(x) > 0$, then

$$E_\pi[f(X)] = \int_{\mathcal{X}} \frac{f(x)\pi(x)}{q(x)} q(x)dx = E_q[\frac{f(X)\pi(X)}{q(X)}] = E_q[f(X)w(X)] \quad (2.20)$$

where $w(x) = \pi(x)/q(x)$ is called importance weight.

Assume $X_1, X_2, ..., X_n$ are iid random variables following proposal distribution q. Then the importance sampling estimator

$$\hat{I}^{IS}(n) = \frac{1}{n}\sum_{i=1}^{n} f(X_i)w(X_i) \quad (2.21)$$

If $E_q[|f(X)|w(X)] < \infty$, the importance sampling estimator $\hat{I}^{IS}(n)$ is unbiased and consistent by the Law of Large Numbers. Moreover, if

$$\sigma^2_{IS}(f) = Var_q(f(X)w(X)) < \infty \quad (2.22)$$

the variance of estimator $Var_q[\hat{I}^{IS}(n)] = \frac{1}{n}Var_q(f(X)w(X))$, by Central Limit Theorem

$$\sqrt{n}(\hat{I}^{IS}(n)) \xrightarrow{D} N(0, \sigma^2_{IS}(f)) \quad (2.23)$$

Example 2.5.1 *Suppose random variable $X \sim N(0,1)$ and we are interested in the tail probability $P(X > 4.5)$. By standard Monte Carlo integration, we sample x_i from standard normal density, assign $f(x_i) = 1$ when $x_i > 4.5$ and $f(x_i) = 0$ otherwise. Since the probability at tails of normal is small, we produce only one hit in nearly 3 million draws. Alternatively, we can add a large importance weight on our target area by using proposal $q(x) = e^{-x+4.5}$ - the pdf of exponential distribution $Exp(1)$ truncated at 4.5. Hence, the importance sampling estimator*

$$\frac{1}{n}\sum_{i=1}^{n}\frac{\pi(X_i)}{q(X_i)} = \frac{1}{\sqrt{2\pi}n}\sum_{i=1}^{n} e^{-\frac{X_i^2}{2}+X_i-4.5} \quad (2.24)$$

where X_i are simulated from q. A comparison between importance sampling and naive Monte Carlo is shown in Figure 2.7.

The variance of our importance sampling estimator varies with the choice of proposal or importance weight, as we discussed earlier, the Central Limit Theorem justifies

$$\sqrt{n}(\hat{I}^{IS}(n)) \xrightarrow{D} N(0, \sigma^2_{IS}(f)) \quad (2.25)$$

It is natural to think of optimizing q in the sense of minimizing $Var_q[\hat{I}^{IS}(n)]$, or equivalently minimizing $\sigma^2_{IS}(f)$. In fact, given f and π, by definition,

$$\begin{aligned}\sigma^2_{IS}(f) &= Var_q[f(X)w(X)] \\ &= E_q[f^2(x)w^2(x)] - (E_q[f(x)w(x)])^2 \\ &= E_q[f^2(x)w^2(x)] - I^2\end{aligned} \quad (2.26)$$

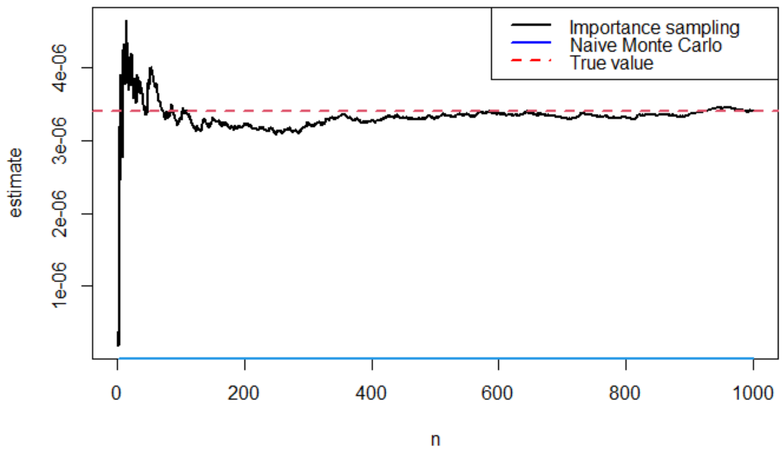


Figure 2.7: Comparison between importance sampling and naive Monte Carlo. For rare events, like sampling from a tail of normal distribution, naive Monte Carlo does not work as the effective sample size is nearly 0 and the estimated standard error is large. Whereas, importance sampling is applicable as long as the proposal or importance weight is chosen properly.

where I is the true value value of target integral. Thus, it is sufficient to minimize $E_q[f^2(x)w^2(x)]$. For optimal proposal q^*

$$\begin{aligned} E_{q^*}[f^2(X)w^2(X)] &= \int_{\mathcal{X}} \frac{f(x)\pi(x)}{|f(x)|\pi(x)} dx \int_{\mathcal{X}} |f(x)|\pi(x)dx \\ &= (\int_{\mathcal{X}} |f(x)|\pi(x)dx)^2 \end{aligned} \tag{2.27}$$

And the minimum is justified by Jensen's inequality

$$E_{q^*}[f^2(X)w^2(X)] \geq (E_{q^*}[f(X)w(X)])^2 = (\int_{\mathcal{X}} |f(x)|\pi(x)dx)^2 \tag{2.28}$$

Therefore, the optimal proposal

$$q^*(x) = \frac{|f(x)|\pi(x)}{\int_{\mathcal{X}} |f(x)|\pi(x)dx} \tag{2.29}$$

and the corresponding variance of importance sampling estimator is 0.

Unfortunately, as the derivation of the optima assumes knowing the true value of the integral which is actually our unknown target, this lower bound of variance can never be attained and this optimal proposal q^* can only be used as a guidance - the chosen proposal is expected to be close to q^* in order to obtain an accurate estimator.

In practice, since modern computers are capable of intensive calculation, optimizing proposal is recommended but not a must. What should be really careful is to make sure the variance is finite. From such perspective, a rough recommendation is to choose $q(x)$ such that $|f(x)|w(x)$ is almost constant or at least enjoys controlled tail behaviour, in this setting the importance sampling estimator is more likely to have finite variance.

2.5.2 Normalized Importance Sampling

Similar to the accept-reject method, it is common in Bayesian analysis that we do not have access to exact target distribution π and proposal q, instead, we only know them up to a constant: $\tilde{\pi} = c_\pi \pi$ and $\tilde{q} = c_q q$. In this case, the importance weight can not be calculated as before.

Alternatively, we define unnormalized importance weight

$$\tilde{w}(x) = \frac{\tilde{\pi}(x)}{\tilde{q}(x)} \tag{2.30}$$

as they do not sum to 1. Since $\tilde{w}(x) = \frac{c_\pi}{c_q} w(x)$, the expectation can be rewritten as

$$\begin{aligned} E_\pi[f(X)] &= \int_{\mathcal{X}} f(x)\pi(x)dx \\ &= \frac{\int_{\mathcal{X}} f(x)w(x)q(x)dx}{\int_{\mathcal{X}} w(x)q(x)dx} \\ &= \frac{\int_{\mathcal{X}} f(x)\tilde{w}(x)\tilde{q}(x)dx}{\int_{\mathcal{X}} \tilde{w}(x)\tilde{q}(x)dx} \\ &= \frac{E_{\tilde{q}}[f(X)\tilde{w}(X)]}{E_{\tilde{q}}[\tilde{w}(X)]} \end{aligned} \tag{2.31}$$

Thus, given iid random variables $X_1, X_2, \ldots, X_n$ following $\tilde{q}$, the normalized importance sampling estimator

$$\hat{I}^{NIS}(n) = \frac{\sum_{i=1}^n f(X_i)\tilde{w}(X_i)}{\sum_{i=1}^n \tilde{w}(X_i)} \tag{2.32}$$

It can be proved that the normalized importance sampling estimator introduces bias, but it is still consistent by the law of large numbers (bias decreases with rate $O(1/n)$) (Rebeschini (2018)). As will be discussed in the following chapter, the Markov Chain Monte Carlo method enables us to generative as many samples as we can given enough time for computation, bias of the estimator can be controlled.

Chapter 3

Markov Chain Monte Carlo

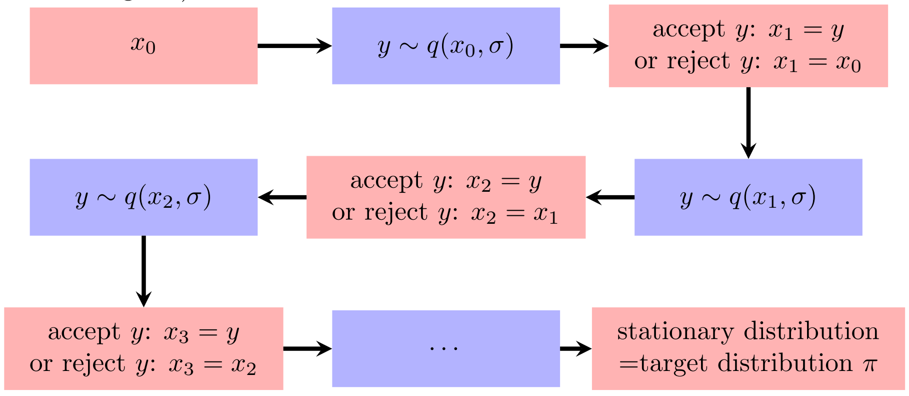
In the last chapter, various classical simulation methods for a target distribution π are discussed for the purpose of sampling-based Monte Carlo. Unfortunately, these methods can not generalize well for high dimensional scenarios, which is often the case for Bayesian analysis: It is almost impossible to obtain cdf from a pdf with unknown normalizing constant, so inverse cdf method fails; General transformation methods only work for a particular family of standard distribution and are computationally inefficient; Accept-reject method requires less information about target distribution and can simulate any distribution, but it is hard to find appropriate proposal Mq in high dimensional spaces and it does not work when we only have conditional distributions for some parameters instead of their joint distribution; Importance sampling reforms the Monte Carlo integration by turning to sample from an easily simulated proposal distribution q, however, it is still difficult to make q easy to sample and provide good approximations at the same time. For the above reason, more sophisticated Markov Chain Monte Carlo (MCMC) is adopted to solve this problem.

In brief, Markov chain Monte Carlo is a technique that allows sampling parameter values from posterior distribution using Markov chain, and then use empirical mean to estimate summary statistics (such as posterior mean, posterior modes and credible interval) by Monte Carlo integration. MCMC is powerful as it works well even we only known posterior up to a constant and tedious sampling process can be implemented by computer simulation. In the past 30 years, a series of algorithms based on MCMC have been rapidly developed and they have become the most common class of algorithms in Bayesian analysis.

3.1 Independency-Efficiency Tradeoff

This section aims to provide a general answer for what actually makes Markov chain Monte Carlo is stand out among other methods. To understand this, we shall recontemplate why the accept-reject method can be inefficient. In fact, the main drawback of our original accept-reject method is that the samples are drawn independently from the proposal distribution. This golden rule of independence caters the need of Monte Carlo method, as only the empirical average of functions of independent variables can guarantee a good approximation to the true function. However, the side effect here is that independent sampling always erase the information from previous samples when acquiring a new sample.

An alternative strategy is to retain some previous information and allow it to affect the next sampling. As shown in Figure 3.1, suppose we already obtained an accepted sample by accept-reject method which lies in the high density region of our target distribution, then it is natural to believe any new sample close to this existed sample may also enjoy a high accept rate. Therefore, instead of sampling from the previous proposal and suffering high reject probability, it is wise to replace it with a proposal adjusted by existed samples. This new simulation strategy can easily adopt Markov chain theory and generates dependent samples in order to allow the proposal to move around the support of target distribution. Then our accept-reject method becomes, for instance, first sampling y from a proposal based on initial point $q(x_0, \sigma)$, if y satisfies certain condition, accept $x_1 = y$, otherwise $x_1 = x_0$, then proceed by sampling y from $q(x_1, 0)$, if y satisfies certain condition, accept $x_2 = y$, otherwise $x_2 = x_1$, etc. In this way, a realization of Markov chain $\{x_0, x_1, x_2, \dots\}$ is obtained. More importantly, as will be discussed in the following section, a great property of Markov chain is that under certain conditions the chain can converge to a stationary distribution. If we can construct a good enough Markov chain in a way that the limiting distribution/stationary distribution is exactly the target posterior distribution, then obtaining samples from Markov chain reached stationary distribution will be equivalent to sampling from the target posterior distribution (shown in diagram).


x_0
$y \sim q(x_0, \sigma)$
accept y: $x_1 = y$ or reject y: $x_1 = x_0$
$y \sim q(x_1, \sigma)$
accept y: $x_2 = y$ or reject y: $x_2 = x_1$
$y \sim q(x_2, \sigma)$
accept y: $x_3 = y$ or reject y: $x_3 = x_2$
$\dots$
stationary distribution =target distribution π

Since, the sampling efficiency is improved, at the same time, we wish to minimize the potential problem of brought by dependence. Ideally, the dependence should quickly diminish to an insignificant level. In practice, it is also viable to use sub-sampling or thinning to ensure the correlations among those samples for Monte Carlo are not significant.

In summary, the dilemma of high dimensional sampling is the conflict in simultaneously controlling the error due to dependence (bias) and the efficiency of sampling algorithms (variance). Markov chain Monte Carlo demonstrates a good balance between independency and efficiency by allowing a dynamic adaptive proposal using Markov chain, where small correlations between samples are brought in, but the efficiency of algorithms is improved significantly. More generally speaking, this tradeoff can be viewed similarly as the bias-variance tradeoff in machine learning theory, where increasing a little bias results in better controlled variance and efficiency. Analogous strategies are widely utilised in high dimensional statistics.

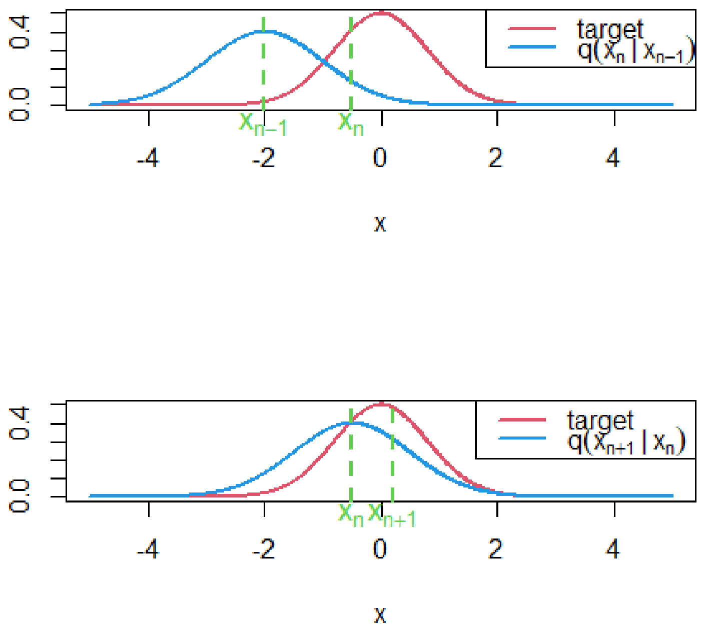


Figure 3.1: The proposal for new sample x_{n+1} is dependent on the previously accepted sample x_n. So if x_n is sampled from a high density area of the target distribution, when sampling x_{n+1}, it is rational to allocate high probabilities for those close to it to be sampled.

3.2 A Mini-refresher on Markov Chain

Since MCMC is a method by combining Markov chain and Monte Carlo, it is necessary to provide a small refresher on theory about Markov chains.

Definition 3.2.1 (Discrete time Markov chain) *Markov chain X_n, $n \geq 0$ is a stochastic process satisfying the Markov property: for any $n \geq 0$, any $X_0, X_1, X_2, \dots \in S$*

$$P(X_{n+1} = x_{n+1}|X_n = x_n) = P(X_{n+1} = x_{n+1}|X_n = x_n, \dots, X_0 = x_0) \quad (3.1)$$

where S is state space - the set of all possible states the stochastic process can be.

In words, a Markov chain have lack of memory property. If we take time n as present and take everything before time n as past, then the future of Markov chains X_n completely forget about the past and only depend on the present.

Definition 3.2.2 (Homogeneous Markov chain) *A Markov chain is homogeneous if for any $n \geq 0$, any $i, j \in S$*

$$P(X_{n+1} = j|X_n = i) = P(X_1 = j|X_0 = i) \quad (3.2)$$

From now on, for the purpose of MCMC, we completely focus on homogeneous Markov chains.

Definition 3.2.3 (Transition probability and matrix) *The transition probability from i to j is defined as*

$$p_{i,j} = P(X_1 = j | X_0 = i) \tag{3.3}$$

and the transition matrix (transition kernel) is $P = (p_{i,j})$. Suppose the Markov chain is homogeneous, then for any $m \geq 0$ and $n \geq 1$, the n-step transition probability

$$p_{i,j}(n) = P(X_{m+n} = j | X_m = i) \tag{3.4}$$

and n-step transition matrix

$$P_n = (p_{i,j}(n)) \tag{3.5}$$

Theorem 3.2.4 (Chapman-Kolmogorov equations) *For any $i, j \in S$, any $m, n \geq 1$*

$$p_{i,j}(m+n) = \sum_{k \in S} p_{i,k}(m) p_{k,j}(n) \tag{3.6}$$

In matrix notation, $P_{m+n} = P_m P_n$, hence $P_n = P^n$.

Theorem 3.2.5 *Let row vector $\mu_n = [\mu_n^{(1)}, \mu_n^{(2)}, \dots]$ denote the distribution at time n, where $\mu_n^{(i)} = P(X_n = i)$. Then, $\mu_{m+n} = \mu_m P^n$ and hence $\mu_n = \mu_0 P^n$, where μ_0 is the stationary distribution of the Markov chain.*

This theorem characterizes the Markov chain is determined by initial states u_0 and transition matrix P. In fact, this is essentially what we do in MCMC algorithms, where we only set initial parameters and the transition probabilities to start iterations. Just as fixed point is important for iterating a function, the distribution that stabilizes a Markov chain is of great significance for Markov chain.

Definition 3.2.6 (Stationary distribution) *A row vector π is a stationary distribution for a Markov chain with transition matrix P, if: (i) $\pi_i \geq 0$ for any $i \in S$; (ii) $\sum_{i \in S} \pi_i = 1$; (iii) $\pi P = \pi$.*

Apart from above basics, there are some properties of Markov chains worth reviewing in order to secure the existence of stationary distributions.

Definition 3.2.7 (Irreduciblity) *A Markov chain is irreducible if for any state pair $i, j \in S$*

$$\exists n \geq 0, \text{ s.t. } p_{i,j}(n) > 0 \tag{3.7}$$

which indicates there is a path of positive probability between any two states. A markov chain is π-irreducible, if the above holds for any state pair in support of distribution π.

This property enables the Markov chain to explore the whole state space from any initial point, rather than limited in a subset. Thus, in Bayesian analysis, π-irreducibility ensures the possibility of sampling from the whole support of target distribution π.

Definition 3.2.8 (Aperiodicity) *Define the period of state i to be the greatest common divisor of epochs where return is possible*

$$d(i) = gcd\{n \geq 1 | p_{i,i}(n) > 0\} \tag{3.8}$$

A Markov chain is aperiodic if $d(i) = 1$ for any state $i \in S$. i.e. there is no sign of periodicity is an aperiodic Markov chain such that the limiting distribution can converge.

Definition 3.2.9 (Recurrence and transience) *State i is recurrent, if*

$$P(X_n = i \ for \ some \ n \geq 1 | X_0 = i) = 1 \tag{3.9}$$

otherwise, it is transient. A Markov chain is recurrent if all states are recurrent. i.e. for each states in a recurrent Markov chain, the probability of returning to the initial state is 1, or equivalently, the expected number of returns in infinite.

Definition 3.2.10 (Positive recurrence and null recurrence) *For any recurrent state $i \in S$, define the recurrence time T_i to be the time of the first return to i, starting from i. Define mean recurrence time $\mu_i = E[T_i]$. Then, state i is positive recurrent, if $\mu_i < \infty$; State i is null recurrent, if $\mu_i = \infty$. A Markov chain is positive recurrent, if all states in state space are positive recurrent.*

The positive recurrence of a Markov chain guarantees that if the limiting distribution exists, then it is not 0. In this case, the limiting distribution converges to the stationary distribution.

Theorem 3.2.11 *(Bagley (2022)) If a Markov chain with countable state space S and transition matrix P is irreducible, aperiodic and positive recurrent, then there exists a unique stationary distribution π and it is equal to the limiting distribution*

$$\lim_{n \to \infty} p_{i,j}(n) = \pi_j, \ for \ any \ i, j \in S \tag{3.10}$$

This theorem illustrates that a 'good' Markov chain converges to a stationary distribution that is irrelevant to its initial state. In this sense, the initial state of Markov chain can be inconsequential if the Markov chain is run sufficiently long. However, it is impossible to wait infinite time for convergence in practice, so it is of great importance to check whether the Markov chain is efficient enough to allow a quick convergence or whether the running time n is large enough to reach an approximated stationary distribution. The process of starting iterations for a Markov chain to converge is called 'burn-in' of the chain, where the samples do not follow the stationary distribution and are discarded. So the length of burn-in can be a measure for the rate of convergence of the chain. These features are usually verified by diagnostic checking of Markov chain Monte Carlo.

On the other hand, the stationary distribution of an irreducible, aperiodic and positive recurrent Markov chain depends only on the transition kernel. Thus, it is sufficient to find such good enough transition kernel such that the stationary distribution can be approached quickly and is equal to the target distribution. In theory, since stationary distribution π can be regarded as a left eigenvalue of transition kernel P, by the Perron-Frobenius in spectral theory, the rate of convergence depends on the second large eigenvalue, the smaller the better (Andrieu et al. (2003)). In practice, thinking of how to construct such a quickly convergent transition kernel with a stationary distribution is a great concern, different ways of constructing transition kernel leads to different MCMC algorithms.

Finally, it is vital to warn that the above is only a scratch of some essentials of discrete state space Markov chain for appreciating how Markov chains work. The discrete state space only consider countable many states. However, the true MCMC methods usually draw samples from a continuous region and hence adopt continuous state space Markov chains. For continuous state spaces, probability of being in any single state is zero, so the transition probability defined above fails and measure theory is brought in for redefinition. To avoid deviating too much from MCMC, the report will not go any further into Markov chain theory, as this non-rigorousness has limited effect on the following discussions.

3.3 Metropolis-Hastings Algorithm

As the transition kernel of a Markov chain is the key objective for designing a MCMC algorithm, this section aims to provide a solution to the generic problem: how to find a transition kernel P for an arbitrary target stationary distribution π. And hence derive the Metropolis-Hastings algorithm.

3.3.1 Detailed Balance Condition

Given a stationary distribution π, it is generally difficult to find corresponding transition kernel P. So one may start by considering which kind of transition kernel can easily relate to a stationary distribution. A straight forward answer is those transition kernel P that satisfy the detailed balance condition.

Definition 3.3.1 (Detailed balanced condition) *An aperiodic transition kernel P satisfies the detailed balance condition for distribution π if for all $i, j \in S$*

$$\pi(i)P(i,j) = \pi(j)P(j,i) \tag{3.11}$$

where $\pi(i) = \pi_i$ and $P(i,j) = P_{i,j}$, the notation is changed for convention.

If we take the summation with respect to i on both sides of the equation

$$\sum_{i\in S} \pi(i)P(i,j) = \sum_{i\in S} \pi(j)P(j,i) = \pi(j) \sum_{i\in S} P(j,i) = \pi(j) \tag{3.12}$$

in matrix notation $\pi P = \pi$. i.e. if P satisfies detailed balance condition for distribution π, then π is the stationary distribution of aperiodic Markov chain with transition kernel P.

To briefly interpret, when a Markov chain reaches its stationary distribution, for any i, j, the probability of being in state i and transit to state j is equal to the probability of being in state j and transit to state i. So if π is the stationary distribution, any transition kernel P satisfies detailed balance condition is acceptable for constructing a Markov chain.

3.3.2 Transition Kernel from Rejection

Unfortunately, it is still too difficult to find P for stationary distribution π such that detailed balanced condition is satisfied. Moreover, not all acceptable transition kernel P satisfies detailed balance condition. In fact, for two dimensional sample space $\mathcal{X}$, all viable transition kernel satisfy detailed balance condition, but for $dim(\mathcal{X}) \geq 3$, many more viable P do not satisfy detailed balance condition. Therefore, it is unwise to limit our attention on transition kernels satisfying detailed balance condition for π.

Consider general stationary distribution π and arbitrary transition kernel Q, such that Q has compatible state space with π

$$\pi(i)Q(i,j) \neq \pi(j)Q(j,i) \tag{3.13}$$

there a way to equate two side by introducing $\alpha(i,j)$ and $\alpha(j,i)$ such that

$$\pi(i)Q(i,j)\alpha(i,j) = \pi(j)Q(j,i)\alpha(j,i) \tag{3.14}$$

where $\alpha(i,j) = \pi(j)Q(j,i)$ and $\alpha(j,i) = \pi(i)Q(i,j)$, which is defined for any stationary distribution π and arbitrary transition kernel Q. Now, we define a new kernel P such that

$$P(i,j) = Q(i,j)\alpha(i,j) \tag{3.15}$$

then the new transition kernel P satisfies detailed balance condition for π

$$\pi(i)P(i,j) = \pi(j)P(j,i) \tag{3.16}$$

By definition, $\alpha(i,j) \in (0,1)$ for any i and j is called acceptance probability. And each side of the equation 3.14 can be decomposed and interpreted: $\pi(i)Q(i,j)$ is the probability of choosing a new state from a proposal based on an old state, and $\alpha(i,j)$ judges whether or not we should transit to the new state. In this way, a transition kernel P can obtained by accepting-rejecting a relatively easily derived transition kernel Q with probability $\alpha(i,j)$ for each i, j.

Algorithm 2 Naive Markov chain accept-reject sampling

```
1. determine the number of simulation $N$ and an arbitrary initial point $x_0$
2. for $i = 0$ to $N$
       generate $U \sim \mathcal{U}_{[0,1]}$, $Y \sim q(Y|X_i)$
       if $U < \alpha(X_i, Y) = \pi(Y)q(X_i|Y)$
           $X_{i+1} = Y$
       else
           $X_{i+1} = X_i$
```

One obvious drawback of this Markov chain based rejection sampling is that the acceptance probability is too small, usually about 0.1, so that most proposed samples are refused. Thus, the Markov chain transits very slowly and takes too much time for converging to the stationary distribution.

3.3.3 Metropolis-Hastings Algorithm

In the equation 3.14, $\alpha(i,j), \alpha(j,i)$ are introduced as multipliers to fulfill detailed balance condition, or equivalently

$$\frac{\alpha(i,j)}{\alpha(j,i)} = \frac{\pi(j)Q(j,i)}{\pi(i)Q(i,j)} \tag{3.17}$$

It is obvious that if we multiply both $\alpha(i,j)$ and $\alpha(j,i)$ by a factor at the same time the detailed balance condition also holds. Since we wish these two acceptance probabilities to be as large as possible, without loss of generality, assume $\alpha(i,j) < \alpha(j,i)$, it is optimal to scale these two probabilities with the same factor such that $\alpha(j,i) = 1$, so that both acceptance probabilities are maximized. In this case

$$\alpha(i,j) = \frac{\pi(j)Q(j,i)}{\pi(i)Q(i,j)}\alpha(j,i) = \frac{\pi(j)Q(j,i)}{\pi(i)Q(i,j)} \tag{3.18}$$

To be concise for any state i, j

$$\alpha(i,j) = min\{\frac{\pi(j)Q(j,i)}{\pi(i)Q(i,j)}, 1\} \tag{3.19}$$

This expression is known as the acceptance probabilities of Metropolis-Hastings algorithm which is arguably the most useful MCMC algorithm based on the naive Markov chain accept-reject sampling, while the low accept rate problem is improved.

Algorithm 3 Metropolis-Hastings algorithm

```
1. determine the number of simulation $N$ and an arbitrary initial point $x_0$
2. for $i = 0$ to $N$
       generate $U \sim \mathcal{U}_{[0,1]}$, $Y \sim q(Y|X_i)$
       compute $\alpha(Y|X_i) = min\{\frac{\pi(Y)q(X_i|Y)}{\pi(X_i)q(Y|X_i)}, 1\}$
       if $U < \alpha(Y|X_i)$
           $X_{i+1} = Y$
       else
           $X_{i+1} = X_i$
```

The probability of accepting a candidate from proposal $q(Y|X_i)$ at state i is

$$a(x_i) = \int_{\mathcal{X}} \alpha(y|x_i)q(y|x_i)dy \tag{3.20}$$

This is to be distinguished from acceptance rate, which is the mean of acceptance probabilities

$$\bar{\alpha} = \lim_{N\to\infty} \frac{1}{N}\sum_{n=0}^{N} \alpha(Y|X_n) = \int \alpha(y|x)\pi(x)q(y|x)dydx \tag{3.21}$$

A great benefit of Metropolis-Hastings algorithm is that it only requires we know target distribution up to a constant, as the fractional form of rejecting criterion allows constants to cancel out. This is necessary for Bayesian analysis where we are interested in sampling from a posterior distribution based on likelihood and prior.

Moreover, the following theorem (Tierney (1994)) guarantees the validity of Metropolis-Hastings algorithm for arbitrary initial point in most cases of Bayesian analysis.

Theorem 3.3.2 *If the Markov chain in Metropolis-Hastings algorithm is π-irreducible, then for any function f and any initial value X_0*

$$\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} f(X_i) = \int_{\mathcal{X}} f(x)\pi(x)dx \tag{3.22}$$

where $\pi(x)$ is the target posterior distribution in Metropolis-Hastings algorithm.

In fact, the Metropolis-Hastings algorithm is a generalization of Metropolis algorithm (Hastings (1970)). In 1953, the first version of the algorithm was designed by Metropolis et al., where they assumed proposals always are symmetric $q(x|y) = q(y|x)$ for any $x, y \in \mathcal{X}$ (Metropolis et al. (1953)). In this assumption, the proposal densities in denominator and numerator cancel out

$$\alpha(Y|X_i) = min\{\frac{\pi(Y)}{\pi(X_i)}, 1\} \tag{3.23}$$

Though Metropolis algorithm was very popular in wide applications in physics, the constraint of only selecting symmetric proposals stumbled its development in Bayesian analysis. Later through works of Hastings et al. in 1970, the improved version Metropolis-Hastings algorithm eliminates this requirement for proposal and now allows proposals of various form.

The choice of proposal q is vital for Metropolis-Hastings algorithm, even though there is no explicit requirement for q by construction. Analogous to the accept-reject method, it is advisable to apply proposal that that $M = \pi/q$ is bounded. Otherwise, the performance of Metropolis-Hastings algorithm can be very poor. A typical example of generating Cauchy random variables is given in Robert and Casella (2004).

3.3.4 Independent Metropolis-Hastings

Suppose the proposal distribution q satisfies $q(Y|X_i) = q(Y)$, i.e. once the proposal is chosen, the distribution of drawing a new candidate is independent of the previous accepted sample. Therefore the acceptance probability

$$\alpha(Y|X_i) = min\{\frac{\pi(Y)q(X_i)}{\pi(X_i)q(Y)}, 1\} \tag{3.24}$$

The independence here refers to the proposal q is fixed and does not move around with X_i. But it is still a sampling procedure based on Markov chain. Compared with the previous accept-reject sampling, independent Metropolis-Hastings never worries about choosing constant M, such that $\pi \le Mq$.

An intuitive interpretation for this acceptance probability is to view in an importance sampling perspective (Tierney (1994)). Define importance weight function $w = \pi/q$, then we can rewrite the accept rate as

$$\alpha(Y|X_i) = min\{\frac{w(Y)}{w(X_i)}, 1\} \tag{3.25}$$

It means that given existed sample X_i and proposed candidate Y, we can calculate their importance weights. If $w(X_i) \leq w(Y)$, then Y is more likely to be in a high density region of the target π where proposal q fails to discover. Since we already accepted X_i, it is reasonable to accept this more 'important' candidate Y with probability 1, otherwise this high density region that q missed can be hardly sampled. If $w(X_i) > w(Y)$, then for those candidates that are relatively more important', we assign them with relatively large acceptance probabilities.

Example 3.3.1 (Simulating $Beta(2.7, 6.3)$) *Now we implement independent Metropolis-Hastings algorithm for sampling from a $Beta(2.7, 6.3)$ distribution using $Beta(2, 6)$ as proposal, the same example we used in accept-reject sampling.*

Figure 3.2 shows a good simulation of the target distribution. A slice of trace plot in Figure 3.3 demonstrates the Markov chain performs stationary after iterations. The horizontal lines indicate no candidate is accepted for some iterations. The acceptance rate is 0.80, meaning that a large proportion of candidates are accepted. This is mainly due to the proposal being very close to the target distribution.

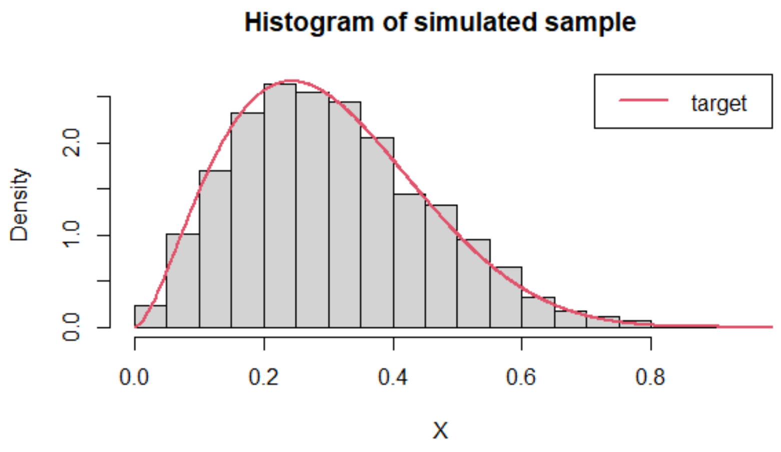


Figure 3.2: The histogram of simulated $Beta(2.7, 6.3)$ via independent sampler, the red curve is the true target density.

3.3.5 Random Walk Metropolis-Hastings

Random walk Metropolis-Hastings algorithm is a special case using random walk $Y = X_n + W$ where W follows a symmetric distribution g. i.e. $g(-x) = g(x)$ for any x. In this scenario, we have $q(Y|X_n) = q(X_n|Y)$, thus the acceptance probabilities

$$\alpha(Y|X_n) = min\{\frac{\pi(Y)}{\pi(X_n)}, 1\} \tag{3.26}$$

Example 3.3.2 (Simulating $Beta(2.7, 6.3)$) *Now, we use random walk Metropolis-Hastings algorithm to simulate the same toy example $Beta(2.7, 6.3)$ by choosing symmetric function g to be $\mathcal{U}_{[-0.3,0.3]}$.*

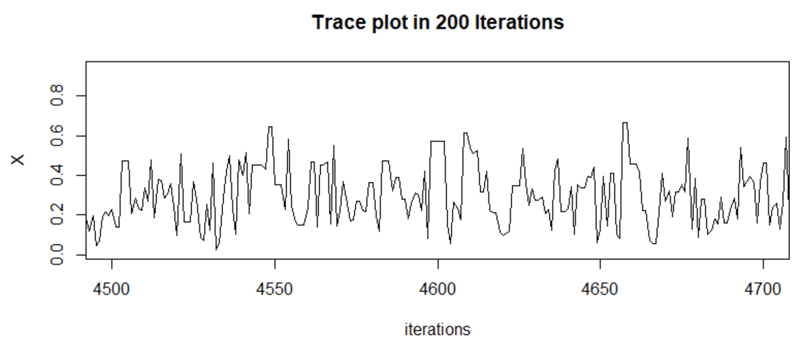


Figure 3.3: Partial magnification of trace plot using independent sampler.

Similar to the independent sampler example, the simulated distribution is very close to the target distribution in Figure 3.4 and Markov chain performs stationary in Figure 3.5. The acceptance rate of random walk Metropolis is 0.54, which is a good number for algorithms with random walk proposal. In fact, Roberts et al. (1997) advocate that an ideal proposal for random walk Metropolis-Hastings should be around 0.25 for high dimensional models ($d \geq 3$) and around 0.5 for one or two dimensional models. This is to avoid the acceptance rate being too large or too small. A large accept rate sometimes (not always) means that the movement on the support is limited, for example, being stuck in one tail. Hence the whole target distribution is not well explored. However, having a low acceptance rate may indicate the random walk frequently 'walk' on the boarder of the support of target distribution and hit 'barrier' too many times. So a low acceptance rate can be a poor convergence pattern as well.

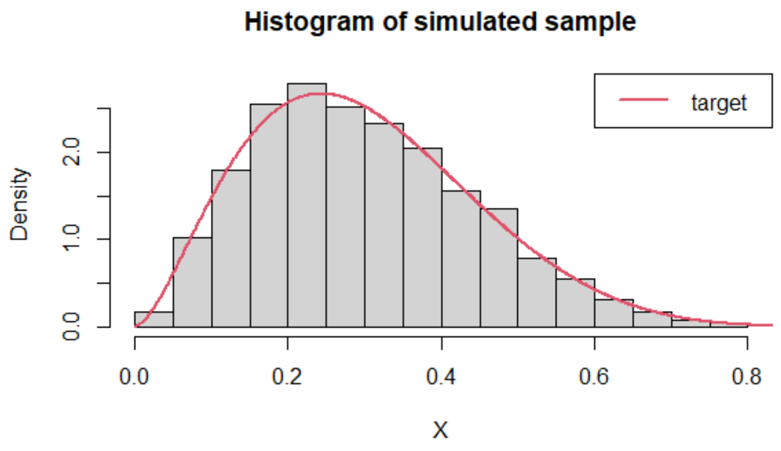


Figure 3.4: The histogram of simulated $Beta(2.7, 6.3)$ via random walk Metropolis, the red curve is the true target density.

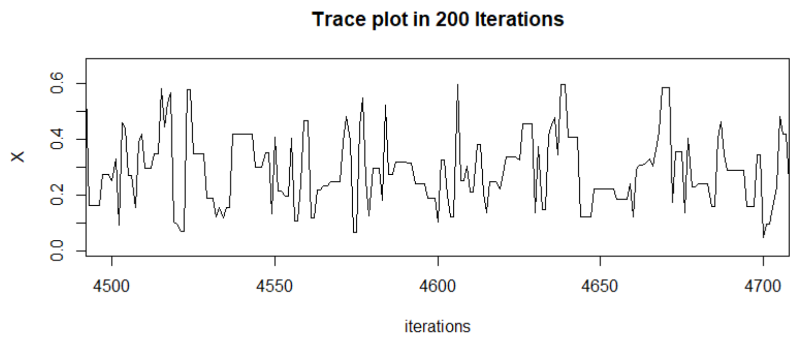


Figure 3.5: Partial magnification of trace plot using random walk Metropolis.

3.4 Gibbs Sampler

Metropolis-Hastings algorithm provides a beautiful solution for sampling from an arbitrary target posterior distribution π of any dimension provided we know it up to a constant. But there are still some points to be improved in practice: (i) The Metropolis-Hastings algorithm requires calculating an acceptance probability for each iteration, which can be computationally costly when the joint posterior distribution and proposal distribution are high dimensional. (ii) In some cases, the acceptance rate is low no matter how the proposal is tuned. This means that a large number of iterations are rejected and it takes a long time to generate enough effective samples. (iii) For high dimensional data, it is likely that the joint distribution is hard to access, while some marginal distribution for our parameter of interest is available. Metropolis-Hastings algorithm can not handle this situation. Fortunately, Gibbs sampler is designed to perform better for these scenarios.

3.4.1 Transition Kernel from Full Conditionals

Recall that the key to constructing a MCMC algorithm is to find its transition kernel P. Any transition kernel satisfying detailed balance condition for target posterior π may give rise to a π-irreducible, aperiodic, positive recurrent Markov chain converging to π. So, once more, we dedicate to find a transition kernel satisfying detailed balanced condition, but this time we wish to utilize marginal distribution for high dimensional cases.

To begin with, we consider 2-dimensional cases in the x-y plane. The target distribution is $\pi(x, y)$. For any two distinct points with the same x-coordinate $A(x_1, y_1)$ and $B(x_1, y_2)$, by definition of conditional probability

$$\begin{aligned}\pi(x_1, y_1)\pi(y_2|x_1) &= \pi(x_1)\pi(y_1|x_1)\pi(y_2|x_1)\\ &= \pi(x_1)\pi(y_2|x_1)\pi(y_1|x_1)\\ &= \pi(x_1, y_2)\pi(y_1|x_1)\end{aligned} \tag{3.27}$$

Rewrite the conditional distribution $\pi(y_2|x_1), \pi(y_1|x_1)$ as $\pi_{x_1}(y_2), \pi_{x_1}(y_1)$ respectively

$$\pi(A)\pi_{x_1}(y_2) = \pi(B)\pi_{x_1}(y_1) \tag{3.28}$$

take π_{x_1} as the transition kernel, then the limiting probability of transition from state y_1 to state y_2 converges to $\pi_{x_1}(y_2)$ and the limiting probability of transition from state y_2 to state y_1 converges to $\pi_{x_1}(y_1)$ (theorem 3.2.11). Hence detailed balance condition is satisfied. Likewise, for any two distinct points with the same second coordinate $A(x_1, y_1)$ and $C(x_2, y_1)$, we have the following for detailed balance condition for π

$$\pi(A)\pi_{y_1}(x_2) = \pi(C)\pi_{y_1}(x_1) \tag{3.29}$$

Now, we can construct the transition kernel Q on two dimensional space: For any two points U, V, if they have the same first coordinate $x_U = x_V = x_0$, the transition probability from U to V is

$$Q(U, V) = \pi_{x_0}(y_V) \tag{3.30}$$

If they have the same second coordinate $y_U = y_V = y_0$, the transition probability from U to V is

$$Q(U, V) = \pi_{y_0}(x_V) \tag{3.31}$$

If no coordinates of U and V are the same, define

$$Q(U, V) = 0 \tag{3.32}$$

The transition kernel Q defined in this way satisfies detailed balance condition

$$\pi(U)Q(U, V) = \pi(V)Q(V, U) \tag{3.33}$$

Thus, the transition kernel Q formed by marginal probability can produce a Markov chain converging to the target posterior π. This construction of transition kernel can be easily generalized to high dimensional cases. For d-dimensional target posterior $\pi(x_1, x_2, \ldots, x_d)$, all we need to change is to consider fixing $d-1$ coordinates and transit the j-th coordinate x_j according to conditional probability (also known as full conditional)

$$\pi(x_j|x_{-j}) = \pi(x_j|x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_d) \tag{3.34}$$

for every $1 \leq j \leq d$ in each iteration. This transition kernel Q induces Gibbs sampler of any dimension.

3.4.2 Gibbs Sampler

Algorithm 4 Gibbs sampler

```
1. determine the number of simulation $N$ and an arbitrary initial point $\boldsymbol{x}^{(0)}$
2. for $i = 0$ to $N$
     generate $X_1^{(i+1)} \sim \pi(x_1|x_2^{(i)}, \ldots, x_d^{(i)})$
     generate $X_2^{(i+1)} \sim \pi(x_2|x_1^{(i+1)}, x_3^{(i)}, \ldots, x_d^{(i)})$
     ...
     generate $X_d^{(i+1)} \sim \pi(x_d|x_1^{(i+1)}, x_2^{(i+1)}, \ldots, x_{d-1}^{(i+1)})$
```

The Gibbs sampler works well for problems of any dimension as long as the full conditionals are available. This provides good decomposition for high dimensional posterior, so that the joint distribution of posterior can be unnecessary.

Even the posterior is very complicated, Gibbs sampler still treat it as several sampling from univariate full conditionals, which is much more computational efficient than calculating joint densities. Another advantage is that Gibbs sampler do not apply the accept-reject procedure, so that every iterations indicate a move in Markov chain and there is no need to concern the acceptance rate. Also, Gibbs sampler do not require manually tuning parameters for proposals. Hence, there has been a domination of Gibbs sampler among MCMC algorithms for Bayesian analysis for a long time. And a special Bayesian analysis software, BUGS(Bayesian inference Using Gibbs Sampling) is made for Gibbs sampler.

Example 3.4.1 *In this toy example, our aim is to simulate the two dimensional target* $\pi(x, \theta)$ *with* $X|\theta \sim Bin(n, \theta)$ *and* $\theta \sim Beta(a, b)$*. The full conditional of* X *is already available, the full conditional of* θ

$$\theta|x \sim Beta(x + a, n - x + b) \tag{3.35}$$

Set $n = 15$*,* $a = 3$*,* $b = 7$*, and we are able to simulate 5000 samples. Histograms in Figure 3.6 illustrate the samples are well drawn from the target distribution by comparing with each marginal distributions. Figure 3.7 demonstrates the trace plots of each parameters, which in comparison to Metropolis-Hastings algorithms are hardly remain in the same value for a long time.*

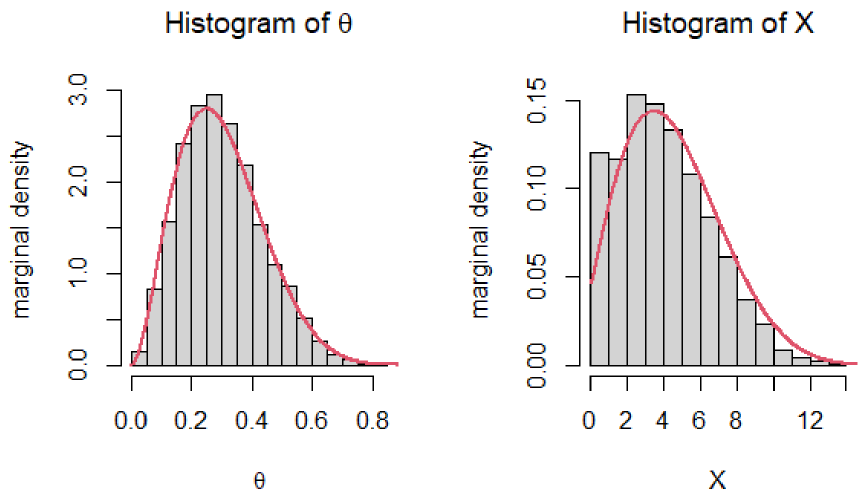


Figure 3.6: Histograms for marginals of sampled μ and τ via Gibbs sampler and the red curves are true marginals.

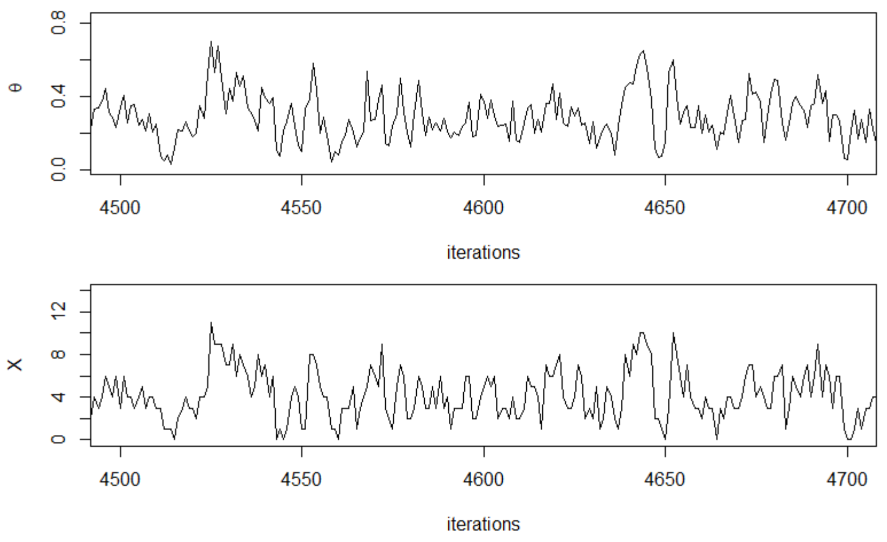


Figure 3.7: Partial magnification of trace plots for μ and τ via Gibbs sampler in 200 iterations.

3.5 Comparison among Algorithms

After deriving and understanding above MCMC algorithms, it is better to look into some examples of Bayesian analysis with generated real data and compare these algorithms in two different examples. The full standard procedure includes necessary forming priors, tuning proposals and diagnostic checking, etc.

3.5.1 Estimating Parameters for Normal Model

Example 3.5.1 *Suppose $x_1, x_2, \dots x_n$ are iid realizations of normal distribution $N(\mu, 1/\tau)$ with unknown mean and precision. Our aim to to use given sample to estimate parameters μ and τ and compare them with true values $\mu = 3$, $\tau = 0.25$.*

Independent Metropolis-Hastings algorithm

To implement the independent Metropolis-Hastings algorithm, we first assign two parameters with diffuse priors $N(0, 1)$ for μ and $Exp(1)$ for τ. Then we choose proposal for μ to be $N(m, s^2)$ and proposal for τ to be $Gamma(\alpha, \beta)$ $(\tau > 0)$, where $m = \bar{x}$, $s = 2\hat{s}^2/\sqrt{n}$ (twice the standard error so that the proposal has a larger variance than $var(\mu)$), $\alpha/\beta = 1/\hat{s}^2$ (sample precision), $\alpha = \sqrt{n}$ (variance of the proposal for τ is larger than $var(\tau)$).

The output of posterior mean are close to true values $\hat{\mu} = 2.626163$, $\hat{\tau} = 0.2126147$. Figure 3.8 below shows that the Markov chains for μ and τ converge to stationary distributions very quickly, and autocorrelations decay quickly as well, which are signs for well constructed Markov chains. Figure 3.9 is the histogram of simulated posterior for two parameters, where the result is almost consistent with true values. Also, the accept rate of 0.3684 is not bad and indicates there might be better proposals leading to greater accept rate.

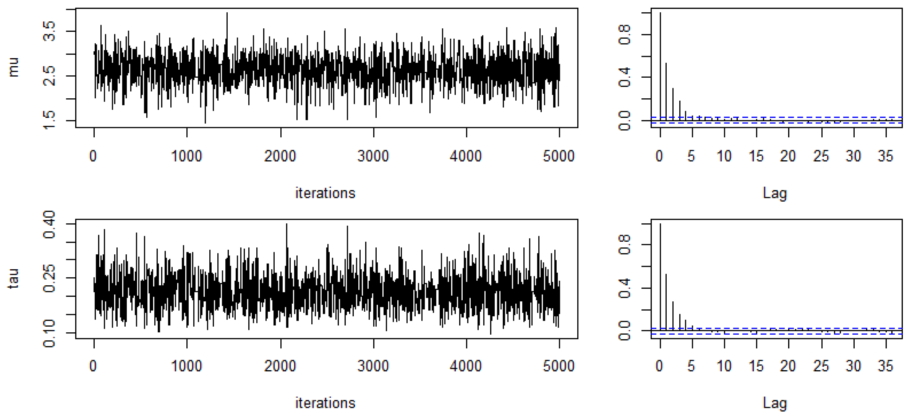


Figure 3.8: Trace plots and autocorrelation plots for μ and τ via independent Metropolis-Hastings.

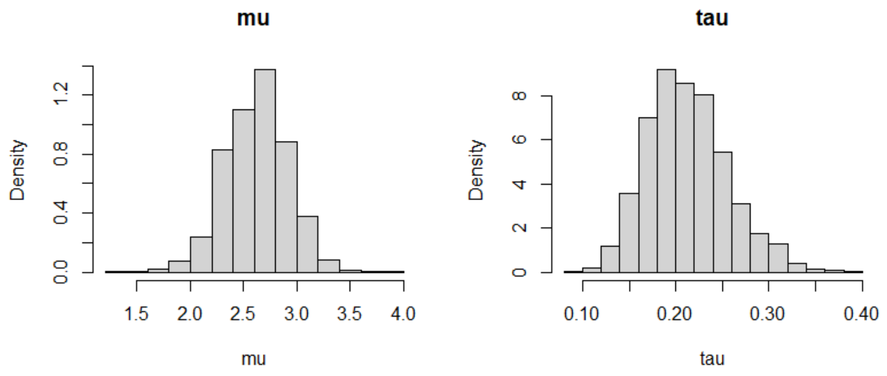


Figure 3.9: Histograms of simulated posterior distribution for μ and τ.

It is important to address that it is necessary to choose proposals that have larger variance than unknown target posteriors. Since independent sampler almost always accept candidates with large weights $w = \pi/q$, it is likely for independent sampler to stuck in some high weight regions (usually some tails) for a long time. Figure 3.10 and 3.11 illustrate the problems of using $s = 0.5\hat{s}^2/\sqrt{n}$ variance of μ is smaller than posterior variance of μ: There are strong correlations in Markov chains and new candidates can not be accepted for a long time so that the accept rate is only 0.2282 and some regions of μ and τ are over-sampled.

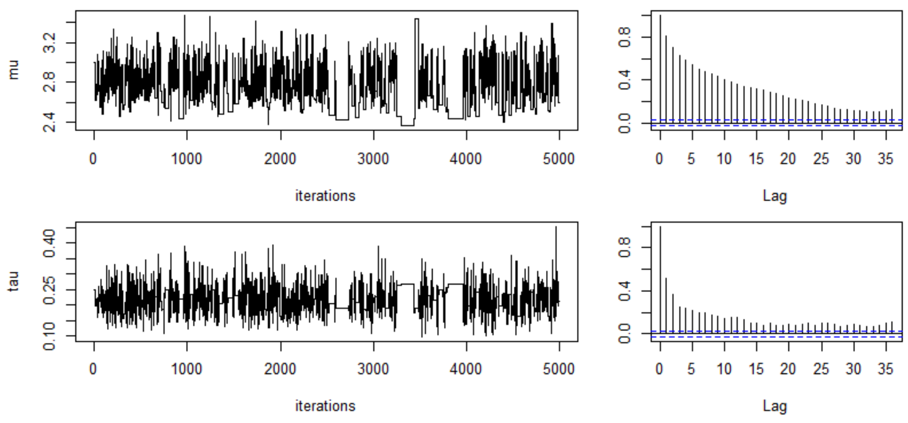


Figure 3.10: Trace plots and autocorrelation plots for μ and τ via independent sampler when the proposal of μ has smaller variance than target posterior.

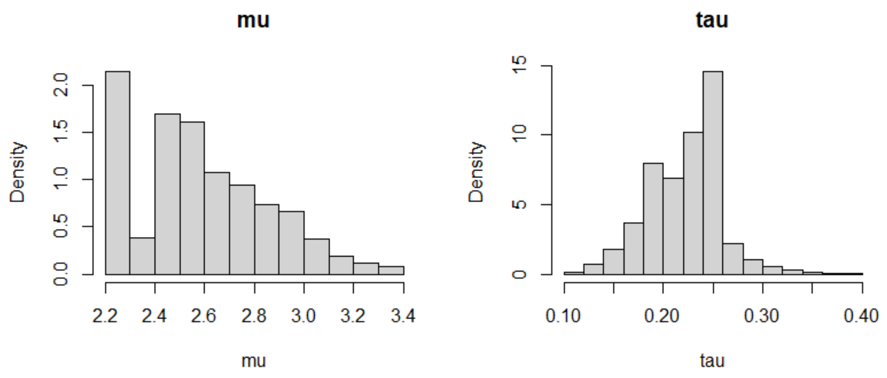


Figure 3.11: Histograms of simulated posterior distribution for μ and τ via independent sampler when the proposal of μ has smaller variance than target posterior.

Random walk Metropolis-Hastings algorithm

Consider the same problem, but this time we use random walk Metropolis-Hastings to provide estimates for μ and τ.

To implement random walk Metropolis-Hastings algorithm, we first set priors to be $\mu \sim N(3, 1)$ and $\tau \sim Exp(1)$. For the increments in random walks, we set proposals $N(0, 0.5^2)$ and $N(0.0.05^2)$ for μ and τ respectively. In contrast to the previous independent Metropolis-Hastings algorithm, though τ must be positive, we have to choose symmetric proposals and then reject any candidate with non-positive value for τ.

The result is visualized by Figure 3.12 and 3.13 , where stationary distribution is generally reached quickly and autocorrelation decays fast, though not as fast as independent sampling. The posterior mean of μ and τ are 2.891286 and 0.2183824 respectively, which are very close to true values 3 and 0.25. The histograms of samples of μ and τ are generally consistent with true values as well.

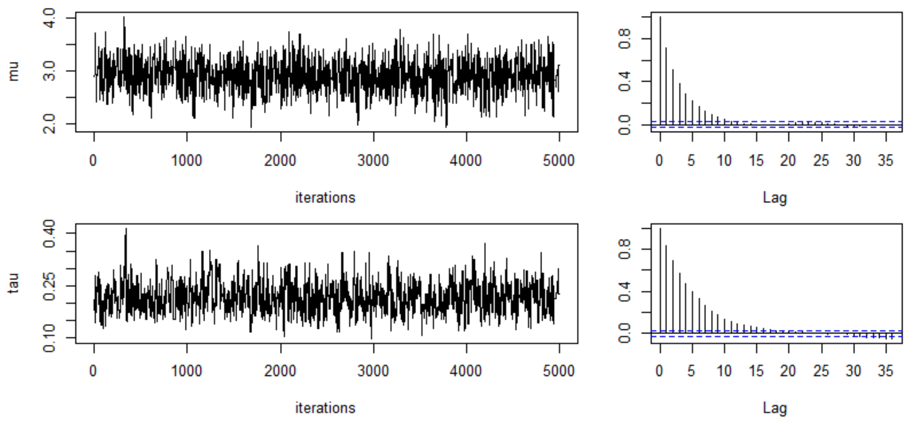


Figure 3.12: Trace plots and autocorrelation plots for μ and τ via random walk Metropolis-Hastings algorithm.

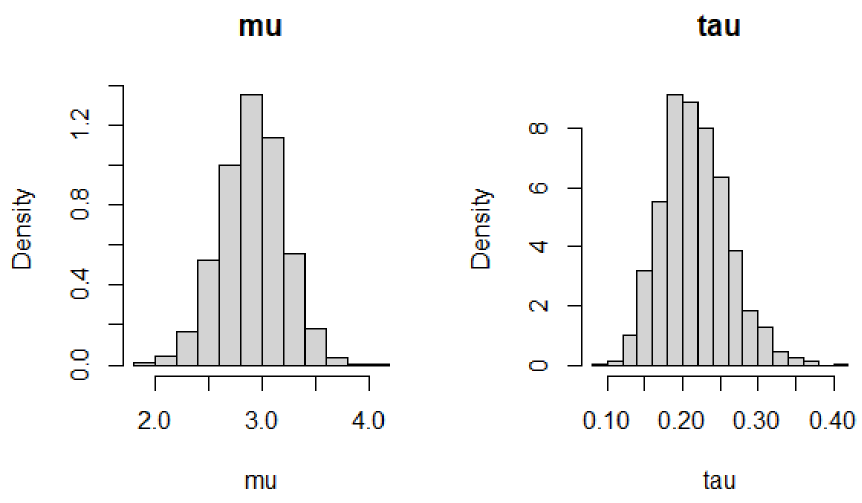


Figure 3.13: Histograms of simulated posterior distribution for μ and τ via random walk Metropolis-Hastings algorithm.

Similar to independent Metropolis-Hastings algorithm, tuning parameters for proposals of μ and τ are of great importance. Theoretically, applying proposals with large variance or large tails usually prevents getting stuck in some local region for a long time. If the variance of proposals are too small, it takes a great number of iterations to escape one area and explore the whole distribution. Therefore, the acceptance rate may be too large, samples can be highly correlated. The final result of overall acceptance rate is 0.41 after tuning. Even though the golden rule recommended by Robert and Casella (2004) is 0.5 for two dimensional model, it is not necessary to always reach that criterion and some models cannot reach it at all. Figure 3.14 and 3.15, demonstrate the outcome of problematic proposals $N(0, 0.05^2)$ for μ and $N(0, 0.005^2)$ for τ, where the trace plots are very trendy and autocorrelation decays extremely slowly.

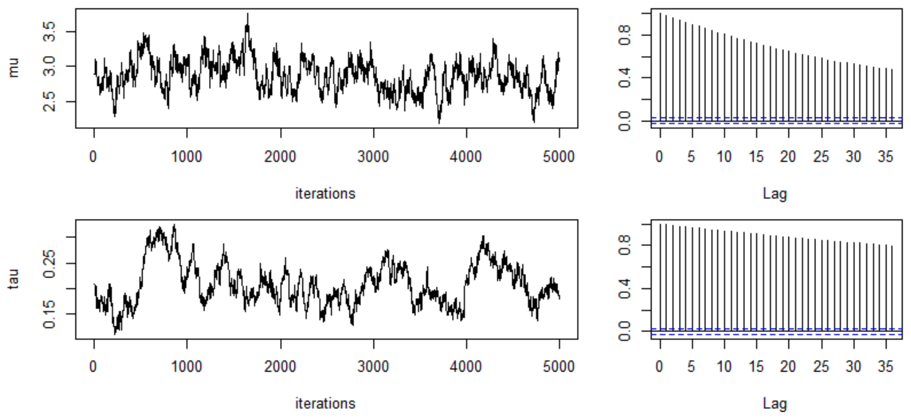


Figure 3.14: Trace plots and autocorrelation plots for μ and τ via random walk Metropolis-Hastings algorithm using proposals with small variances.

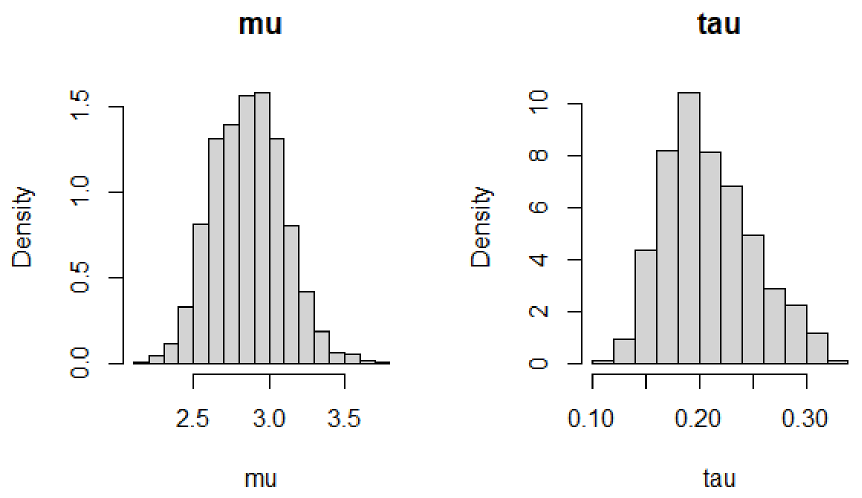


Figure 3.15: Histograms of simulated posterior distribution for μ and τ via random walk Metropolis-Hastings algorithm using proposals with small variances.

Gibbs Sampler

Consider the same example using Gibbs sampler. Set diffuse priors for μ and τ to be $\mu \sim N(\lambda_0, 1/w_0)$ and $\tau \sim Gamma(a_0, b_0)$ and assume μ and τ are apriori independent. Then it is easy to find full conditionals for μ and τ given data $\boldsymbol{x}$

$$\mu|\tau, \boldsymbol{x} \sim N(\frac{\tau \sum_{i=1}^{n} x_i + \lambda_0 w_0}{n\tau + w_0}, \frac{1}{n\tau + w_0}) \tag{3.36}$$

$$\tau|\mu, \boldsymbol{x} \sim Gamma(a_0 + \frac{n}{2}, b_0 + \frac{1}{2}\sum_{i=1}^{n}(x_i - u)^2) \tag{3.37}$$

In this example, hyperparameter values are set to be $\lambda_0 = 3$, $w_0 = 1$, $a_0 = 3$, $b_0 = 7$. The posterior mean of μ and τ are 2.89 and 0.22 respectively. Histograms of simulated μ and τ in Figure 3.16. Both of these shows good alignments with hidden true values $\mu = 3$ and $\tau = 0.25$. In Figure 3.17, the trace plot demonstrates a very short burn-in and quick convergence to target posterior. There is a cut off in autocorrelation at lag 1, which is even more desirable than the quick decay we expected.

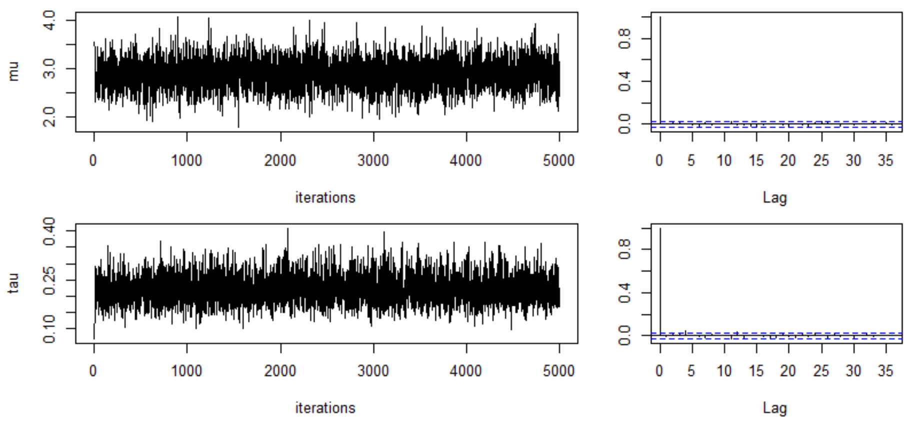


Figure 3.16: Trace plots and autocorrelation plots for μ and τ via Gibbs sampler.

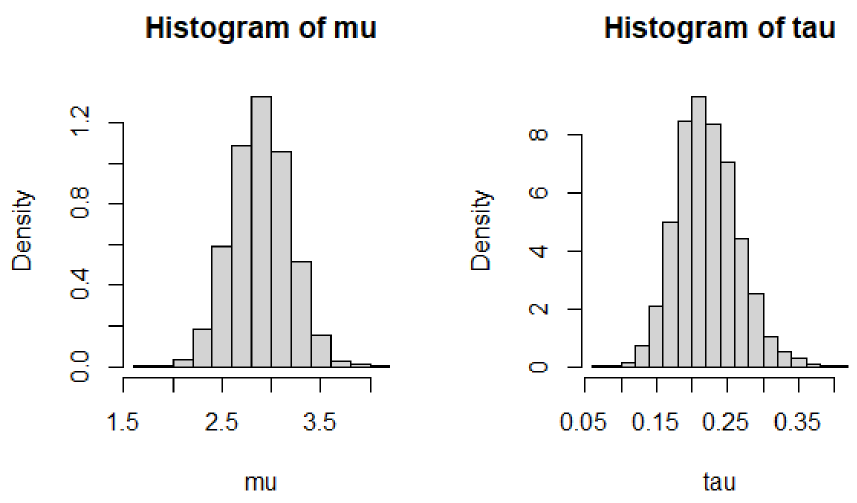


Figure 3.17: Histograms of simulated posterior distribution for μ and τ via Gibbs sampler.

Conclusion

Three different algorithms are conducted above for estimating parameters μ, τ in the normal model $N(\mu, 1/\tau)$ with provided data. All posterior estimates are very close to the hidden true parameter values of μ, τ.

Among these algorithms, Gibbs sampler gives the best result, where the trace plots show it explores the parameter space very well, the Markov chain converges most quickly and autocorrelation disappear after lag 1. Gibbs sampler should be the first option for this example, since the full conditional can be easily derived in a close form and it does not require any parameter tuning. For computational consideration, Gibbs sampler is the more computationally efficient algorithm compared to the two Metropolis-Hastings algorithms. The Gibbs sampler demonstrates very robust performance in multiple running chains.

The independent Metropolis-Hastings algorithm performs a little worse than Gibbs sampler, but still provide very good estimates in terms of trace plot and autocorrelation when proposal variances are ensured to be larger than corresponding targets. A big drawback is that calculating proposals for each acceptance probabilities are very computational costly. The independent Metropolis-Hastings is also very robust when running multiple chains.

Random walk Metropolis algorithm is the least recommended for this example, because it is really hard to find adequate random proposals for both μ and τ such that the overall acceptance rate is neither too small nor too large, the autocorrelation decays relatively slowly as well. Sometimes, it is necessary to consider thinning the Markov chain. In addition, the random walk on a bounded support requires the algorithm to mind the boundary at every step. The additional conditions complex the MCMC computation, even though proposals can be cancelled in calculation of acceptance probabilities. Finally, random walk Metropolis is not a very robust algorithm for this example. Results from several chains can sometimes disagree.

3.5.2 Estimating Parameters for Gaussian Mixture Model

Example 3.5.2 *Consider data X is a sample drawn from a Gaussian mixture with two components $0.7N(-1,1)+0.3(2,1)$. Suppose the true values of this model is known and our aim is to provide estimates for parameters in the model $pN(\mu_1, 1/\tau_1)+(1-p)N(\mu_2, 1/\tau_2)$ given generated data. i.e. there are 5 parameters to estimate μ_1, τ_1, μ_2, τ_2, p. To integrate the prior information, choose priors to be*

$$\mu_1 \sim N(-1,4), \mu_2 \sim N(2,4) \tag{3.38}$$

$$\tau_1 \sim Exp(1), \tau_2 \sim Exp(1) \tag{3.39}$$

$$p \sim Beta(3,2) \tag{3.40}$$

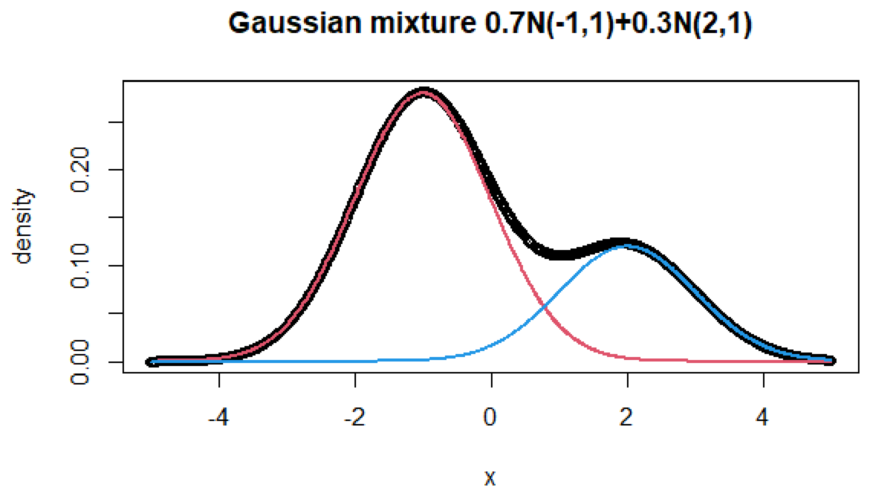


Figure 3.18: Mixture distribution and its two normal components.

Independent Metropolis-Hastings algorithm

For the independent sampler, we can set proposals

$$\mu_1 \sim N(-1,s^2), \mu_2 \sim N(2,s^2) \tag{3.41}$$

$$\tau_1 \sim Gamma(\alpha_1, \beta_1), \tau_2 \sim Gamma(\alpha_2, \beta_2) \tag{3.42}$$

$$p \sim Beta(a,b) \tag{3.43}$$

Given density plot of generated data x, we can assign values for parameters: $s = sd(x)/\sqrt{n}$ (use mixture standard error so that the variance of proposals are larger than unknown target), $\alpha_1 = 2$, $\beta_1 = 2$, $\alpha_2 = 2$, $\beta_2 = 2$ ($\alpha/\beta = 0.5$, also propose relatively large variances for τ_1 and τ_2), $a = 7$, $b = 3$ (the left component is higher so set $a > b$).

The two component of the mixture can be well distinguished by implementation of independent Metropolis-Hastings. The algorithm gives point estimates (posterior mean) for parameters

$$\mu_1 = -1.03, \tau_1 = 0.96, \mu_2 = 2.10, \tau_2 = 0.90, p = 0.68 \tag{3.44}$$

which are very close to their hidden true values. Figure 3.19 shows the trace plots and autocorrelation for each parameters, where the five Markov chains perform stationarity in their traces and autocorrelations are not significant after lag 20. Autocorrealations can not decay as quick as previous examples due to the complexity of mixture model. The acceptance rate is 0.26, which is a result of tuning proposal parameters, even it is relatively small number compared to the normal example. However, this example considers a 5 dimensional parameter space, so finding ideal proposal for target distribution can be harder, and one drawback of Metropolis-Hastings algorithm is that it can have very small acceptance rate when dimension gets high.

Random walk Metropolis-Hastings algorithm

For random walk Metropolis, we set proposed random walk in $\mu_1, \tau_1, \mu_2, \tau_2$ to be $N(0, 0.25^2)$ and random walk in p to be $N(0.0.05^2)$.

The point estimates given by random walk Metropolis are

$$\mu_1 = -0.91, \tau_1 = 1.10, \mu_2 = 1.96, \tau_2 = 1.53, p = 0.66 \tag{3.45}$$

Apart from the estimate on τ_2, which is a little far from true value, the rest estimates are acceptable for corresponding parameters. The implemented trace plots and acf plots are shown in Figure 3.20, where the trace plots are sometimes trendy and not stable. The autocorrelation decays very slowly, even though the acceptance rate is a good number of 0.26. In general, random walk Metropolis gives less accurate estimates and performs worse than independent sampler on these diagnostic issues in this example.

Gibbs sampler

To implement Gibbs sampler, we need to first figure out full conditions for each parameters. However, this is not an easy task for Gaussian mixture model where the likelihood is a product of some component sums

$$L(\boldsymbol{\theta}|\boldsymbol{x}) = \prod_{i=1}^{n} [pf_1(x_i|\mu_1, 1/\tau_1) + (1-p)f_2(x_i|\mu_2, 1/\tau_2)] \tag{3.46}$$

So we can do a data argumentation (Lu (2021)) by introducing latent allocation variables $z_1, z_2, \ldots, z_n$, where $z_i = 1$ if x_i comes from the first component and $z_i = 0$ if x_i comes from the second component. Then, all parameters include $\mu_1, \mu_2, \tau_1, \tau_2, p, \boldsymbol{z}$, the likelihood becomes

$$L(\boldsymbol{\theta}|\boldsymbol{x}) = \prod_{i=1}^{n} [pf_1(x_i|\mu_1, 1/\tau_1)]^{z_i} [(1-p)f_2(x_i|\mu_2, 1/\tau_2)]^{1-z_i} \tag{3.47}$$

Recall that the priors set for the example are

$$\mu_1 \sim N(-1, 4), \mu_2 \sim N(2, 4) \tag{3.48}$$

$$\tau_1 \sim Exp(1), \tau_2 \sim Exp(1) \tag{3.49}$$

$$p \sim Beta(3, 2) \tag{3.50}$$

Then we have full conditionals

$$p|\boldsymbol{\theta}_{-p}, \boldsymbol{x} \sim Beta(3 + n_1, 2 + n_2), with\ n_1 = \sum z_i, n_2 = n - \sum z_i \tag{3.51}$$

$$\mu_1|\boldsymbol{\theta}_{-\mu_1}, \boldsymbol{x} \sim N(\frac{0.25 \cdot (-1) + \tau_1 \sum x_i z_i}{0.25 + n_1\tau_1}, \frac{1}{0.25 + n_2\tau_1}) \tag{3.52}$$

$$\mu_2|\boldsymbol{\theta}_{-\mu_2}, \boldsymbol{x} \sim N(\frac{0.25 \cdot 2 + \tau_2 \sum x_i(1 - z_i)}{0.25 + n_2\tau_2}, \frac{1}{0.25 + n_2\tau_2}) \tag{3.53}$$

$$\tau_1|\boldsymbol{\theta}_{-\tau_1}, \boldsymbol{x} \sim Gamma(1 + \frac{n_1}{2}, 1 + \frac{1}{2}\sum[(x_i - \mu_1)z_i]^2) \tag{3.54}$$

$$\tau_2|\boldsymbol{\theta}_{-\tau_2}, \boldsymbol{x} \sim Gamma(1 + \frac{n_2}{2}, 1 + \frac{1}{2}\sum[(x_i - \mu_2)(1 - z_i)]^2) \tag{3.55}$$

$$\boldsymbol{z}|\boldsymbol{\theta}_{-z}, \boldsymbol{x} \sim Bin(n, \frac{pf_1}{pf_1 + (1 - p)f_2}) \tag{3.56}$$

As a result, the estimated values for parameters are

$$\mu_1 = -1.06, \tau_1 = 0.99, \mu_2 = 2.84, \tau_2 = 0.92, p = 0.70 \tag{3.57}$$

which are very accurate except μ_2 is a little bit larger than the true value. In figure 3.21, trace plots visualize the convergence of Markov chains to stationary distribution. Autocorrelations also decay fast and almost disappear in lag 20.

Conclusion

Both independent Metropolis-Hastings and Gibbs sampler are viable for estimating parameters in this two component Gaussian mixture model. According to trace plots, Gibbs sampler seems to explore the parameter space a bit better. The independent Metropolis-Hastings algorithm requires selecting forms of proposals and tuning parameters for proposals, which may take one's energy. Sometimes we know there is something wrong, but we have no idea which proposal of the five proposals still need to optimized, so multiple attempts are demanding. Moreover, it is necessary to also ensure the variances of proposals are not too large. When proposals have large variances, interestingly, it is possible for the Markov chain to switch from one state to the other state or even jump back and forth. In brief, both algorithms performs good job in this example. For avoiding tedious tuning process in high dimensional space, it is preferable to apply Gibbs sampler first.

The random walk Metropolis is not recommended route for this example, as the trace plots have little sign of convergence and there are a great amount of correlation among samples. The overall result of this algorithm is not very satisfying even though the point estimates are close to true value and acceptance rate is near 0.25.

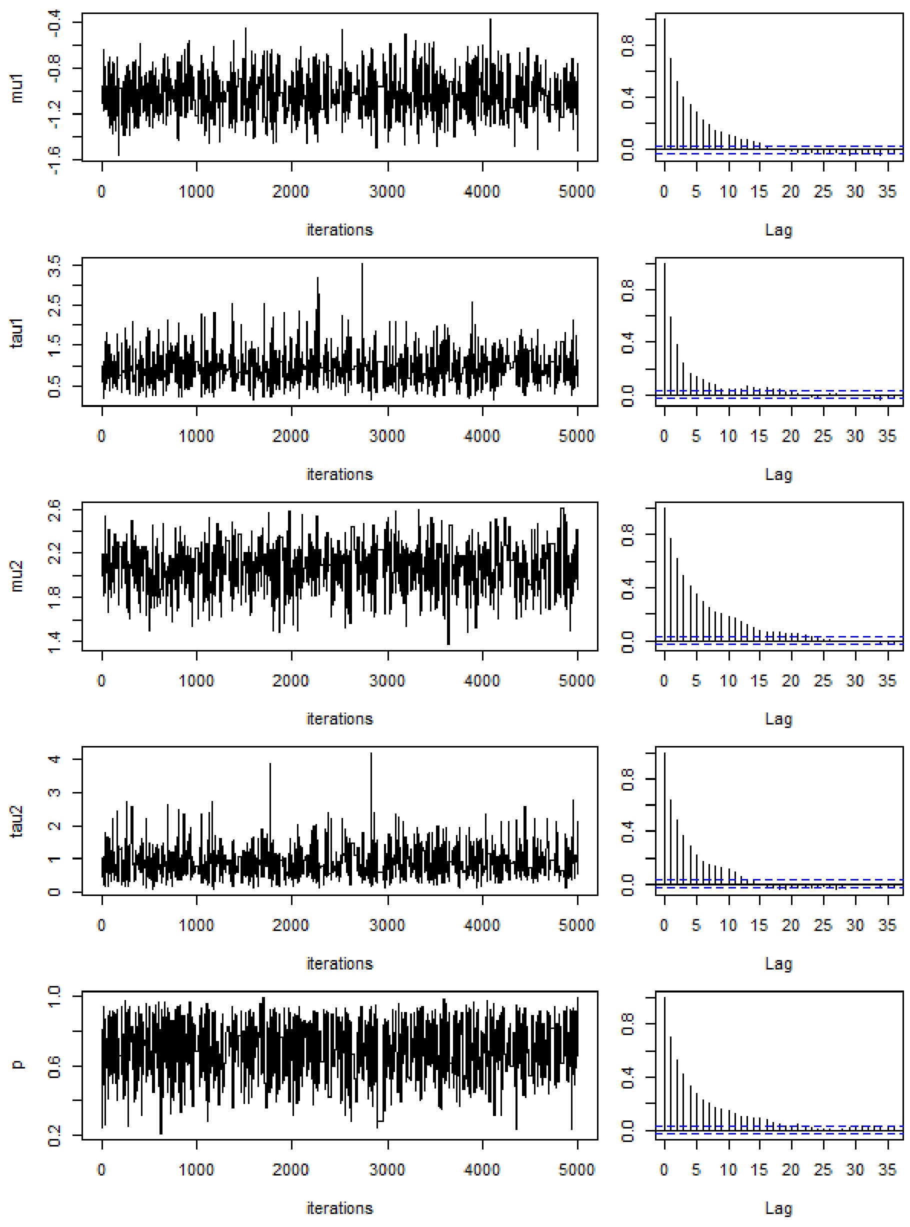


Figure 3.19: Trace plots and autocorrelation plots for each parameters via independent Metropolis-Hastings.

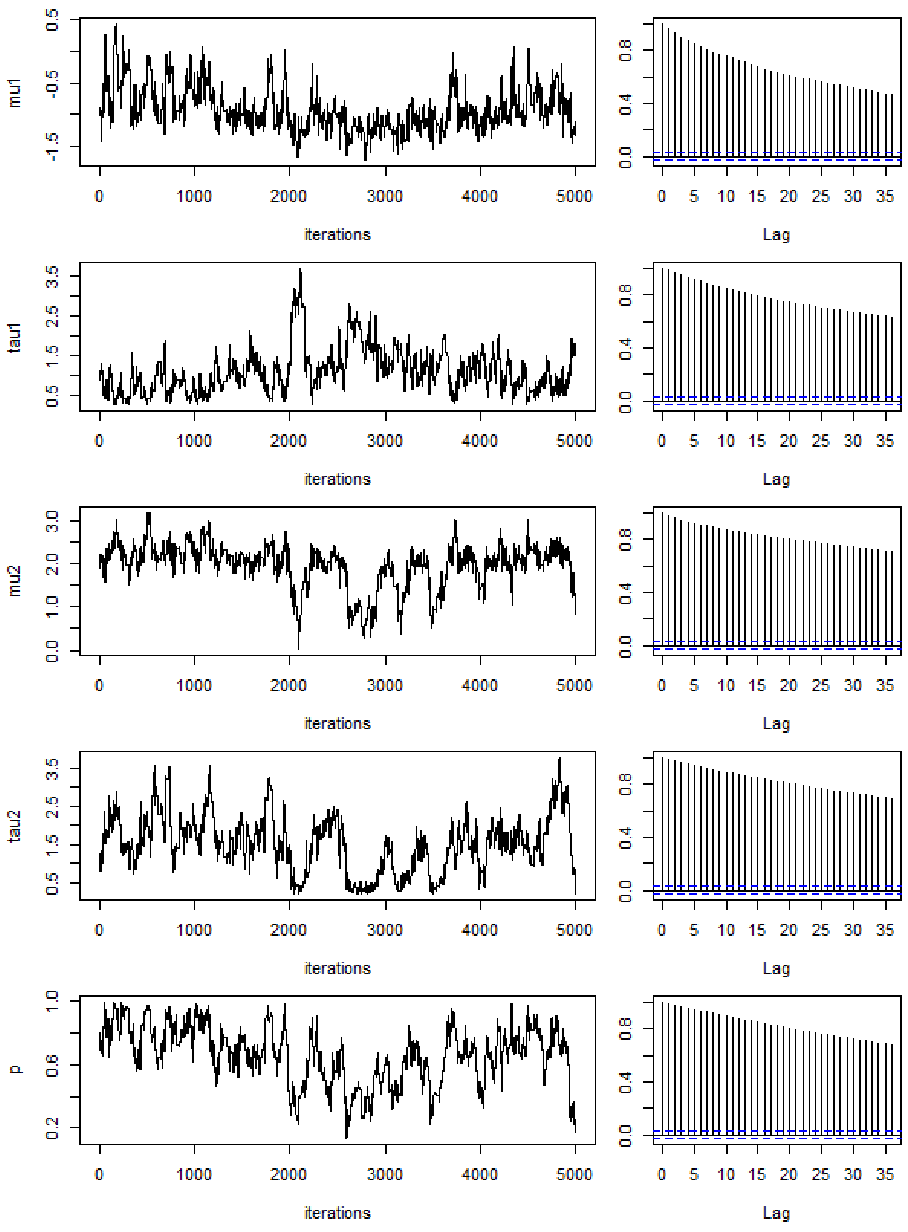


Figure 3.20: Trace plots and autocorrelation plots for each parameters via random walk Metropolis.

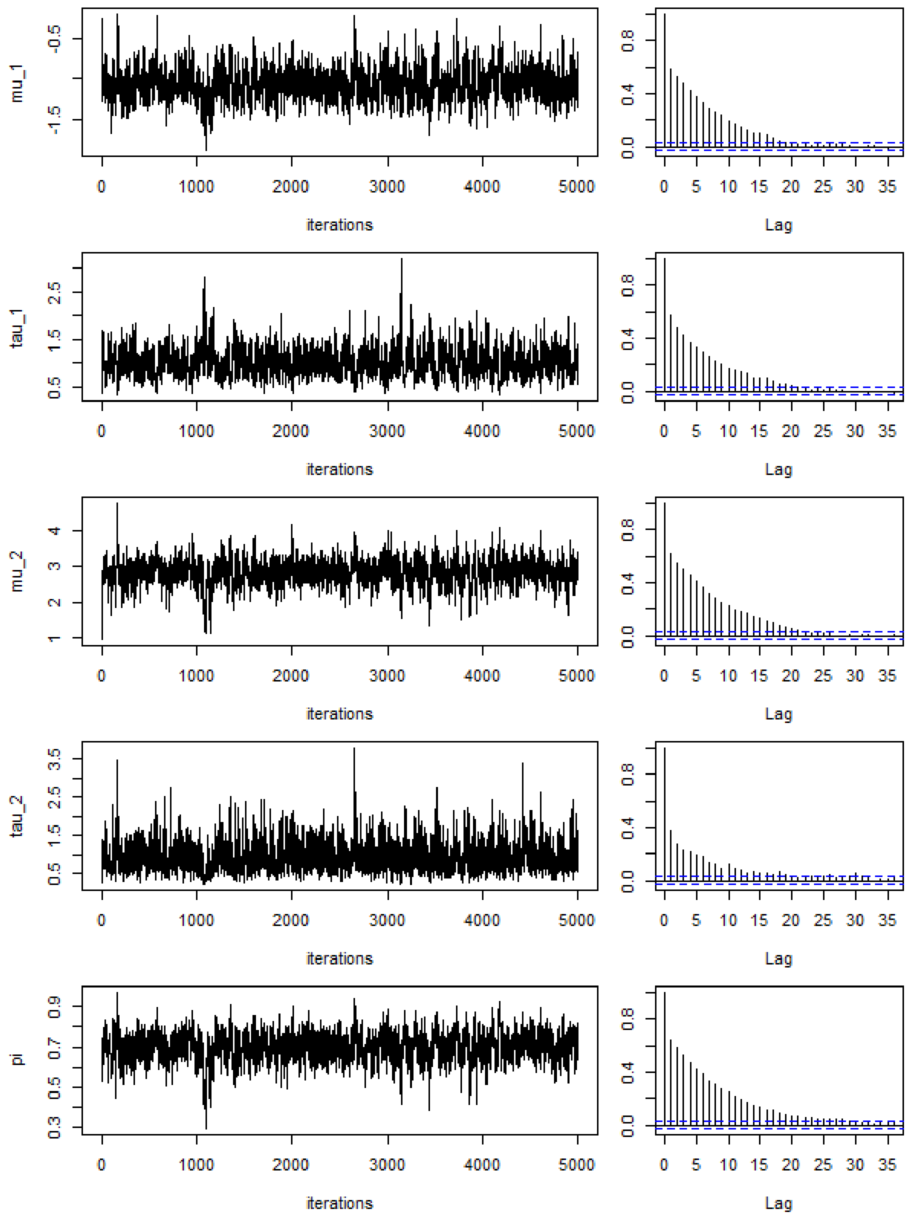


Figure 3.21: Trace plots and autocorrelation plots for each parameters via Gibbs sampler.

Chapter 4

Summary and Outlook

4.1 Summary of the Report

The report mainly talks about Bayesian analysis via basic Markov chain Monte Carlo methods. The first chapter, we introduces Bayesian analysis is the statistical inference paradigm in Bayesian framework. The work flow of Bayesian analysis mainly consists of formalizing likelihood, prior and calculate posterior for inferences. This updating information process is a key difference between frequentist and Bayesian. Some simple examples are given to demonstrate Bayesian analysis procedure. Then prior elicitation and checking are briefly discuss to have a glance of the selection of priors and prior predictive checking for the purpose of getting a bigger picture of Bayesian analysis. The core challenge in Bayesian analysis lies in how to do integration on high dimensional regions, which leads to the curse of dimensionality of classical methods ($O(n^{-\frac{1}{d}})$) and the rise of the sampling-based method - Monte Carlo integration ($O(\frac{1}{\sqrt{n}})$). Finally, the history of Bayesian analysis development is reviewed.

Since Monte Carlo method is a simulation-based method, chapter 2 tries to solve the generic problem of sampling from an arbitrary target distribution. For this purpose, several classical simulation methods are introduced. The chapter starts from simulating uniform random variables, which is the bedrock of all simulations. Then inverse cdf method generalizes the uniform generator by establishing the target and uniform distribution through generalized inverse cdf when the cdf of target distribution is available. When cdf is not available, general transformation methods can be considered by first sampling from a distribution that is easy to simulate and then transforming it to the target distribution. In order to provide a sampling method for non-standard distributions, accept-reject method is put forward, where a proposal distribution is sampled first and then each samples are rejected with some probabilities to get the target distribution. Importance sampling is another indirect sampling strategy in Monte Carlo, where we also sample from a proposal and add some additional importance weights for samples to make it have the same performance as sampling from the target distribution.

However, these classical methods are either not capable of high dimensional sampling or too inefficient approaching the target distribution, so the Markov chain is brought in to form Markov chain Monte Carlo in chapter 3. The brilliant idea of MCMC is independency -efficiency tradeoff, which means that a little dependency between samples are produced in sampling for a big increase in efficiency of algorithms. After a mini-refresher on Markov chain, chapter 3 delves into Metropolis-Hastings algorithm and Gibbs sampler by first naturally demonstrate the deviation of their transition kernels via detailed balance condition, then describe formal algorithms and provide toy examples. This narrative follows a complete 'rejection-Metropolis-Hastings-Gibbs' story line and should be easier for beginners to grasp why the given algorithms construct a Markov chain converging to the target distribution. Finally, to implement MCMC algorithms in a Bayesian analysis context, two examples with normal data and Gaussian mixture are used for comparison among different algorithms, parameter tuning and diagnostics are discussed in those examples as well. As a result, when full conditionals of parameters can be derived, it is generally favorable to consider Gibbs sampler in the first place for avoiding tedious tuning issues and saving computational forces. Random walk should be adopted carefully, as samples can be unstable and highly correlated in some models.

4.2 Overview of Other Techniques

Due to the widespread adoption of Bayesian analysis, a great many procedures and methods are developed in this flexible and powerful inference framework. The report focuses on the necessities of MCMC method, so they are beyond our scope, but it is still important to mind their existence.

In Bayesian framework, it is usually very important to do variable selection to identify the predictors in the model, especially when there are many potential explanatory variables. Any variables leading to multicollinearity or overfitting should be excluded from the model. There are two ways of variable selection: hypothesis test methods, such as Bayes factor and penalization methods, like conducting shrinkage priors. When the posterior is obtained by a MCMC method, usually there will be a posterior predictive checking procedure, where new data will be simulated based on posterior to check whether the model can well predict future data. This is the evaluation of the ability for fitting underlying data generating process.

For MCMC methods, there are also more advanced methods and most of them are based on Metropolis-Hastings algorithm. Reversible jump MCMC is an extension to Metropolis-Hastings algorithm, which allows trans-dimensional sampling on parameters spaces. Unlike Metropolis-Hastings algorithm and Gibbs, reversible jump MCMC can provide estimate for parameters even we are not sure number of parameters in the model. Hamiltonian Monte Carlo generalizes Metropolis-Hastings algorithm using Hamiltonian dynamics. When direct sampling is difficult, Hamiltonian Monte Carlo can be applied. It avoid algorithms such as random walk Metropolis by imposing additional gradient information.

MCMC is certainly not the only way of sampling in Bayesian analysis. Many other techniques not based on the MCMC idea can also be important in various scenarios. Sequential Monte Carlo is a method mainly based on importance sampling, where several steps can be taken for each data. Thus it is well applied in real time processing and solving Hidden Markov Model problems. Approximate Bayesian computation is an approximation method mainly for solving problems when likelihood is intractable. It can widen the realm of models for inferences when likelihood is not accessible or too computationally costly.

Appendix A

Proof in 1.4.1

we can consider the integral of a differentiable function $f(\boldsymbol{x})$ over a hypercube $[0,1]^d$, where $M = \sup\limits_{\boldsymbol{x}\in[0,1]^d} |\nabla f(\boldsymbol{x})| < \infty$. The Riemann sum approximation by $n = m^d$ partitioned hypercubes (partition each dimension into m parts) is

$$\hat{I}(n) = \frac{1}{m^d} \sum_{a_1=0}^{m-1} \sum_{a_2=0}^{m-1} \cdots \sum_{a_d=0}^{m-1} f(\frac{a_1+1/2}{m}, \frac{a_2+1/2}{m}, \ldots, \frac{a_d+1/2}{m}) \tag{A.1}$$

The approximation error for one partitioned hypercube C_i is

$$\begin{aligned} \varepsilon_{C_i} &= \int_{\boldsymbol{x}\in C_i} f(\boldsymbol{x}) d\boldsymbol{x} - \frac{1}{m^d} f(\frac{a_1+1/2}{m}, \frac{a_2+1/2}{m}, \ldots, \frac{a_d+1/2}{m})) \\ &= \int_{\boldsymbol{x}\in C_i} (f(\boldsymbol{x}) - f(\frac{a_1+1/2}{m}, \frac{a_2+1/2}{m}, \ldots, \frac{a_d+1/2}{m})) d\boldsymbol{x} \\ &\leq \int_{\boldsymbol{x}\in C_i} |f(\boldsymbol{x}) - f(\frac{a_1+1/2}{m}, \frac{a_2+1/2}{m}, \ldots, \frac{a_d+1/2}{m})| d\boldsymbol{x} \end{aligned} \tag{A.2}$$

By the mean value theorem, there exists a $c \in (0,1)$, s.t.

$$f(\boldsymbol{x}) - f(\boldsymbol{y}) = \nabla f((1-c)\boldsymbol{x} + c\boldsymbol{y})(\boldsymbol{x} - \boldsymbol{y}) \tag{A.3}$$

Thus,

$$\begin{aligned} \varepsilon_{C_i} &\leq M \int_{\boldsymbol{x}\in C_i} |\boldsymbol{x} - (\frac{a_1+1/2}{m}, \frac{a_2+1/2}{m}, \ldots, \frac{a_d+1/2}{m})| d\boldsymbol{x} \\ &\leq M \frac{1}{m^d} \frac{\sqrt{d}}{2m} \end{aligned} \tag{A.4}$$

The total approximation error

$$\varepsilon \leq \sum_{i=1}^{m^d} \sup \varepsilon_{C_i} \leq m^d M \frac{1}{m^d} \frac{\sqrt{d}}{2m} = M \frac{\sqrt{d}}{2m} = \frac{M\sqrt{d}}{2} n^{-\frac{1}{d}} \sim O(n^{-\frac{1}{d}}) \tag{A.5}$$

Appendix B

R code

B.1 beta-binomial example

```
library(ggplot2)
library(patchwork)
theme_set(theme_minimal())
# seq creates evenly spaced values
df1 <- data.frame(theta = seq(0.7, 1, 0.001))

#p1
a <- 11
b <- 1
# dbeta computes the posterior density
df1$p <- dbeta(df1$theta, a, b)
# seq creates evenly spaced values from 2.5% quantile
# to 97.5% quantile (i.e., 95% central interval)
# qbeta computes the value for a given quantile given parameters a and b
df2 <- data.frame(theta = seq(qbeta(0.025, a, b), qbeta(0.975, a, b), length.ou
# compute the posterior density
df2$p <- dbeta(df2$theta, a, b)
p1 <- ggplot(mapping = aes(theta, p)) +
  geom_line(data = df1) +
  # Add a layer of colorized 95% posterior interval
  geom_area(data = df2, aes(fill='1')) +
  # Add the proportion of girl babies in general population
  geom_vline(xintercept = (a-1)/(a-1+b-1), linetype='longdash') +
  geom_vline(xintercept = a/(a+b), , linetype='longdash', color='blue') +
  # Decorate the plot a little
  labs(title='Posterior␣A:␣Beta(11,1)', y = '') +
  scale_y_continuous(expand = c(0, 0.1), breaks = NULL) +
  scale_fill_manual(values = 'lightblue', labels = '95%␣posterior␣interval') +
  theme(legend.position = 'bottom', legend.title = element_blank())

# p2
a <- 49
```

```
b <- 3
# dbeta computes the posterior density
df1$p <- dbeta(df1$theta, a, b)
# seq creates evenly spaced values from 2.5% quantile
# to 97.5% quantile (i.e., 95% central interval)
# qbeta computes the value for a given quantile given parameters a and b
df2 <- data.frame(theta = seq(qbeta(0.025, a, b), qbeta(0.975, a, b), length.ou
# compute the posterior density
df2$p <- dbeta(df2$theta, a, b)
p2 <- ggplot(mapping = aes(theta, p)) +
  geom_line(data = df1) +
  # Add a layer of colorized 95% posterior interval
  geom_area(data = df2, aes(fill='1')) +
  # Add the proportion of girl babies in general population
  geom_vline(xintercept = (a-1)/(a-1+b-1), linetype='longdash') +
  geom_vline(xintercept = a/(a+b), , linetype='longdash', color='blue') +
  # Decorate the plot a little
  labs(title='Posterior B: Beta(49,51)', y = '') +
  scale_y_continuous(expand = c(0, 0.1), breaks = NULL) +
  scale_fill_manual(values = 'lightblue', labels = '95% posterior interval') +
  theme(legend.position = 'bottom', legend.title = element_blank())

# p3
a <- 187
b <- 15
# dbeta computes the posterior density
df1$p <- dbeta(df1$theta, a, b)
# seq creates evenly spaced values from 2.5% quantile
# to 97.5% quantile (i.e., 95% central interval)
# qbeta computes the value for a given quantile given parameters a and b
df2 <- data.frame(theta = seq(qbeta(0.025, a, b), qbeta(0.975, a, b), length.ou
# compute the posterior density
df2$p <- dbeta(df2$theta, a, b)
p3 <- ggplot(mapping = aes(theta, p)) +
  geom_line(data = df1) +
  # Add a layer of colorized 95% posterior interval
  geom_area(data = df2, aes(fill='1')) +
  # Add the proportion of girl babies in general population
  geom_vline(xintercept = (a-1)/(a-1+b-1), linetype='longdash') +
  geom_vline(xintercept = a/(a+b), , linetype='longdash', color='blue') +
  # Decorate the plot a little
  labs(title='Posterior C: Beta(187,201)', y = '') +
  scale_y_continuous(expand = c(0, 0.1), breaks = NULL) +
  scale_fill_manual(values = 'lightblue', labels = '95% posterior interval') +
  theme(legend.position = 'bottom', legend.title = element_blank())

(p1+p2+p3)
```

B.2 Monte Carlo integration

```
h=function(x){(cos(50*x)+sin(20*x))^2}

integrate(h,0,1)

x=h(runif(10^4))
estint=cumsum(x)/(1:10^4)
esterr=sqrt(cumsum((x-estint)^2))/(1:10^4)
plot(estint, xlab='n', ylab="I", type = "l", lwd=2, ylim = c(0.5,1.4))
lines(estint+3*esterr, col=4, lwd=1)
lines(estint-3*esterr, col=4, lwd=1)
```

B.3 uniform simulation: LCG

```
# LCG
lcg.rand <- function(n=10000) {
  rng <- vector(length = n)
  m <- 2 ** 32
  a <- 1103515245
  c <- 12345
  # Set the seed using the current system time in microseconds
  d <- as.numeric(Sys.time()) * 1000
  for (i in 1:n) {
    d <- (a * d + c) %% m
    rng[i] <- d / m
  }
  return(rng)
}
u <- lcg.rand()
par(mfrow=c(1,3))
hist(u, freq = F)
plot(u[seq(1,9999)], u[seq(2,10000)])
acf(u)

# Mersenne Twister
u2 <- runif(10000)
hist(u2, freq = F)
plot(u2[seq(1,9999)], u2[seq(2,10000)])
acf(u2)
```

B.4 inverse cdf method

```
# continuous Exp(1)
Nsim=10^4
U=runif(Nsim)
```

```
X=-log(1-U) # inverse transform of exponential from uniforms
Y=rexp(Nsim) # random exponentials from R
par(mfrow=c(1,2))
hist(X, freq = F, breaks=20,main = "Inverse␣cdf␣transform␣from␣uniform")
lines(density(X), lty=1,lwd=2,col=2)
hist(Y, freq = F, breaks=20,main = "Directly␣from␣R")
lines(density(Y), lty=1,lwd=2,col=2)
```

B.5 normal generator

```
# Normal Generator
u1=runif(10^4)
u2=runif(10^4)
x1=sqrt(-2*log(u1))*cos(2*pi*u2)
x2=sqrt(-2*log(u1))*sin(2*pi*u2)
mean(x1)
var(x1)
mean(x2)
var(x2)
par(mfrow=c(1,2))
hist(x1, freq = F)
lines(density(x1),lwd=2,col=2)
hist(x2, freq = F)
lines(density(x2),lwd=2,col=2)
cor(x1,x2) # For normal distributed variables, uncorrelated implies independent
# composition
Nsim=10^4
n=6; p=0.3
y=rgamma(Nsim,n, rate = p/(1-p))
x=rpois(Nsim,y)
hist(x, freq = F, col = "grey", breaks = 40)
lines(density(x),lwd=2,col=2) # kernel density of x
lines(1:50,dnbinom(1:50,n,p),lwd=2,col=3) # density of nbin
legend("topright",c("composition␣method","␣rnbinom()␣in␣R"),lty=c(1,1),col=c(2,
```

B.6 accept-reject method

```
# demo for accept-reject
x <- seq(-5,5,length.out = 1000)
p <- exp(-x^2/5)*(3*cos(x)^2*sin(4*x)^2+2*sin(6+x)^2)
q <- 1/(1.6*sqrt(2*pi))*exp(-0.5*(x/1.6)^2)
q <- max(p/q)*q
plot(x,p,col='red',type = 'l',ylab = 'f')
lines(x,q,col='blue')
tt1 <- expression(tilde(pi))
tt2 <- expression(tilde(M)*tilde(q))
legend('topright',legend = c(tt1,tt2),lty=c(1,1),col = c('red','blue'))
```

```
# sim beta(2.7,6.3)
Nsim=1000
a=2.7; b=6.3
M=optimise(f=function(x){dbeta(x,2.7,6.3)},interval=c(0,1),maximum=T)$objective
u=runif(Nsim,max = M) # U[0,M]
y=runif(Nsim) # proposal
x=y[u<dbeta(y,a,b)] # accepted subsample
z=seq(0,1,length.out = Nsim)
par(mfrow=c(2,2))
plot(y,u,xlab = "y",ylab = "u",main = "Uniform␣proposal")
lines(z,M*dunif(z),lwd=2,col=4)
plot(x,u[u<dbeta(y,a,b)],xlab = "y",ylab = "u",main = "Accepted␣samples")
lines(z,dbeta(z,a,b),lwd=2,col=2)

Nsim=1000
a=2.7; b=6.3
M=optimise(f=function(x){dbeta(x,2.7,6.3)/dbeta(x,2,6)},interval=c(0,1),maximum
y=rbeta(Nsim,2,6) # proposal
u=runif(Nsim,max = M*2) # U[0,M]
z=seq(0,1,length.out = Nsim)
x1=y[u<dbeta(y,a,b)] # accepted subsample
u1=u[u<dbeta(y,a,b)]
x2=y[u<dbeta(y,2,6)] # proposal
u2=u[u<dbeta(y,2,6)]

plot(x2,1.1*u2,xlim = c(0,1),ylim=c(0,3.2),xlab = "y",ylab = "u",main = 'Beta(2
lines(z,0.68*M*dbeta(z,2,6),col=4,lwd=2)
plot(x1,u1,xlim = c(0,1),ylim = c(0,3.2),xlab = "y",ylab = "u",main = 'Accepted
lines(z,dbeta(z,a,b),col=2,lwd=2)
```

B.7 importance sampler

```
Nsim=10^3
y1=rnorm(Nsim)
z1=y1>4.5

y=rexp(Nsim)+4.5
weit=dnorm(y)/dexp(y-4.5)
plot(cumsum(weit)/(1:Nsim), type="l",xlab = "n", ylab="estimate")
lines(cumsum(z1)/(1:Nsim),col=4)
abline(a=pnorm(-4.5),b=0,col=2,lwd=2,lty=2)
legend('topright',c('importance␣sampling','naive␣Monte␣Carlo','true␣value'),lty
```

B.8 demonstration of MCMC idea

```
par(mfrow=c(2,1))
```

```
x <- seq(-5,5,length.out = 1000)
plot(x,dnorm(x,0,0.8),type = 'l',col=2,lwd=2,ylab="")
lines(x,dnorm(x,-2,1),col=4,lwd=2)
abline(v=-2,col=3,lwd=2,lty=2)
abline(v=-0.5,col=3,lwd=2,lty=2)
legend('topright',c('target',expression(q(x[n]~"|"~x[n-1]))),lty=c(1,1),col=c(2
mtext(expression(x[n-1]),side = 1,at=-2,line = 0,col=3,cex = 1.2)
mtext(expression(x[n]),side = 1,at=-0.5,line = 0,col=3,cex = 1.2)

plot(x,dnorm(x,0,0.8),type = 'l',col=2,lwd=2,ylab="")
lines(x,dnorm(x,-0.5,1),col=4,lwd=2)
legend('topright',c('target',expression(q(x[n+1]~"|"~x[n]))),lty=c(1,1),col=c(2
abline(v=-0.5,col=3,lwd=2,lty=2)
abline(v=0.2,col=3,lwd=2,lty=2)
mtext(expression(x[n]),side = 1,at=-0.5,line = 0,col=3,cex = 1.2)
mtext(expression(x[n+1]),side = 1,at=0.2,line = 0,col=3,cex = 1.2)
```

B.9 toy example of independent Metropolis-Hastings and random walk Metropolis

```
# independent sampler for beta(2.7,6.3)
Nsim=5000
ac=0
X=rep(runif(1),Nsim)
for (i in 2:Nsim) {
  Y=rbeta(1,2,6) # beta(2,6)
  rho=(dbeta(Y,2.7,6.3)*dbeta(X[i-1],2,6))/(dbeta(X[i-1],2.7,6.3)*dbeta(Y,2,6))
  if (runif(1)<rho){
    X[i]=Y
    ac=ac+1}
  else
    X[i]=X[i-1]
}
ac_rate=ac/Nsim

plot(X,type = "l",xlim = c(4500, 4700),
      main = "Trace plot in 200 Iterations",
      xlab = "iterations",ylab = "X")

hist(X,freq = F, main = "Histogram of simulated sample")
w <- seq(0,1,length.out = 1000)
lines(w,dbeta(w,2.7,6.3),col=2,lwd=2)
legend('topright','target',lty=1,col=2,lwd=2)

# random walk metropolis beta(2.7,6.3)
Nsim=5000
ac=0
X=rep(runif(1),Nsim)
```

```
for (i in 2:Nsim) {
  Y=X[1]+runif(1,-0.4,0.4) # unif()
  if (Y>0 & Y<1){
    rho=dbeta(Y,2.7,6.3)/dbeta(X[i-1],2.7,6.3)
    if (runif(1)<rho){
      X[i]=Y
      ac=ac+1
    }
    else
      X[i]=X[i-1]
  }
  else
    X[i]=X[i-1]
}
ac_rate=ac/Nsim

hist(X, freq = F, main = "Histogram␣of␣simulated␣sample", xlim = c(0,0.8))
w <- seq(0,1,length.out = 1000)
lines(w,dbeta(w,2.7,6.3),col=2,lwd=2)
legend('topright','target',lty=1,col=2,lwd=2)

plot(X,type = "l",xlim = c(4500, 4700),
     main = "Trace␣plot␣in␣200␣Iterations",
     xlab = "iterations",ylab = "X")
```

B.10 toy example of Gibbs sampler

```
Nsim=5000 #initial values
n=15
a=3
b=7

X=T=array(0,dim=c(Nsim,1)) #init arrays
T[1]=rbeta(1,a,b) #init chains
X[1]=rbinom(1,n,T[1])
for (i in 2:Nsim){ #sampling loop
  X[i]=rbinom(1,n,T[i-1])
  T[i]=rbeta(1,a+X[i],n-X[i]+b)}

par(mar=c(5.1, 4.1, 4.1, 2.1), mgp=c(3, 1, 0), las=0)
par(mfrow=c(1,2))
hist(T,freq = F,main = 'Histogram␣of'~theta,xlab = expression(theta),ylab = 'm
x1 <- seq(0,1,length.out = 1000)
lines(x1,dbeta(x1,3,7),col=2,lwd=2)
hist(X,freq = F,main = 'Histogram␣of'~X,ylab = 'marginal␣density')
x2 <- seq(0.1,14.9,length.out = 1000)
x3 <- gamma(16)*gamma(10)*gamma(x2+3)*gamma(22-x2)/(gamma(3)*gamma(7)*gamma(x2+
lines(x2,x3,col=2,lwd=2)
```

```
par(mfrow=c(2,1))
par(mar = c(4, 4, 1, 2))
plot(T,type = "l",xlim = c(4500, 4700),
     main = "",
     xlab = "iterations",ylab = "␣"~theta)
par(mar = c(4, 4, 1, 2))
plot(X,type = "l",xlim = c(4500, 4700),
     main = "",
     xlab = "iterations",ylab = "X")
```

B.11 normal model via independent Metropolis-Hastings

```
rm(list = ls())
setwd(dir = "C:/Users/asus/Desktop/RData")

x=scan("normal.txt")
Nsim=5000
m=mean(x)
  s_hat=sd(x)
  n=length(x)
  s=2*s_hat/sqrt(n) # twice the std error, ensures it is larger than var(mu)
  alpha=sqrt(n) # ensures the proposal for tau has a larger variance than var(t
  beta=alpha*s_hat^2

indepMH=function(x,Nsim){
  mu=rep(rnorm(1,m,sd=s),Nsim) # init mu
  tau=rep(rgamma(1,alpha,beta),Nsim) # init tau
  mu[1]=3
  tau[1]=0.25
  ac=0
  for (t in 2:Nsim){
    y_mu=rnorm(1, mean=m, sd=s) # proposal for mu
    y_tau=rgamma(1,alpha,beta) # proposal for tau
    l_y=sqrt(y_tau/(2*pi))^n*exp(-y_tau/2*(sum(x^2)-2*y_mu*sum(x)+n*y_mu^2)) #
    l_x=sqrt(tau[t-1]/(2*pi))^n*exp(-tau[t-1]/2*(sum(x^2)-2*mu[t-1]*sum(x)+n*mu
    num=l_y*dnorm(y_mu)*dexp(y_tau,1)*dnorm(mu[t-1],m,s)*dgamma(tau[t-1],alpha,
    deno=l_x*dnorm(mu[t-1])*dexp(tau[t-1],1)*dnorm(y_mu,m,s)*dgamma(y_tau,alpha
    rho=num/deno
    if (runif(1)<rho){
      mu[t]=y_mu
      tau[t]=y_tau
      ac=ac+1
    }
    else{
      mu[t]=mu[t-1]
      tau[t]=tau[t-1]
    }
  }
  ac_rate=ac/Nsim
  out=list(mu,tau,ac_rate)
  return(out)
}

output <- indepMH(x,Nsim)
mu <- output[[1]]
tau <- output[[2]]
ac <- output[[3]]

mean(mu) # true mu is 3
mean(tau) # true tau is 0.25

layout(mat = matrix(c(1, 2, 3, 4), nrow = 2, ncol = 2),
       heights = c(2, 2),     # Heights of the two rows
       widths = c(2, 1))      # Widths of the two columns

par(mar = c(4, 4, 1, 2))
ts.plot(mu,xlab='iterations')
par(mar = c(4, 4, 1, 2))
ts.plot(tau,xlab='iterations')
par(mar = c(4, 2, 1, 1))
acf_mu=acf(mu,plot = F)
plot(acf_mu,main = '')
par(mar = c(4, 2, 1, 1))
acf_tau=acf(tau,plot = F)
plot(acf_tau,main = '')

par(mfrow=c(1,2))
hist(mu,main = 'mu',freq = F)
w=seq(0,4,length.out = 1000)

hist(tau,main = 'tau',freq = F)
w=seq(0,4,length.out = 1000)
```

B.12 normal model via random walk Metropolis

```
x = scan("normal.txt")
Nsim = 5000
sd_mu = 0.5 # sd for propsal of mu
sd_tau = 0.05 # sd for proposal of tau

RWM <- function(x,Nsim,sd_mu,sd_tau){
  n <- length(x)
  mu = rep(0,Nsim) # init mu
  tau = rep(0,Nsim) # init tau
  mu[1] = mean(x)
  tau[1] = 1/var(x)
  ac = 0
  for (i in 2:Nsim) {
    y_mu <- mu[i-1]+rnorm(1,0,sd_mu)
    y_tau <- tau[i-1]+rnorm(1,0,sd_tau)
    u <- runif(1)
    if (y_tau > 0){
      l_y = prod(dnorm(x,y_mu,sqrt(1/y_tau)))
      l_x = prod(dnorm(x,mu[i-1],sqrt(1/tau[i-1])))
      py <- l_y*dnorm(y_mu,3,1)*dexp(y_tau) # prior
      px <- l_x*dnorm(mu[i-1],3,1)*dexp(tau[i-1])
      if (u < min(py/px,1)){
        mu[i] = y_mu
        tau[i] = y_tau
        ac = ac + 1
      }
      else{
        mu[i] <- mu[i-1]
        tau[i] <- tau[i-1]
      }
    }
    else{
      mu[i] <- mu[i-1]
      tau[i] <- tau[i-1]
    }
  }
  ac_rate=ac/Nsim

  return(list(mu,tau,ac_rate))
}
output <- RWM(x,Nsim,sd_mu,sd_tau)
mu <- output[[1]]
tau <- output[[2]]
ac <- output[[3]]

mean(mu)
mean(tau)
layout(mat = matrix(c(1, 2, 3, 4), nrow = 2, ncol = 2),
       heights = c(2, 2),      # Heights of the two rows
       widths = c(2, 1))       # Widths of the two columns

par(mar = c(4, 4, 1, 2))
ts.plot(mu,xlab='iterations')
par(mar = c(4, 4, 1, 2))
ts.plot(tau,xlab='iterations')
par(mar = c(4, 2, 1, 1))
acf_mu=acf(mu,plot = F)
plot(acf_mu,main = '')
par(mar = c(4, 2, 1, 1))
```

```
acf_tau=acf(tau,plot = F)
plot(acf_tau,main = '')

par(mar=c(5.1, 4.1, 4.1, 2.1), mgp=c(3, 1, 0), las=0)
par(mfrow=c(1,2))
hist(mu,main = 'mu',freq = F)
w=seq(0,4,length.out = 1000)

hist(tau,main = 'tau',freq = F)
w=seq(0,4,length.out = 1000)
```

B.13 normal model via Gibbs sampler

```
x=scan("normal.txt")
Nsim=5000
n <- length(x)[1] # sample size

a=3 # alpha_0
b=7 # beta_0
l=3 # lambda_0
om=1 # omega_0
scale=(n/2)+a # scale parameter of posterior conditional distribution for tau

tau = mu = rep(0,Nsim)
tau[1] = rgamma(1,shape=a,rate=b)
var=1/(n*tau[1]+om) # variance of posterior condition distribution for mu
mu[1]=rnorm(1,(tau[1]*sum(x)+l*om)/(n*tau[1]+om),sqrt(var))
for (i in 2:Nsim){
   rate=b+0.5*sum((x-mu[i-1])^2) # rate parameter of posterior conditional dist
   tau[i]=rgamma(1,scale,rate)
   var=1/(n*tau[i]+om)
   mu[i]=rnorm(1,(tau[i]*sum(x)+l*om)/(n*tau[i]+om),sqrt(var))
   }
mean(mu)
mean(tau)

par(mfrow=c(1,2))
hist(mu,freq = F)
hist(tau,freq = F)

layout(mat = matrix(c(1, 2, 3, 4), nrow = 2, ncol = 2),
        heights = c(2, 2),     # Heights of the two rows
        widths = c(2, 1))      # Widths of the two columns

par(mar = c(4, 4, 1, 2))
ts.plot(mu,xlab='iterations')
par(mar = c(4, 4, 1, 2))
ts.plot(tau,xlab='iterations')
```

```
par(mar = c(4, 2, 1, 1))
acf_mu=acf(mu, plot = F)
plot(acf_mu,main = '')
par(mar = c(4, 2, 1, 1))
acf_tau=acf(tau,plot = F)
plot(acf_tau,main = '')
```

B.14 mixture model via independent Metropolis-Hastings

```
# generate mixture data x: 0.7*N(-1,1)+0.3*N(2,1)
n <- 100
u <- rbinom(n,1,0.7)
x <- vector()
for (i in 1:n) {
  x[i] <- rnorm(1,-1,1)^u[i]*rnorm(1,2,1)^(1-u[i])
}
dx <- seq(-5,5,length.out = 1000)
plot(dx,0.7*dnorm(dx,-1,1)+0.3*dnorm(dx,2,1),main = 'Gaussian mixture 0.7N(-1,1
lines(dx,0.7*dnorm(dx,-1,1),col=2,lwd=2)
lines(dx,0.3*dnorm(dx,2,1),col=4,lwd=2)

# number of simulations
Nsim <- 5000

# parameters for proposals
n=length(x)
s=sd(x)/sqrt(n) # std error of mixture, ensured to have larger variance than mu
alpha1=2
alpha2=2 # ensures the proposals have a larger variance than tau1 and tau2
beta1=2
beta2=2
a=7
b=3
prop <- c(-1,s,2,s,alpha1,beta1,alpha2,beta2,a,b)

# independent Metropolis for mixture model
indepMH=function(x,prop,Nsim){
  # initialize
  mu1=numeric(Nsim)
  mu2=numeric(Nsim)
  tau1=numeric(Nsim)
  tau2=numeric(Nsim)
  p=numeric(Nsim)
  # starting point of the chain
  mu1[1]=-1
  mu2[1]=2
  tau1[1]=1
  tau2[1]=1
  p[1]=0.5
  ac=0
  # independent Metropolis algorithm
  for (t in 2:Nsim){
    # proposals for mu1,mu2,tau1,tau2,p
    y_mu1=rnorm(1, mean=prop[1], sd=prop[2])
    y_mu2=rnorm(1, mean=prop[3], sd=prop[4])
    y_tau1=rgamma(1,prop[5],prop[6])
    y_tau2=rgamma(1,prop[7],prop[8])
    y_p=rbeta(1,prop[9],prop[10])
    # likelihood
    l_y=prod(y_p*sqrt(y_tau1/(2*pi))*exp(-0.5*y_tau1*(x-y_mu1)^2)
             +(1-y_p)*sqrt(y_tau2/(2*pi))*exp(-0.5*y_tau2*(x-y_mu2)^2))
    l_x=prod(y_p*sqrt(tau1[t-1]/(2*pi))*exp(-0.5*tau1[t-1]*(x-mu1[t-1])^2)
             +(1-y_p)*sqrt(tau2[t-1]/(2*pi))*exp(-0.5*tau2[t-1]*(x-mu2[t-1])^2)
    # prior: mu1~N(-1,4), mu2~N(2,4), tau1~exp(1), tau2~exp(1), p~beta(3,2)
    prior_y=dnorm(y_mu1,-1,sd=2)*dnorm(y_mu2,2,sd=2)*dexp(y_tau1,1)*dexp(y_tau2
    prior_x=dnorm(mu1[t-1],-1,sd=2)*dnorm(mu2[t-1],2,sd=2)*dexp(tau1[t-1],1)*de
    q_y=dnorm(y_mu1,prop[1],prop[2])*dnorm(y_mu2,prop[3],prop[4])*dgamma(y_tau1
    q_x=dnorm(mu1[t-1],prop[1],prop[2])*dnorm(mu2[t-1],prop[3],prop[4])*dgamma(
    rho=(prior_y*l_y*q_x)/(prior_x*l_x*q_y)
    if (runif(1)<rho){
      mu1[t]=y_mu1
      mu2[t]=y_mu2
      tau1[t]=y_tau1
      tau2[t]=y_tau2
      p[t]=y_p
      ac=ac+1
    }
    else{
      mu1[t]=mu1[t-1]
      mu2[t]=mu2[t-1]
      tau1[t]=tau1[t-1]
      tau2[t]=tau2[t-1]
      p[t]=p[t-1]
    }
  }
  ac_rate=ac/Nsim
  out=list(mu1,mu2,tau1,tau2,p,ac_rate)
  return(out)
}

output <- indepMH(x,prop,Nsim)
mu1 <- output[[1]]
mu2 <- output[[2]]
tau1 <- output[[3]]
tau2 <- output[[4]]
p <- output[[5]]
ac_rate <- output[[6]]
```

```
mean(mu1)
mean(tau1)
mean(mu2)
mean(tau2)
mean(p)

layout(mat = matrix(c(1, 2, 3, 4,5,6,7,8,9,10), nrow = 5, ncol = 2),
       heights = c(2, 2, 2,2,2),    # Heights of the two rows
       widths = c(2, 1))      # Widths of the two columns

par(mar = c(4, 4, 1, 1))
ts.plot(mu1,xlab='iterations')
par(mar = c(4, 4, 1, 1))
ts.plot(tau1,xlab='iterations')
par(mar = c(4, 4, 1, 1))
ts.plot(mu2,xlab='iterations')
par(mar = c(4, 4, 1, 1))
ts.plot(tau2,xlab='iterations')
par(mar = c(4, 4, 1, 1))
ts.plot(p,xlab='iterations')

par(mar = c(4, 2, 1, 1))
acf_mu1=acf(mu1,plot = F)
plot(acf_mu1,main = '')
par(mar = c(4, 2, 1, 1))
acf_tau1=acf(tau1,plot = F)
plot(acf_tau1,main = '')
par(mar = c(4, 2, 1, 1))
acf_mu2=acf(mu2,plot = F)
plot(acf_mu2,main = '')
par(mar = c(4, 2, 1, 1))
acf_tau2=acf(tau2,plot = F)
plot(acf_tau2,main = '')
par(mar = c(4, 2, 1, 1))
acf_p=acf(p,plot = F)
plot(acf_p,main = '')
```

B.15 mixture model via random walk Metropolis

```
n <- 100
u <- rbinom(n,1,0.7)
x <- vector()
for (i in 1:n) {
  x[i] <- rnorm(1,-1,1)^u[i]*rnorm(1,2,1)^(1-u[i])
}
```

```
# Random walk Metropolis
Nsim <- 5000
sd_mu=1
sd_tau=0.05
sd_p=0.1

RMH=function(x,Nsim,sd_mu,sd_tau,sd_p){
  n=length(x)
  # init
  mu1 = rep(0,Nsim)
  mu2 = rep(0,Nsim)
  tau1 = rep(0,Nsim)
  tau2 = rep(0,Nsim)
  p = rep(0,Nsim)
  ac=0

  mu1[1]=-1
  mu2[1]=2
  tau1[1]=1
  tau2[1]=1
  p[1]=0.7

  for (i in 2:Nsim){
    y_mu1 <- mu1[i-1]+rnorm(1,0,sd_mu)
    y_mu2 <- mu2[i-1]+rnorm(1,0,sd_mu)
    y_tau1 <- tau1[i-1]+rnorm(1,0,sd_tau)
    y_tau2 <- tau2[i-1]+rnorm(1,0,sd_tau)
    y_p=p[i-1]+rnorm(1,0,sd_p) # proposal for p
    u <- runif(1)
    if (y_tau1 > 0 & y_tau2 > 0 & y_p>0 & y_p<1){
      l_y=prod(y_p*sqrt(y_tau1/(2*pi))*exp(-0.5*y_tau1*(x-y_mu1)^2)+(1-y_p)*sqr
      l_x=prod(y_p*sqrt(tau1[i-1]/(2*pi))*exp(-0.5*tau1[i-1]*(x-mu1[i-1])^2)+(1
      py <- l_y*dnorm(y_mu1,-1,2)*dnorm(y_mu2,2,2)*dexp(y_tau1)*dexp(y_tau2)*db
      px <- l_x*dnorm(mu1[i-1],-1,2)*dnorm(mu2[i-1],2,2)*dexp(tau1[i-1])*dexp(t
      if (u < min(py/px,1)){
        mu1[i] = y_mu1
        mu2[i] = y_mu2
        tau1[i] = y_tau1
        tau2[i] = y_tau2
        p[i] = y_p
        ac = ac + 1
      }
      else{
        mu1[i] <- mu1[i-1]
        mu2[i] <- mu2[i-1]
        tau1[i] <- tau1[i-1]
        tau2[i] <- tau2[i-1]
        p[i] <- p[i-1]
      }
    }
    else{
      mu1[i] <- mu1[i-1]
      mu2[i] <- mu2[i-1]
      tau1[i] <- tau1[i-1]
      tau2[i] <- tau2[i-1]
      p[i] <- p[i-1]
    }
  }
  ac_rate=ac/Nsim
  out=list(mu1,mu2,tau1,tau2,p,ac_rate)
  return(out)
}

out <- RMH(x,Nsim = 5000,sd_mu=0.25,sd_tau=0.25,sd_p=0.05)
mu1 <- out[[1]]
mu2 <- out[[2]]
tau1 <- out[[3]]
tau2 <- out[[4]]
p <- out[[5]]
ac_rate <- out[[6]]

mean(mu1)
mean(tau1)
mean(mu2)
mean(tau2)
mean(p)

layout(mat = matrix(c(1, 2, 3, 4,5,6,7,8,9,10), nrow = 5, ncol = 2),
       heights = c(2, 2, 2,2,2),    # Heights of the two rows
       widths = c(2, 1))     # Widths of the two columns

par(mar = c(4, 4, 1, 1))
ts.plot(mu1,xlab='iterations')
par(mar = c(4, 4, 1, 1))
ts.plot(tau1,xlab='iterations')
par(mar = c(4, 4, 1, 1))
ts.plot(mu2,xlab='iterations')
par(mar = c(4, 4, 1, 1))
ts.plot(tau2,xlab='iterations')
par(mar = c(4, 4, 1, 1))
ts.plot(p,xlab='iterations')

par(mar = c(4, 2, 1, 1))
acf_mu1=acf(mu1,plot = F)
plot(acf_mu1,main = '')
par(mar = c(4, 2, 1, 1))
acf_tau1=acf(tau1,plot = F)
plot(acf_tau1,main = '')
par(mar = c(4, 2, 1, 1))
acf_mu2=acf(mu2,plot = F)
```

```
plot(acf_mu2,main = '')
par(mar = c(4, 2, 1, 1))
acf_tau2=acf(tau2,plot = F)
plot(acf_tau2,main = '')
par(mar = c(4, 2, 1, 1))
acf_p=acf(p,plot = F)
plot(acf_p,main = '')

mean(mu1)
mean(tau1)
mean(mu2)
mean(tau2)
mean(p)
ac_rate
```

B.16 mixture model via Gibbs sampler

```
# generate mixture data 0.7*N(-1,1)+0.3*N(2,1)
n <- 100
u <- rbinom(n,1,0.7)
x <- vector()
for (i in 1:n) {
  x[i] <- rnorm(1,-1,1)^u[i]*rnorm(1,2,1)^(1-u[i])
}
plot(density(x),ylim=c(0,0.3))
w <- seq(-5,5,length.out = 500)
dw1 <- dnorm(w,-1,1)
dw2 <- dnorm(w,2,1)
lines(w,0.7*dw1,col=2,lty=2)
lines(w,0.3*dw2,col=3,lty=2)
legend("topright",legend=c("simulated mixture","0.7*N(-1,1)","0.3*N(2,1)"),
                         col=c("black", 2, 3), lty=c(1,2,2))

Nsim <- 5000
pi <- rep(0,Nsim)
mu_1 <- rep(0,Nsim)
mu_2 <- rep(0,Nsim)
tau_1 <- rep(0,Nsim)
tau_2 <- rep(0,Nsim)
Z <- matrix(nrow = Nsim,ncol = n)

a <- 3; b <- 2
m1 <- -1; m2 <- 2
l1 <- 0.25; l2 <- 0.25
s1 <- 1
s2 <- 1
r1 <- 1
r2 <- 1
```

```
pi[1] <- rbeta(1,a,b)
mu_1[1] <- m1
mu_2[1] <- m2
tau_1[1] <- 1
tau_2[1] <- 1
Z[1,] <- rbinom(n,1,0.7)

for (i in 2:Nsim) {
  n1 <- sum(Z[i-1,])
  n2 <- n-n1
  pi[i] <- rbeta(1,a+n1,b+n2)

  mu_1[i] <- rnorm(1,(l1*m1+tau_1[i-1]*sum(x*Z[i-1,])
                      /(l1+n1*tau_1[i-1])),1/sqrt(l1+n1*tau_1[i-1]))
  mu_2[i] <- rnorm(1,(l2*m2+tau_2[i-1]*sum(x*(rep(1,n)-Z[i-1,]))
                      /(l2+n2*tau_2[i-1])),1/sqrt(l2+n2*tau_2[i-1]))
  scale_1 <- s1+n1/2
  rate_1 <- r1+0.5*sum(((x-mu_1[i])*Z[i-1,])^2)
  scale_2 <- s2+n2/2
  rate_2 <- r2+0.5*sum(((x-mu_2[i])*(1-Z[i-1,]))^2)
  tau_1[i]=rgamma(1,scale_1,rate_1)
  tau_2[i]=rgamma(1,scale_2,rate_2)
  p <- pi[i]*dnorm(x,mu_1[i],1/sqrt(tau_1[i]))/(pi[i]*dnorm(x,mu_1[i],1/sqrt(ta
  Z[i,] <- rbinom(n,1,p)
}

layout(mat = matrix(c(1, 2, 3, 4,5,6,7,8,9,10), nrow = 5, ncol = 2),
       heights = c(2, 2, 2,2,2),    # Heights of the two rows
       widths = c(2, 1))    # Widths of the two columns


par(mar = c(4, 4, 1, 1))
ts.plot(mu_1,xlab='iterations')
par(mar = c(4, 4, 1, 1))
ts.plot(tau_1,xlab='iterations')
par(mar = c(4, 4, 1, 1))
ts.plot(mu_2,xlab='iterations')
par(mar = c(4, 4, 1, 1))
ts.plot(tau_2,xlab='iterations')
par(mar = c(4, 4, 1, 1))
ts.plot(pi,xlab='iterations')

par(mar = c(4, 2, 1, 1))
acf_mu1=acf(mu_1,plot = F)
plot(acf_mu1,main = '')
par(mar = c(4, 2, 1, 1))
acf_tau1=acf(tau_1,plot = F)
plot(acf_tau1,main = '')
par(mar = c(4, 2, 1, 1))
```

```
acf_mu2=acf(mu_2,plot = F)
plot(acf_mu2,main = '')
par(mar = c(4, 2, 1, 1))
acf_tau2=acf(tau_2,plot = F)
plot(acf_tau2,main = '')
par(mar = c(4, 2, 1, 1))
acf_p=acf(pi,plot = F)
plot(acf_p,main = '')

mean(mu_1)
mean(tau_1)
mean(mu_2)
mean(tau_2)
mean(pi)
```

Bibliography

Andrieu, C., De Freitas, N., Doucet, A. and Jordan, M. I. (2003). An introduction to mcmc for machine learning, *Machine learning* **50**(1): 5–43.

Bagley, J. (2022). Markov processes lecture notes.

Blei, D. M., Kucukelbir, A. and McAuliffe, J. D. (2017). Variational inference: A review for statisticians, *Journal of the American statistical Association* **112**(518): 859–877.

Gelman, A., Carlin, J. B., Stern, H. S. and Rubin, D. B. (1995). *Bayesian data analysis*, Chapman and Hall/CRC.

Gilks, W. R., Richardson, S. and Spiegelhalter, D. (1995). *Markov chain Monte Carlo in practice*, CRC press.

Hastings, W. K. (1970). Monte carlo sampling methods using markov chains and their applications.

Lu, J. (2021). A survey on bayesian inference for gaussian mixture model, *arXiv preprint arXiv:2108.11753* .

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. and Teller, E. (1953). Equation of state calculations by fast computing machines, *The journal of chemical physics* **21**(6): 1087–1092.

Rebeschini, P. (2018). Advanced simulation methods lecture notes, *available on https://www.stats.ox.ac.uk/ rebeschi/teaching/AdvSim/18/index.html* .

Robert, C. P. and Casella, G. (2004). The metropolis—hastings algorithm, *Monte Carlo statistical methods*, Springer, pp. 267–320.

Robert, C. P., Casella, G. and Casella, G. (2010). *Introducing monte carlo methods with r*, Vol. 18, Springer.

Strimmer, K. (2021). Statistical methods lecture notes.

Tierney, L. (1994). Markov Chains for Exploring Posterior Distributions, *The Annals of Statistics* **22**(4): 1701 – 1728.

van de Schoot, R., Depaoli, S., King, R., Kramer, B., Märtens, K., Tadesse, M. G., Vannucci, M., Gelman, A., Veen, D., Willemsen, J. et al. (2021). Bayesian statistics and modelling, *Nature Reviews Methods Primers* **1**(1): 1–26.