PSEUDO-MARGINAL METROPOLIS-HASTINGS APPROACH AND ITS APPLICATION TO BAYESIAN COPULA MODEL

Xuebin Zheng

Supervisor: Associate Professor Josef Dick

Co-Supervisor: Dr. David Gunawan

School of Mathematics and Statistics

UNSW Australia

October 2016

Submitted in partial fulfilment of the requirements of the degree of

Master of Financial Mathematics

Plagiarism statement

I declare that this thesis is my own work, except where acknowledged, and has not been submitted for academic credit elsewhere.

I acknowledge that the assessor of this thesis may, for the purpose of assessing it:

- Reproduce it and provide a copy to another member of the University; and/or,
- Communicate a copy of it to a plagiarism checking service (which may then retain a copy of it on its database for the purpose of future plagiarism checking).

I certify that I have read and understood the University Rules in respect of Student Academic Misconduct, and am aware of any potential plagiarism penalties which may apply.

By signing this declaration I am agreeing to the statements and conditions above.

Signed: _____ Date: _____

Acknowledgements

First of all, I would like to express my great appreciations to my supervisor A/Prof. Josef Dick, who gives me the most valuable and constructive suggestions and helps during the entire process of this thesis study. Without his assistance, this thesis report would have never been accomplished.

Secondly, I am particularly grateful for the enthusiastic assistance and advice given by my co-supervisor Dr. David Gunawan. I would also wish to extend my thanks to my peers who are doing postgraduate courses with me in this year for their generous information sharing and discussion.

Finally, most importantly, none of this could have happened without my family. Here is my most especially thanks to my parents and my wife Chen Xu, for always being with me, taking care of me and giving me unconditional love to pass through every difficulty that I faced. Their encouragement and support enabled me to complete this thesis.

Abstract

Metropolis-Hastings algorithm is a universal technique used for sampling approximately from high-dimensional distribution which is only known up to a multiplicative constant. However, there are many statistical models, such as state space models and latent variable models whose likelihood functions are intractable or at least computationally demanding in Bayesian estimation. This fact makes the Metropolis-Hastings algorithm infeasible. In this thesis, we discuss the pseudo-marginal Metropolis-Hastings (PMMH) method which is initially proposed by Beaumont [2003]. The PMMH algorithm overcomes the problem of working with an intractable likelihood by performing Markov chain Monte Carlo simulation on an expanded state space and substituting an unbiased estimator of the likelihood to the true likelihood in the acceptance probability. Moreover, we apply the PMMH method to Bayesian estimation of high-dimensional Copula models with discrete and mixed margins where evaluation of each likelihood contribution requires computing 2^J terms (J is the number of discrete margins). The numerical experiments confirm the competence of the PMMH algorithm in this new application area.

Keywords: Intractable likelihood, Metropolis-Hastings algorithm, Markov chain Monte Carlo, Unbiased estimator, Copula model, Pseudo-marginal algorithm.

Contents

Chapter 1

Introduction

Markov chain Monte Carlo (MCMC) methods are a class of algorithms for sampling approximately from a probability distribution based on constructing a Markov chain that has the desired distribution as its invariant distribution. The idea of Markov chain Monte Carlo was first introduced by Metropolis et al. [1953] as a method for efficient simulation of the energy levels of atoms in a crystalline structure and was subsequently adopted by Hastings [1970] to focus upon statistical problems. Suppose we have an arbitrary distribution

$$\pi(\boldsymbol{\theta}), \quad \boldsymbol{\theta} \in E \subseteq \mathbb{R}^p,$$

which is known only up to some multiplicative constant and we aim to simulate samples from the target distribution $\pi(\cdot)$. If the target distribution is significantly complex so that we cannot sample from it directly, an indirect method for obtaining samples from $\pi(\cdot)$ is to construct a Markov chain with state space E and whose invariant distribution is $\pi(\cdot)$. If we run the chain for a sufficiently long time, the simulated values from the chain can be regarded as an approximate sample from the target distribution and used as a basis for summarising important characteristics of $\pi(\cdot)$. The standard approach to generate a Markov chain with an invariant distribution $\pi(\cdot)$ consists of using the Metropolis - Hastings algorithm where new values $\boldsymbol{\theta}^*$ are drawn from an arbitrary proposal distribution $q(\cdot \,|\, \cdot)$ and then to be accepted or rejected according to an acceptance probability

$$\alpha(\boldsymbol{\theta}, \boldsymbol{\theta}^*) = \min \left\{ 1, \, \frac{\pi(\boldsymbol{\theta}^* \,|\, \boldsymbol{x}) \, q(\boldsymbol{\theta} \,|\, \boldsymbol{\theta}^*)}{\pi(\boldsymbol{\theta} \,|\, \boldsymbol{x}) \, q(\boldsymbol{\theta}^* \,|\, \boldsymbol{\theta})} \right\}.$$

In Bayesian inference, we are interested in simulation from the posterior distribution $\pi(\boldsymbol{\theta} \mid \boldsymbol{x})$ which is calculated based on the following *Bayes' theorem*

$$\begin{aligned}\pi(\boldsymbol{\theta} \mid \boldsymbol{x}) &= \frac{p(\boldsymbol{x} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta})}{\int_{\boldsymbol{\Theta}} p(\boldsymbol{x} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta})\, \mathrm{d}\boldsymbol{\theta}} \\ &= p(\boldsymbol{x} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta}) / p(\boldsymbol{x}) \\ &\propto p(\boldsymbol{x} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta}),\end{aligned}$$

where $p(\boldsymbol{x} \mid \boldsymbol{\theta})$ denotes the likelihood function and $p(\boldsymbol{\theta})$ is the prior density of parameters. Thus, implementing the Metropolis - Hastings algorithm requires computing the acceptance ratio

$$\begin{aligned}\alpha(\boldsymbol{\theta}, \boldsymbol{\theta}^*) &= \min\left\{1,\ \frac{\pi(\boldsymbol{\theta}^* \mid \boldsymbol{x})\, q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^*)}{\pi(\boldsymbol{\theta} \mid \boldsymbol{x})\, q(\boldsymbol{\theta}^* \mid \boldsymbol{\theta})}\right\} \\ &= \min\left\{1,\ \frac{p(\boldsymbol{x} \mid \boldsymbol{\theta}^*)\, p(\boldsymbol{\theta}^*)\, q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^*)}{p(\boldsymbol{x} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta})\, q(\boldsymbol{\theta}^* \mid \boldsymbol{\theta})}\right\}.\end{aligned}$$

However, evaluation of the likelihood function $p(\boldsymbol{x} \mid \boldsymbol{\theta})$ is analytically or computationally intractable in many statistical models. These models include state space models and generalised linear mixed models (GLMM) where the likelihood function is present as a form of high-dimensional integral. This fact makes the Metropolis - Hastings algorithm infeasible and motivates the following pseudo - marginal Metropolis - Hastings (PMMH) approach.

A direct way to overcome the problem of working with an intractable likelihood is to construct an unbiased and non-negative estimator of the likelihood and use this estimate within an Markov chain Monte Carlo (MCMC) simulation on an expanded space which includes the auxiliary random variables used to obtain the likelihood estimator. This idea was initially proposed by Lin et al. [2000] in the Physics literature and Beaumont [2003] in the Statistics literature. Andrieu and Roberts [2009] named this method the pseudo - marginal Metropolis - Hastings (PMMH) algorithm and established some of its theoretical properties. The most appealing feature of the PMMH algorithm is that the invariant distribution of the generated Markov chain will remain exactly as it would be if the true likelihood had been available as long as the constructed estimator of the likelihood function is non-negative and unbiased. Therefore, the PMMH method is regarded as "exact approximation" to the standard Metropolis - Hastings algorithm.

In the literature, Andrieu et al. [2010] apply the PMMH approach to the state space models and use particle filter to unbiasedly estimate the intractable likelihood. In this thesis, we are interested in utilizing the PMMH algorithm to conduct Bayesian estimation for high-dimensional copula model with discrete margins. Estimation of copula model with discrete margins in high dimensions is computationally challenging, because evaluation of the associated likelihood function requires computing 2^J terms and each of which is also expensive to compute, where J is the number of discrete margins in the copula model. On the other hand, we found that the likelihood function can be estimated unbiasedly by a computationally relatively cheap approach. Pitt et al. [2006] propose a computationally efficient method to Bayesian estimation of Gaussian copula with discrete margins by augmenting latent variables to the model and generate these latent variables within an MCMC simulation. Since the number of latent variables has the same size as the data, this method still suffer from computational issues because it requires sampling nJ latent variables, where n is the number of observations. However, the PMMH algorithm has advantage in the cases with higher dimensions J and large size of data set n.

The rest of the thesis is structured as follows. In Chapter 2, we first provide an overview of Markov chain Monte Carlo methods and their implementation issues. In Chapter 3, we discuss the underlying mechanism and procedure of the PMMH algorithm. Moreover, we describe the additional implementation issue associated to this particular approach. Furthermore, in Chapter 4, we apply the PMMH algorithm to the Bayesian copula model with discrete and mixed margins. In this thesis, we focus on the Gaussian copula model but the application to other family of copula model is worthy of being further studied. The application of the PMMH algorithm to Bayesian estimation of Gaussian copula model is the main purpose of the thesis and the materials produced in Chapter 4 are the main originality works we contributed in this thesis. We provide some basic knowledge about Markov chain and copula models in Appendix A and Appendix C respectively, which are helpful for the reader to understand the thesis work. In Appendix B, we present a proof of theorem 2.2.1 which is fundamental to the standard Metropolis - Hastings (MH) algorithm and the proof fully explains why the MH algorithm works. Finally, all the MATLAB® programs we wrote for the numerical examples are attached in Appendix D in order to make the reader easier to conduct any further study.

Chapter 2

Review of Markov Chain Monte Carlo Methods

At the beginning of this chapter, we will discuss some important theorems which are essential to the Markov chain Monte Carlo methods. Then, the Metropolis-Hastings algorithm and the Gibbs sampler will be introduced. Finally, we will describe some implementation issues which might be encountered in practice.

2.1 Some Essential Theorems

Markov chain Monte Carlo allows us to sample approximately from any complex unnormalised high-dimensional distribution. This appealing feature enables a vast range of Bayesian models to be estimated with ease. In Bayesian statistics, once samples from the posterior density $\pi(\theta \,|\, \boldsymbol{x})$ are obtained, we can find the point estimate of the parameter θ. Since the mean of the posterior is the most popular point estimator $\hat{\theta}$, we state the following limit theorem.

Theorem 2.1.1 (Ergodic Theorem)**.** *If X_t is an ergodic Markov chain with invariant distribution $\pi(\cdot)$ and $g(\cdot)$ is a real valued function with $\int |g(x)|\, \pi(\mathrm{d}x) < \infty$, then*

$$\frac{1}{M} \sum_{t=1}^{M} g(X_t) \xrightarrow{M \to \infty} \int g(x) \pi(\mathrm{d}x) < \infty$$

for π-almost all X_0.

In contrast to some exact simulation schemes, including Inverse Transform Sampling and Acceptance-Rejection Method, MCMC generates dependent samples. However, Theorem 2.1.1 asserts that ergodic average of chain values converges to the expected value under the target density $\pi(\cdot)$ despite their dependence. The following theorem ensures convergence of the transition probabilities of the Markov chain X_t.

Theorem 2.1.2. *If X_t is an aperiodic and π-irreducible Markov chain with transition kernel $K(\cdot,\,\cdot)$ and invariant distribution $\pi(\cdot)$, then there is a convergence in total variation*

$$\left\|K^t(X_0,\,\cdot) - \pi(\cdot)\right\| \to 0$$

as $t \to \infty$, π-almost all X_0.

This theorem shows that the sample path of the Markov chain mimics a random sample from $\pi(\cdot)$ and the distribution of the realisation converges to $\pi(\cdot)$, that is,

$$X_t \xrightarrow[t\to\infty]{d} \pi(x).$$

The detailed balance condition is not necessary but sufficient for $\pi(\cdot)$ to be a invariant measure associated with the transition kernel $K(\cdot,\,\cdot)$, we introduce the following important theorem with proof.

Theorem 2.1.3. *If a Markov transition kernel $K(\cdot,\,\cdot)$ satisfies the detailed balance condition with respect to some probability measure $\pi(\cdot)$, then $\pi(\cdot)$ is the invariant measure of the Markov chain with transition kernel $K(\cdot,\,\cdot)$.*

Proof. Suppose the transition kernel $K(\cdot,\,\cdot)$ satisfies the detailed balance condition with respect to some probability measure $\pi(\cdot)$, we have

$$K(\theta,\theta')\pi(\theta) = K(\theta',\theta)\pi(\theta'), \qquad \forall\,\theta,\,\theta' \in E,$$

where E is the state space of the Markov chain. From the above, we can get

$$\begin{aligned}(\pi K)(\theta') &= \sum_{\theta\in E} \pi(\theta)K(\theta,\theta') \\ &= \sum_{\theta\in E} \pi(\theta')K(\theta',\theta) \\ &= \pi(\theta')\sum_{\theta\in E} K(\theta',\theta) \\ &= \pi(\theta'),\end{aligned}$$

which matches the definition of invariant distribution of the Markov chain with a transition kernel $K(\cdot,\,\cdot)$. This completes the proof. □

In summary, for establishing a Markov chain whose invariant distribution of successive iterates from the chain converges to the target distribution $\pi(\cdot)$ regardless of the starting values of the chain, we require the following properties to be held.

- The invariant distribution of the Markov transition kernel is $\pi(\cdot)$. In other words, the transition kernel $K(\cdot, \cdot)$ is reversible with respect to target distribution $\pi(\cdot)$.
- The Markov chain is aperiodic and irreducible.

In the following sections, we introduce two extremely popular and widely used MCMC algorithms. They are the Metropolis - Hastings algorithm and the Gibbs sampler.

2.2 Metropolis - Hastings Algorithm

The Metropolis - Hastings algorithm can be considered as a form of generalised rejection sampling, where values are drawn from an arbitrary proposal distribution and then to be accepted or rejected according to an acceptance probability. If the proposed value is accepted, then the process moves to the proposed state, otherwise it remains at the current state. Asymptotically, the process behaves as random observations from the target distribution. The most important aspect of the Metropolis - Hastings algorithm is that the approximating distribution is improved at each iteration of the simulation. The procedure of the algorithm is outlined below.

Algorithm 1 The Metropolis - Hastings Algorithm

Require: A suitable proposal distribution $q(\cdot \,|\, \cdot)$, the target density $\pi(\boldsymbol{\theta} \,|\, \boldsymbol{x})$ and set the length of MCMC run M.

Initialise $\boldsymbol{\theta}^{(i=1)}$ with any value within a support of $\pi(\boldsymbol{\theta} \,|\, \boldsymbol{x})$.

for $i = 2, 3, \ldots, M$ **do**

Generate a proposal $\boldsymbol{\theta}^*$ from $q(\boldsymbol{\theta}^* \,|\, \boldsymbol{\theta}^{(i-1)})$.

Accept proposal with the acceptance probability

$$\alpha(\boldsymbol{\theta}^{(i-1)}, \boldsymbol{\theta}^*) = \min\left\{1,\, \frac{\pi(\boldsymbol{\theta}^* \,|\, \boldsymbol{x})\, q(\boldsymbol{\theta}^{(i-1)} \,|\, \boldsymbol{\theta}^*)}{\pi(\boldsymbol{\theta}^{(i-1)} \,|\, \boldsymbol{x})\, q(\boldsymbol{\theta}^* \,|\, \boldsymbol{\theta}^{(i-1)})}\right\},$$

that is, we simulate U from the uniform distribution $\mathsf{U}(0,\, 1)$ and set $\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^*$ if $U < \alpha(\boldsymbol{\theta}^{(i-1)}, \boldsymbol{\theta}^*)$, otherwise we set $\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)}$.

Note that the normalisation constant of the target distribution does not contribute here, since it can be canceled out in the acceptance ratio. The following theorem confirms that Algorithm 1 generates a Markov chain $\left\{\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots, \boldsymbol{\theta}^{(M)}\right\}$ whose invariant distribution is the target density $\pi(\boldsymbol{\theta} \,|\, \boldsymbol{x})$. The proof of the theorem is provided in Appendix B.

Theorem 2.2.1. *The simulated sequence* $\left\{\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots, \boldsymbol{\theta}^{(M)}\right\}$ *from the Metropolis - Hastings algorithm is a Markov chain with the target distribution* $\pi(\boldsymbol{\theta} \,|\, \boldsymbol{x})$ *as its unique invariant distribution.*

The Metropolis - Hastings algorithm is a general approach to construct MCMC sampler, there are some common choices for the proposal distribution:

- If the proposal distribution is symmetric,

$$q(\boldsymbol{\theta}^* \,|\, \boldsymbol{\theta}) = q(\boldsymbol{\theta} \,|\, \boldsymbol{\theta}^*), \quad \forall\, \boldsymbol{\theta},\, \boldsymbol{\theta}^* \in E,$$

  the algorithm is so-called *the random-walk Metropolis sampler*. The frequently used choices for $q(\cdot \,|\, \cdot)$ include the Uniform distribution and Multivariate Normal distribution. In this case, the acceptance probability is of the form

$$\alpha(\boldsymbol{\theta}, \boldsymbol{\theta}^*) = \min\left\{1,\, \frac{\pi(\boldsymbol{\theta}^* \,|\, \boldsymbol{x})}{\pi(\boldsymbol{\theta} \,|\, \boldsymbol{x})}\right\}.$$

- The case where the proposal distribution is independent of the current state of the Markov chain,

$$q(\boldsymbol{\theta}^* \,|\, \boldsymbol{\theta}) = q(\boldsymbol{\theta}^*), \quad \forall\, \boldsymbol{\theta},\, \boldsymbol{\theta}^* \in E,$$

  leads to *the independence sampler*. Therefore, the acceptance probability can be written as

$$\alpha(\boldsymbol{\theta}, \boldsymbol{\theta}^*) = \min\left\{1,\, \frac{\pi(\boldsymbol{\theta}^* \,|\, \boldsymbol{x})\, q(\boldsymbol{\theta})}{\pi(\boldsymbol{\theta} \,|\, \boldsymbol{x})\, q(\boldsymbol{\theta}^*)}\right\}.$$

2.3 Gibbs Sampler

The Gibbs sampler is an extremely powerful technique for generating random samples from a high dimensional distribution without evaluating the density. The method was initially introduced by Geman and Geman [1984] to the image analysis models and subsequently to the statistics literature by Besag and York [1989].

However, the sampler requires simulation from all the conditional distributions of the target density. In the 2-dimensional case, we have a pair of random variables (X, Y) with joint density $\pi(x, y)$. Assume that the direct simulation from each of $\pi(x, y)$, $\pi(x)$ and $\pi(y)$ is not possible, then we can utilise Gibbs sampler if the conditional distributions $\pi_1(x \,|\, y)$ and $\pi_2(y \,|\, x)$ can be easily simulated. The following algorithm outlines the Gibbs sampler for k dimensions.

Algorithm 2 The Gibbs Sampler

Require: Conditional distributions of the target density $\pi(x_1, x_2, \ldots, x_k)$ and set the length of MCMC run M.

Initialise $X_1^{(i=1)}, X_2^{(i=1)}, \ldots, X_k^{(i=1)}$ with an arbitrary value.

for $i = 2, 3, \ldots, M$ **do**

Simulate $X_1^{(i)}$ from $\pi_1\left(x_1 \,|\, X_2^{(i-1)}, X_3^{(i-1)}, \ldots, X_k^{(i-1)}\right)$.

Simulate $X_2^{(i)}$ from $\pi_2\left(x_2 \,|\, X_1^{(i)}, X_3^{(i-1)}, \ldots, X_k^{(i-1)}\right)$.

$\vdots$

Simulate $X_k^{(i)}$ from $\pi_k\left(x_k \,|\, X_1^{(i)}, X_2^{(i)}, \ldots, X_{k-1}^{(i)}\right)$.

Under reasonably general conditions, the joint density $\pi(x_1, x_2, \ldots, x_k)$ is an invariant distribution of the generated chain $\left\{\boldsymbol{X}^{(1)}, \boldsymbol{X}^{(2)}, \ldots, \boldsymbol{X}^{(M)}\right\}$ where $\boldsymbol{X}^{(i)} = \left(X_1^{(i)}, X_2^{(i)}, \ldots, X_k^{(i)}\right)^\top$ and the distribution of $\left\{X_i^{(1)}, X_i^{(2)}, \ldots, X_i^{(M)}\right\}$ converges to $\pi(x_i)$ as $M \to \infty$. Hence, the generated chain can be used to approximate both of the joint distribution and the marginal distribution of each of the variables. Moreover, the Gibbs sampler can also be regarded as a special case of the Metropolis - Hastings algorithm whose proposal distribution is the conditional density of the target distribution,

$$q_i(\boldsymbol{x}^* \,|\, \boldsymbol{x}) = \pi_i(x_i^* \,|\, \boldsymbol{x}_{-i}), \quad \forall\, i = 1, 2, \ldots, k,$$

where $\boldsymbol{x}_{-i} = \{x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_k\}$.

Theorem 2.3.1. *The Gibbs updates are accepted with probability* 1.

Proof. In this case, the proposal distribution is the conditional density of the target distribution. So for $i = 1, 2, \ldots, k$, we have

$$\begin{aligned} q_i(\boldsymbol{x}^* \,|\, \boldsymbol{x}) &= \pi_i(x_i^* \,|\, \boldsymbol{x}_{-i}) \\ &= \frac{1}{c}\, \pi(x_1, x_2, \ldots, x_{i-1}, x_i^*, x_{i+1}, \ldots, x_k)\,, \end{aligned}$$

where c is the unknown normalizing constant that makes $\pi(\cdot)$ a proper conditional probability density. Thus, we have the following equalities

$$\begin{aligned}
\boldsymbol{x} &= \{x_1, x_2, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_k\} \\
\boldsymbol{x}_{-i} &= \{x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_k\} \\
\boldsymbol{x}^* &= \{x_1, x_2, \ldots, x_{i-1}, x_i^*, x_{i+1}, \ldots, x_k\} \\
\boldsymbol{x}^*_{-i} &= \{x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_k\} = \boldsymbol{x}_{-i}
\end{aligned}$$

The acceptance probability can be written as

$$\alpha(\boldsymbol{x}, \boldsymbol{x}^*) = \min\{1,\ M\},$$

where

$$\begin{aligned}
M &= \frac{\pi(\boldsymbol{x}^*)\, q_i(\boldsymbol{x} \mid \boldsymbol{x}^*)}{\pi(\boldsymbol{x})\, q_i(\boldsymbol{x}^* \mid \boldsymbol{x})} \\
&= \frac{\pi(x_1, x_2, \ldots, x_{i-1}, x_i^*, x_{i+1}, \ldots, x_k)\, \pi_i(x_i \mid \boldsymbol{x}^*_{-i})}{\pi(x_1, x_2, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_k)\, \pi_i(x_i^* \mid \boldsymbol{x}_{-i})} \\
&= \frac{\pi(x_1, x_2, \ldots, x_{i-1}, x_i^*, x_{i+1}, \ldots, x_k)\, \pi(x_1, x_2, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_k)/c}{\pi(x_1, x_2, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_k)\, \pi(x_1, x_2, \ldots, x_{i-1}, x_i^*, x_{i+1}, \ldots, x_k)/c} \\
&= 1.
\end{aligned}$$

Hence, the acceptance probability $\alpha(\boldsymbol{x}, \boldsymbol{x}^*) = 1$, which completes the proof. □

By applying the Gibbs sampler, all generated samples get accepted. This fact makes the sampler highly efficient in practice when the parameters are not strongly correlated with each other. However, the full conditional distributions for each parameter of the target density must be easy to sample from.

2.4 Implementation Issues

In this section, we discuss several practical issues when implementing the Markov chain Monte Carlo methods. These issues include tuning of the proposal distribution, assessment of mixing of the generated chain, Burn-in period and the numerical error of the obtained estimate.

2.4.1 Tuning the Proposal Distribution

The use of MCMC methods can be substantially inefficient if the proposal distribution is not properly chosen. Theoretically, a good proposal distribution should resemble the actual target distribution. In Bayesian inference, large sample theory states that as more and more data arrive, the posterior distribution of the parameter vector approaches multivariate normal (see Gelman et al. [2014, Appendix B]). Thus, a multivariate normal proposal density $q(\boldsymbol{\theta}^* \mid \boldsymbol{\theta}^{(i-1)}) = \mathsf{MVN}(\boldsymbol{\theta}^* \mid \boldsymbol{\theta}^{(i-1)}, c^2\,\Sigma)$ works well in the majority of situations.

The scale parameter c in the proposal distribution $q(\cdot \mid \cdot)$ effectively controls the acceptance probability (the fraction of proposed moves which are accepted) of the algorithm. For a random walk Metropolis sampler, the Markov chain will move slowly and not fully explore the parameter space if the acceptance rate is too high (corresponds to a small scale parameter c). On the other hand, if the acceptance probability is too low (amounts to a large scale parameter c), then the proposed values are often rejected and the chain will randomly get stuck at some states. The acceptance rate is closely related to the mixing of the generated chain. Slow mixing indicates that the chain should be run much longer in order to get a good estimate. In contrast, if a Markov chain is mixing rapidly, then the chain will converge to its invariant distribution quickly. The following figures demonstrate the results from different choices of the scale parameter c.

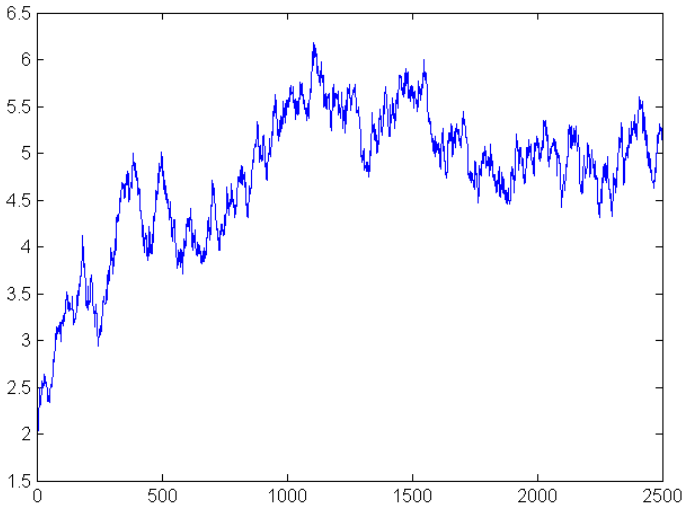


Figure 2.1: Trace plot with small c, large acceptance rate 0.96, poor mixing.

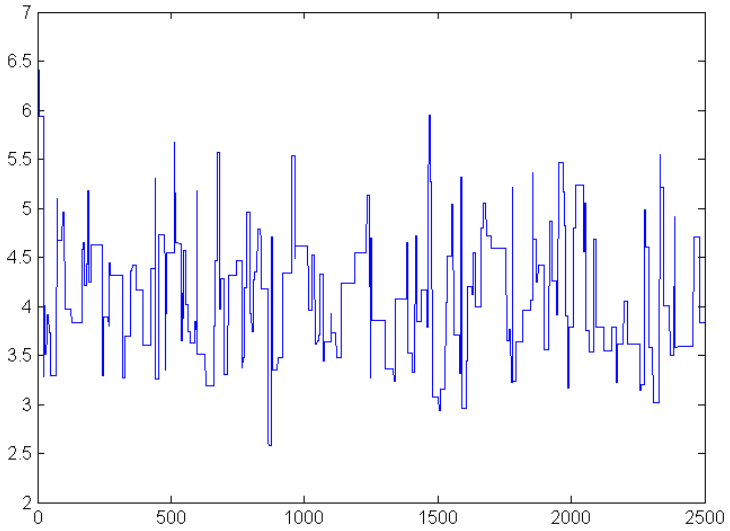


Figure 2.2: Trace plot with large c, small acceptance rate 0.08, poor mixing.

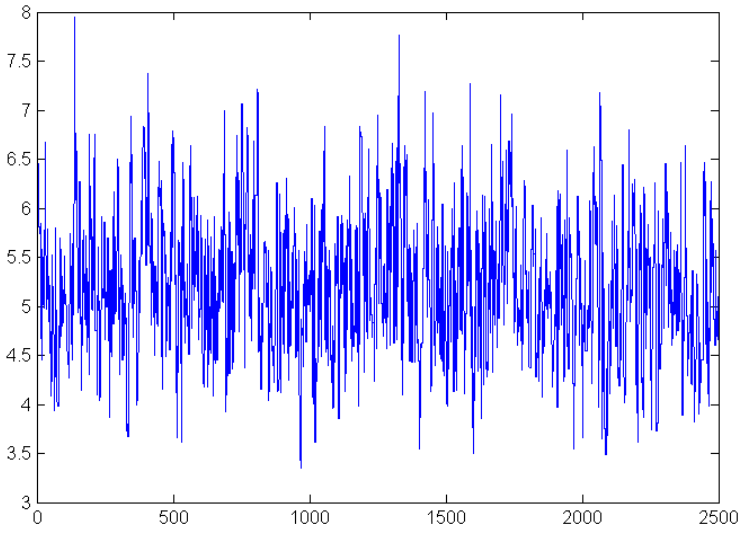


Figure 2.3: Trace plot with medium c, medium acceptance rate 0.67, good mixing.

The above figures suggest that the scale parameter c has to be tuned so that the generated chain achieves a reasonable acceptance rate. Roberts et al. [1997] showed that if both the target and proposal distributions are normal, then the optimal acceptance probability of the algorithm should be around 0.45 in a single dimensional problem, and asymptotically approaches 0.234 in higher dimensions. In practice, the chains with acceptance rate between 0.2 and 0.8 often work well. Recent studies

regarding the optimal scaling of Metropolis - Hastings algorithms include Roberts et al. [2001] and Garthwaite et al. [2015].

2.4.2 Burn-in Stage

In practice, the initial values of the Markov chain Monte Carlo methods are arbitrarily chosen. Thus, the starting point of the algorithm might be distant from the true value of the parameter. In this situation, the chain needs to spend some time to become stationary. To reduce the influence of the starting value, we normally discard the first portion of each generated sequence, which is called the *Burn-in Stage*. Visual inspection of the chain plot is the most widely used technique to determine the number of iterations to be discarded. The following figure provides an example of the Burn-in stage.

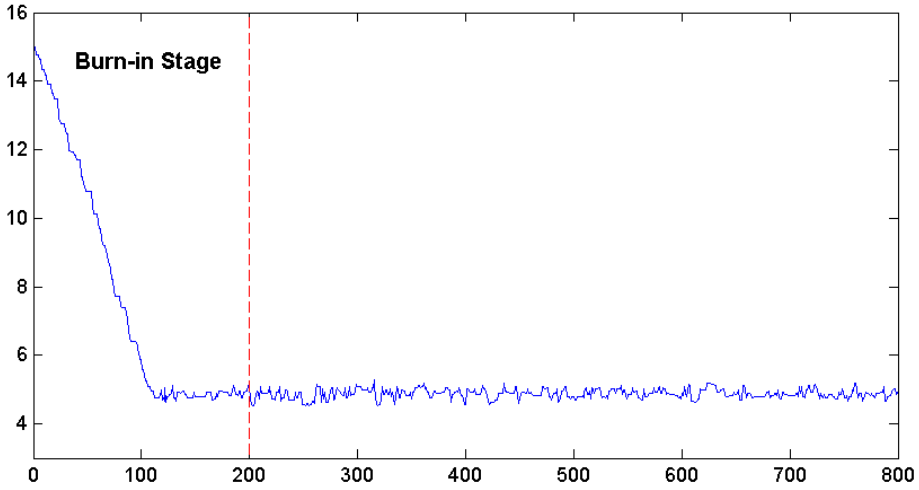


Figure 2.4: Trace plot of the generated chain.

2.4.3 Assessment of Mixing

The autocorrelation plot is a commonly used tool for checking the mixing of the generated chain. In well mixed Markov chain samples, the autocorrelation falls to near zero quickly and stays around zero at larger lags. On the other hand, if the chain has a poor mixing, then its autocorrelation will decay gradually and still exist at a large lag. However, the first lag $h^{\max}$ where the autocorrelations decline to zero is highly dependent on the dimensionality of the posterior distribution. The figures

below exhibit the behaviours of the autocorrelations from a well mixed chain and a chain with poor mixing after the Burn-in stages in one-dimensional problem.

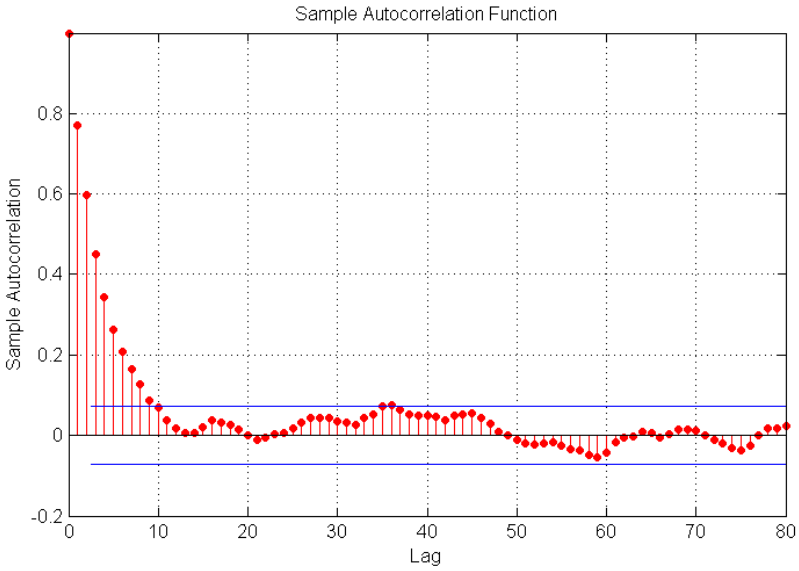


Figure 2.5: Sample autocorrelation function of a well mixed chain (after Burn-in).

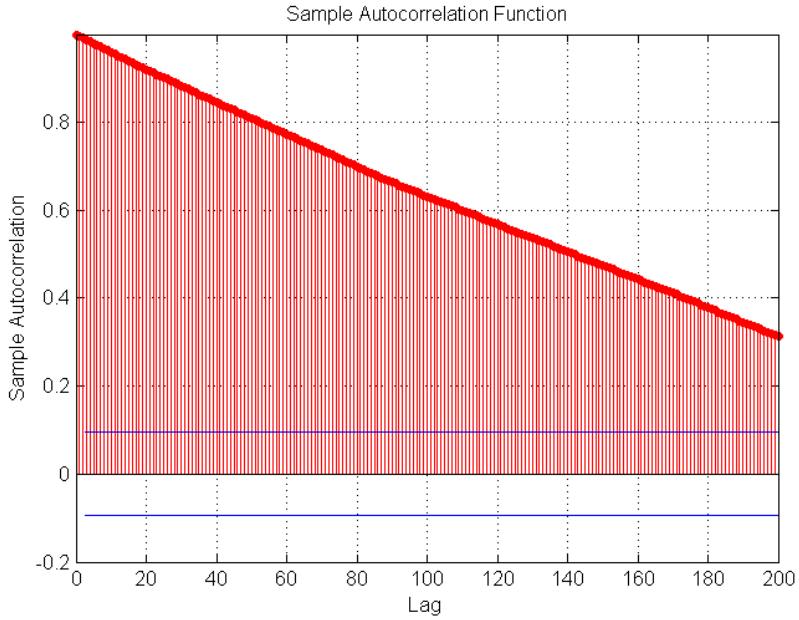


Figure 2.6: Sample autocorrelation function of a poorly mixed chain (after Burn-in).

2.4.4 Numerical Error

Due to the finite number of iterations performed, we have to evaluate the numerical errors of the obtained estimates. Consider the following estimator

$$\widehat{\Omega} = \widehat{\mathbb{E}}[\, g(\theta)\,] = \frac{1}{M}\sum_{i=1}^{M} g(\theta^{(i)}).$$

However, the standard error of the estimator $\widehat{\Omega}$ cannot be computed in the usual way, since there are serial correlations between the samples simulated from MCMC algorithms. The obvious method is to keep every $h^{\max}$-th (the first lag where the autocorrelations decline to zero) sample from the chain to receive approximately independent samples. Moreover, one of the popular and well developed approaches is to calculate the *effective sample size*, $M_{\text{eff}} = M/\tau$, where τ is given by

$$\tau = 1 + 2\sum_{k=1}^{\infty} \text{ACF}[\theta,\, k]. \tag{2.4.1}$$

Generally, we cut off the sum in (2.4.1) at $k = h^{\max}$ in order to get the estimate of τ. Hence, the standard error of $\widehat{\Omega}$ can be estimated by

$$\text{stderr}[\,\widehat{\Omega}\,] = \frac{\text{stdev}[\, g(\theta)\,]}{\sqrt{M_{\text{eff}}}}.$$

Chapter 3

Pseudo-Marginal Metropolis-Hastings (PMMH) Approach

In this chapter, we first describe the underlying mechanism of the PMMH algorithm and formally outline the procedure of this approach. Furthermore, the additional implementation issue associated with the PMMH method is considered. Finally, we briefly introduce two recently proposed modifications of the PMMH algorithm.

3.1 Mechanism and Algorithm

Consider an independent sample of T observations, $\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_T$, each $\vec{x}_i$ has a dimension $J \geqslant 1$. Let $p(\vec{x}_i \,|\, \boldsymbol{\theta})$ denotes the common density function of each $\vec{x}_i$ for $i = 1, 2, \ldots, T$, where $\boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^d$ represents the unknown parameter. Define $\boldsymbol{x} \overset{\text{def}}{=} (\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_T)$ as a vector of T observations. Thus, the likelihood function can be written as

$$p(\boldsymbol{x} \,|\, \boldsymbol{\theta}) = \prod_{i=1}^{T} p(\vec{x}_i \,|\, \boldsymbol{\theta}).$$

Assume that each likelihood contribution $p(\vec{x}_i \,|\, \boldsymbol{\theta})$ is intractable but can be estimated unbiasedly and non-negatively by $\widehat{p}(\vec{x}_i \,|\, \boldsymbol{\theta})$, then the estimated likelihood can be written as

$$\widehat{p}_N(\boldsymbol{x} \,|\, \boldsymbol{\theta}) = \widehat{p}_N(\boldsymbol{x} \,|\, \boldsymbol{\theta}, \boldsymbol{u}) = \prod_{i=1}^{T} \widehat{p}(\vec{x}_i \,|\, \boldsymbol{\theta}),$$

where $\boldsymbol{u}$ denotes all the auxiliary random variables or the set of independent random numbers used to compute $\widehat{p}_N(\boldsymbol{x} \,|\, \boldsymbol{\theta})$ and N is the quantity of random numbers or the number of particles used to construct the estimator of each likelihood contribution $\widehat{p}(\vec{x}_i \,|\, \boldsymbol{\theta})$. We denote the density function of $\boldsymbol{u}$ by $p_U(\boldsymbol{u})$, then unbiasedness of the likelihood estimator $\widehat{p}_N(\boldsymbol{x} \,|\, \boldsymbol{\theta}, \boldsymbol{u})$ implies

$$\mathbb{E}\left[\, \widehat{p}_N(\boldsymbol{x} \,|\, \boldsymbol{\theta}, \boldsymbol{u}) \,\right] = \int \widehat{p}_N(\boldsymbol{x} \,|\, \boldsymbol{\theta}, \boldsymbol{u})\, p_U(\boldsymbol{u})\, \mathrm{d}\boldsymbol{u} = p(\boldsymbol{x} \,|\, \boldsymbol{\theta}). \tag{3.1.1}$$

The joint density of $\boldsymbol{\theta}$ and $\boldsymbol{u}$ can be defined as

$$\begin{aligned}\pi_N(\boldsymbol{\theta}, \boldsymbol{u} \,|\, \boldsymbol{x}) &\stackrel{\text{def}}{=} \frac{p(\boldsymbol{\theta})\, \widehat{p}_N(\boldsymbol{x} \,|\, \boldsymbol{\theta}, \boldsymbol{u})\, p_U(\boldsymbol{u})}{\int p(\boldsymbol{x} \,|\, \boldsymbol{\theta})\, p(\boldsymbol{\theta})\, \mathrm{d}\boldsymbol{\theta}} \\ &= p(\boldsymbol{\theta})\, \widehat{p}_N(\boldsymbol{x} \,|\, \boldsymbol{\theta}, \boldsymbol{u})\, p_U(\boldsymbol{u}) / p(\boldsymbol{x}) \\ &\propto p(\boldsymbol{\theta})\, \widehat{p}_N(\boldsymbol{x} \,|\, \boldsymbol{\theta}, \boldsymbol{u})\, p_U(\boldsymbol{u}).\end{aligned}$$

By using (3.1.1), we can see that $\pi(\boldsymbol{\theta} \,|\, \boldsymbol{x})$ is a marginal distribution of the joint density $\pi_N(\boldsymbol{\theta}, \boldsymbol{u} \,|\, \boldsymbol{x})$ because

$$\begin{aligned}\int \pi_N(\boldsymbol{\theta}, \boldsymbol{u} \,|\, \boldsymbol{x})\, \mathrm{d}\boldsymbol{u} &= \int p(\boldsymbol{\theta})\, \widehat{p}_N(\boldsymbol{x} \,|\, \boldsymbol{\theta}, \boldsymbol{u})\, p_U(\boldsymbol{u}) / p(\boldsymbol{x})\, \mathrm{d}\boldsymbol{u} \\ &= p(\boldsymbol{\theta}) / p(\boldsymbol{x}) \int \widehat{p}_N(\boldsymbol{x} \,|\, \boldsymbol{\theta}, \boldsymbol{u})\, p_U(\boldsymbol{u})\, \mathrm{d}\boldsymbol{u} \\ &= p(\boldsymbol{x} \,|\, \boldsymbol{\theta})\, p(\boldsymbol{\theta}) / p(\boldsymbol{x}) \\ &= \pi(\boldsymbol{\theta} \,|\, \boldsymbol{x}).\end{aligned}$$

Therefore, we can obtain samples from the posterior distribution $\pi(\boldsymbol{\theta} \,|\, \boldsymbol{x})$ by sampling from the joint density $\pi_N(\boldsymbol{\theta}, \boldsymbol{u} \,|\, \boldsymbol{x})$.

Let $q(\boldsymbol{\theta}^* \,|\, \boldsymbol{\theta})$ denotes a proposal density for the parameter $\boldsymbol{\theta}^*$ conditional on the current state $\boldsymbol{\theta}$ and $q(\boldsymbol{u}^* \,|\, \boldsymbol{u})$ denotes a proposal density for the set of random numbers $\boldsymbol{u}^*$, where $\boldsymbol{u}$ is the corresponding current set of random numbers used to compute $\widehat{p}_N(\boldsymbol{x} \,|\, \boldsymbol{\theta}, \boldsymbol{u})$. We assume that the transition kernel $q(\boldsymbol{u}^* \,|\, \boldsymbol{u})$ satisfies the detailed balance condition with respect to $p_U(\boldsymbol{u})$, that is,

$$p_U(\boldsymbol{u}^*)\, q(\boldsymbol{u}^* \,|\, \boldsymbol{u}) = p_U(\boldsymbol{u})\, q(\boldsymbol{u} \,|\, \boldsymbol{u}^*).$$

The pseudo - marginal Metropolis - Hastings algorithm simulates samples from $\pi(\boldsymbol{\theta} \,|\, \boldsymbol{x})$ by generating a Markov chain whose invariant distribution is $\pi_N(\boldsymbol{\theta}, \boldsymbol{u} \,|\, \boldsymbol{x})$ with proposal density

$$\begin{aligned}q(\boldsymbol{\theta}^*, \boldsymbol{u}^* \,|\, \boldsymbol{\theta}, \boldsymbol{u}) &= q(\boldsymbol{\theta}^* \,|\, \boldsymbol{\theta})\, q(\boldsymbol{u}^* \,|\, \boldsymbol{u}) \\ &= q(\boldsymbol{\theta}^* \,|\, \boldsymbol{\theta})\, p_U(\boldsymbol{u}^*).\end{aligned}$$

Thus, the proposal $(\boldsymbol{\theta}^*, \boldsymbol{u}^*)$ is accepted with probability

$$\begin{aligned}\alpha(\boldsymbol{\theta}, \boldsymbol{u}; \boldsymbol{\theta}^*, \boldsymbol{u}^*) &= \min\left\{1,\ \frac{\pi_N(\boldsymbol{\theta}^*, \boldsymbol{u}^* \,|\, \boldsymbol{x})\, q(\boldsymbol{\theta}, \boldsymbol{u} \,|\, \boldsymbol{\theta}^*, \boldsymbol{u}^*)}{\pi_N(\boldsymbol{\theta}, \boldsymbol{u} \,|\, \boldsymbol{x})\, q(\boldsymbol{\theta}^*, \boldsymbol{u}^* \,|\, \boldsymbol{\theta}, \boldsymbol{u})}\right\} \\ &= \min\left\{1,\ \frac{p(\boldsymbol{\theta}^*)\, \widehat{p}_N(\boldsymbol{x} \,|\, \boldsymbol{\theta}^*, \boldsymbol{u}^*)\, p_U(\boldsymbol{u}^*)\, q(\boldsymbol{\theta} \,|\, \boldsymbol{\theta}^*)\, p_U(\boldsymbol{u})}{p(\boldsymbol{\theta})\, \widehat{p}_N(\boldsymbol{x} \,|\, \boldsymbol{\theta}, \boldsymbol{u})\, p_U(\boldsymbol{u})\, q(\boldsymbol{\theta}^* \,|\, \boldsymbol{\theta})\, p_U(\boldsymbol{u}^*)}\right\} \\ &= \min\left\{1,\ \frac{p(\boldsymbol{\theta}^*)\, \widehat{p}_N(\boldsymbol{x} \,|\, \boldsymbol{\theta}^*, \boldsymbol{u}^*)\, q(\boldsymbol{\theta} \,|\, \boldsymbol{\theta}^*)}{p(\boldsymbol{\theta})\, \widehat{p}_N(\boldsymbol{x} \,|\, \boldsymbol{\theta}, \boldsymbol{u})\, q(\boldsymbol{\theta}^* \,|\, \boldsymbol{\theta})}\right\},\end{aligned}$$

where every component in the acceptance ratio is computable. The pseudo - marginal Metropolis - Hastings (PMMH) algorithm is summarized below.

Algorithm 3 The Pseudo - Marginal Metropolis - Hastings Algorithm

Require: A suitable proposal distribution $q(\boldsymbol{\theta}^* \,|\, \boldsymbol{\theta})$, the density function $p_U(\boldsymbol{u})$, the target density $\pi_N(\boldsymbol{\theta}, \boldsymbol{u} \,|\, \boldsymbol{x})$ and set the length of MCMC run M.

Initialise $\boldsymbol{\theta}$ with any value within a support of $\pi_N(\boldsymbol{\theta}, \boldsymbol{u} \,|\, \boldsymbol{x})$.

Sample $\boldsymbol{u}$ from $p_U(\boldsymbol{u})$.

Compute the estimator $\widehat{p}_N(\boldsymbol{x} \,|\, \boldsymbol{\theta}, \boldsymbol{u})$ of $p(\boldsymbol{x} \,|\, \boldsymbol{\theta})$.

for $i = 2, 3, \ldots, M$ **do**

Generate a proposal $\boldsymbol{\theta}^*$ from $q(\boldsymbol{\theta}^* \,|\, \boldsymbol{\theta})$.

Sample $\boldsymbol{u}^*$ from $p_U(\boldsymbol{u}^*)$.

Compute the estimator $\widehat{p}_N(\boldsymbol{x} \,|\, \boldsymbol{\theta}^*, \boldsymbol{u}^*)$ of $p(\boldsymbol{x} \,|\, \boldsymbol{\theta}^*)$.

Accept proposal $(\boldsymbol{\theta}^*, \boldsymbol{u}^*)$ with the acceptance probability

$$\alpha(\boldsymbol{\theta}, \boldsymbol{u}; \boldsymbol{\theta}^*, \boldsymbol{u}^*) = \min\left\{1,\ \frac{p(\boldsymbol{\theta}^*)\, \widehat{p}_N(\boldsymbol{x} \,|\, \boldsymbol{\theta}^*, \boldsymbol{u}^*)\, q(\boldsymbol{\theta} \,|\, \boldsymbol{\theta}^*)}{p(\boldsymbol{\theta})\, \widehat{p}_N(\boldsymbol{x} \,|\, \boldsymbol{\theta}, \boldsymbol{u})\, q(\boldsymbol{\theta}^* \,|\, \boldsymbol{\theta})}\right\},$$

that is, we simulate U from the uniform distribution $\mathsf{U}(0,\ 1)$ and output $(\boldsymbol{\theta}^*, \boldsymbol{u}^*)$ if $U < \alpha(\boldsymbol{\theta}, \boldsymbol{u}; \boldsymbol{\theta}^*, \boldsymbol{u}^*)$, otherwise we output $(\boldsymbol{\theta}, \boldsymbol{u})$.

In practice, we only need to record $\{\boldsymbol{\theta},\ \widehat{p}_N(\boldsymbol{x} \,|\, \boldsymbol{\theta}, \boldsymbol{u})\}$ instead of $\{\boldsymbol{\theta}, \boldsymbol{u}\}$ since $\widehat{p}_N(\boldsymbol{x} \,|\, \boldsymbol{\theta}, \boldsymbol{u})$ will be re-used in the next iteration. In order to state the convergence result of the PMMH algorithm, we introduce the following notations. Let P denotes the transition probability of a standard Metropolis - Hastings algorithm which targets the posterior density $\pi(\boldsymbol{\theta} \,|\, \boldsymbol{x})$ (the marginal distribution of the joint density $\pi_N(\boldsymbol{\theta}, \boldsymbol{u} \,|\, \boldsymbol{x})$) and uses $q(\boldsymbol{\theta}^* \,|\, \boldsymbol{\theta})$ as proposal distribution. Also, we denote the transition probability of the PMMH algorithm by $\widetilde{P}_N$ where N is the quantity of random numbers used to form the estimator of each likelihood contribution $\widehat{p}(\vec{x}_i \,|\, \boldsymbol{\theta})$. The following theorem is adapted from Andrieu and Roberts [2009].

Theorem 3.1.1. *Assume that P defines a π-irreducible and aperiodic Markov chain with invariant distribution $\pi(\boldsymbol{\theta} \mid \boldsymbol{x})$. Then for any $N \in \mathbb{N}$, $\widetilde{P}_N$ is also π-irreducible and aperiodic, and there is a convergence in total variation*

$$\left\| \widetilde{P}_N^k(\boldsymbol{\theta}_0, \boldsymbol{u}_0; \cdot\,, \cdot) - \pi_N(\cdot\,, \cdot \mid \boldsymbol{x}) \right\| \to 0$$

as $k \to \infty$, where $(\boldsymbol{\theta}_0, \boldsymbol{u}_0) \in \boldsymbol{\Theta} \times \mathcal{U}^N$ denotes the starting point.

Theorem 3.1.1 shows that the PMMH algorithm generates a Markov chain which converges to its invariant distribution $\pi_N(\boldsymbol{\theta}, \boldsymbol{u} \mid \boldsymbol{x})$ if the corresponding standard Metropolis - Hastings algorithm produces an aperiodic and irreducible Markov chain with invariant distribution $\pi(\boldsymbol{\theta} \mid \boldsymbol{x})$. Andrieu and Vihola [2015] found that the asymptotic variance of the PMMH algorithm is always at least as large as that of the standard Metropolis - Hastings algorithm. However, they proved that under general conditions, the asymptotic variance of the PMMH algorithm converges to that of the standard Metropolis - Hastings algorithm if the accuracy of the estimators is increased.

The key feature of PMMH algorithm is that if the likelihood estimator is non-negative and unbiased, then the invariant distribution of the generated Markov chain will remain exactly as it would be if the true likelihood had been available. Furthermore, it is exact even if the estimator of the likelihood is very noisy and imprecise despite the fact that the mixing of the chain will be poorer in this case. Hence, the PMMH method is in fact "exact approximation" to the standard Metropolis - Hastings algorithm.

3.2 Implementation Issue

In addition to the choice of proposal distribution inherent to any Metropolis - Hastings algorithm, the main practical issue with the PMMH algorithm is the choice of N which is the number of Monte Carlo samples or particles used to estimate the likelihood. In this section, we name the averages computed under the chain generated by the PMMH algorithm as *pseudo - marginal averages*.

Andrieu and Vihola [2015] show that for estimating the likelihood based on importance sampling, using many Monte Carlo samples usually results in pseudo - marginal averages with asymptotic variances lower than the corresponding averages using fewer samples. Empirical evidence suggests that this result also holds when

the likelihood is unbiasedly estimated by particle filters. However, the computational cost of construction of the likelihood estimator increases with N.

Therefore, when an unbiased estimator of the likelihood is used within a Metropolis - Hastings algorithm, it is necessary to trade-off between the number of Monte Carlo samples or particles used to construct this estimator and the asymptotic variance of the computed pseudo - marginal average. Pitt et al. [2012] and Doucet et al. [2015] studied the effect of estimating the likelihood within Markov chain Monte Carlo simulation and found that the number of particles N has to be selected such that the variance of the log-likelihood estimator should be around 1 in order to obtain an optimal trade-off between the efficiency of the Markov chain and the computing cost; moreover, to avoid the chain getting stuck in the simulation. The figure below provides a sample result when the variance of the log-likelihood estimator is set beyond 1 and the chain getting stuck at some states.

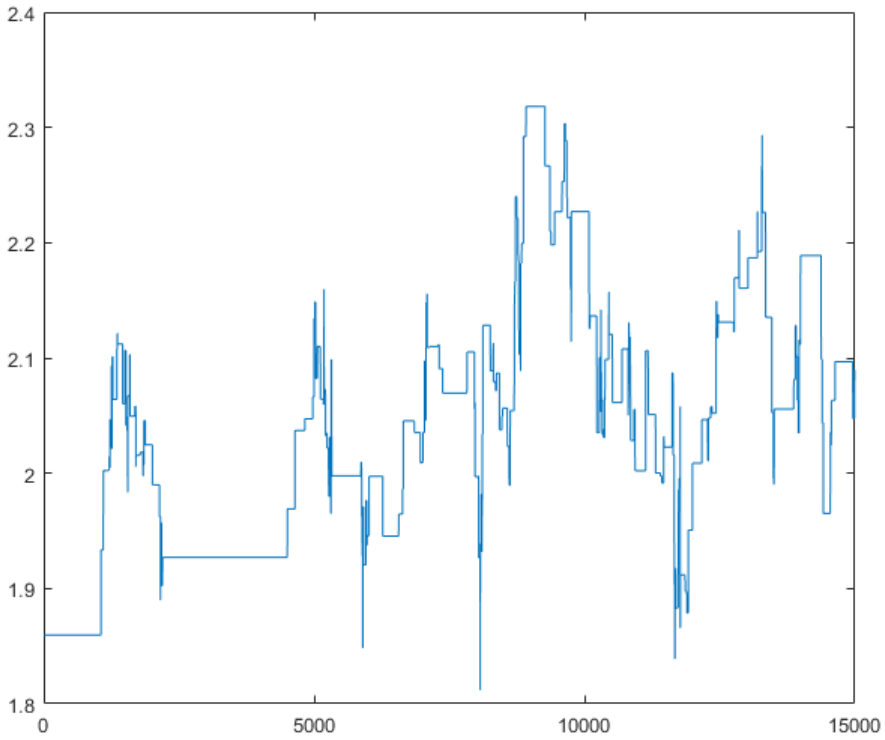


Figure 3.1: The standard PMMH algorithm with $\widehat{\mathbb{V}\mathrm{ar}}[\log \widehat{p}_N(\boldsymbol{x} \mid \boldsymbol{\theta})] = 10.311$.

Furthermore, the variance of the log-likelihood estimator grows linearly with the number of observations T. Hence, it is necessary to increase the number of Monte Carlo samples N in proportion to T so that the variance of the log-likelihood estimator can be kept below 1. This fact implies that the PMMH algorithm requires $O(T^2)$ operations at each MCMC iteration and thus makes the algorithm highly inefficient when dealing with a huge dataset.

3.3 Variants of the PMMH Algorithm

In this section, we are going to briefly introduce two newly proposed variants of the pseudo - marginal Metropolis - Hastings algorithm. These modifications greatly reduce the computational cost of the standard PMMH algorithm.

3.3.1 Correlated Pseudo - Marginal Algorithm

Deligiannidis et al. [2016] propose a modification of the standard PMMH algorithm which is so-called the correlated pseudo - marginal (CPM) method. The proposed algorithm positively correlates the unbiased log-likelihood estimators at the current and proposed values of the parameters so that the variation of the difference $\log \widehat{p}_N(\boldsymbol{x} \mid \boldsymbol{\theta}^*, \boldsymbol{u}^*) - \log \widehat{p}_N(\boldsymbol{x} \mid \boldsymbol{\theta}, \boldsymbol{u})$ in the Metropolis - Hastings acceptance ratio can be reduced. Correlation between the estimators is introduced by correlating the auxiliary random variables $\boldsymbol{u}$ used to obtain the unbiased estimators. The following algorithm outlines the procedure of the correlated pseudo - marginal algorithm where the constant ρ controls the strength of the correlation.

Algorithm 4 The Correlated Pseudo - Marginal Algorithm

Steps

1: Sample $\boldsymbol{\theta}^* \sim q(\boldsymbol{\theta}^* \mid \boldsymbol{\theta})$.

2: If the likelihood estimator is computed using

- standard normal random numbers, then sample $\boldsymbol{u}' \sim \mathsf{N}(\mathbf{0}, \mathrm{I})$ and set $\boldsymbol{u}^* = \rho \boldsymbol{u} + \sqrt{1-\rho^2}\, \boldsymbol{u}'$
- uniform random numbers, then we need one more step: $\Phi(\boldsymbol{u}^*)$

where $\boldsymbol{u}$ has a dimension $J \geqslant 1$.

3: Compute the estimator $\widehat{p}_N(\boldsymbol{x} \mid \boldsymbol{\theta}^*, \boldsymbol{u}^*)$ of $p(\boldsymbol{x} \mid \boldsymbol{\theta}^*)$.

4: Accept the proposal $(\boldsymbol{\theta}^*, \boldsymbol{u}^*)$ with the acceptance probability

$$\alpha(\boldsymbol{\theta}, \boldsymbol{u}; \boldsymbol{\theta}^*, \boldsymbol{u}^*) = \min \left\{ 1, \frac{p(\boldsymbol{\theta}^*)\, \widehat{p}_N(\boldsymbol{x} \mid \boldsymbol{\theta}^*, \boldsymbol{u}^*)\, q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^*)}{p(\boldsymbol{\theta})\, \widehat{p}_N(\boldsymbol{x} \mid \boldsymbol{\theta}, \boldsymbol{u})\, q(\boldsymbol{\theta}^* \mid \boldsymbol{\theta})} \right\}.$$

In contrast to the standard PMMH algorithm, we also need to store $\boldsymbol{u}$ in addition to $\boldsymbol{\theta}$ and $\widehat{p}_N(\boldsymbol{x} \mid \boldsymbol{\theta}, \boldsymbol{u})$ in practice since the proposed $\boldsymbol{u}^*$ in the next iteration is dependent on the current state of $\boldsymbol{u}$. Generally, we select ρ close to 1 to induce a strong positive correlation between the estimators $\widehat{p}_N(\boldsymbol{x} \mid \boldsymbol{\theta}, \boldsymbol{u})$ and $\widehat{p}_N(\boldsymbol{x} \mid \boldsymbol{\theta}^*, \boldsymbol{u}^*)$. However, the parameter ρ cannot be set too high, because we have to ensure that the CPM algorithm will cover the whole space of $\boldsymbol{u}$ for ergodicity. See Deligiannidis et al. [2016] for the detailed guidelines on the parameter settings. They show that the correlated pseudo - marginal (CPM) method requires $O(T^{3/2})$ operations at each MCMC iteration compared to the $O(T^2)$ per iteration in standard PMMH algorithm. Thus, the number of particles N can be scaled sub-linearly with the number of observations T in order to keep the variance of the log-likelihood estimator below 1. Moreover, the CPM approach also helps the chain to mix well even if highly variable likelihood estimates are used. Hence, the CPM method has a much lower computational cost at each iteration than the standard PMMH algorithm.

3.3.2 Block-wise PMMH Algorithm

Tran et al. [2016] propose an alternative approach to the CPM method, which is called the block-wise PMMH algorithm. The proposed method works similarly to the CPM approach but is less general. The block-wise PMMH algorithm divides the set of random numbers $\boldsymbol{u}$ into blocks and updates one block at each iteration instead

of updating a complete set of random numbers $\boldsymbol{u}$ as in the standard PMMH approach. By dividing the set of random numbers $\boldsymbol{u}$ into G blocks $\boldsymbol{u}_{(1)}, \boldsymbol{u}_{(2)}, \ldots, \boldsymbol{u}_{(G)}$, the target density from the standard PMMH algorithm can be written as

$$\pi_N(\boldsymbol{\theta}, \boldsymbol{u}_{(1)}, \ldots, \boldsymbol{u}_{(G)} \,|\, \boldsymbol{x}) = p(\boldsymbol{\theta})\, \widehat{p}_N(\boldsymbol{x} \,|\, \boldsymbol{\theta}, \boldsymbol{u}_{(1)}, \ldots, \boldsymbol{u}_{(G)})\, p_U(\boldsymbol{u}_{(1)}, \ldots, \boldsymbol{u}_{(G)})/p(\boldsymbol{x}).$$

The proposal $(\boldsymbol{\theta}^*, \boldsymbol{u}^*_{(K)})$ is generated in every MCMC iteration, where the block index K is randomly chosen from the set $\{1, \ldots, G\}$ with $\mathbb{P}(K = k) > 0$, for $k = 1, 2, \ldots, G$. Generally, we choose uniformly distributed probabilities $\mathbb{P}(K = k) = G^{-1}$. Therefore, the acceptance probability becomes

$$\alpha(\boldsymbol{\theta}, \boldsymbol{u}_{(k)}; \boldsymbol{\theta}^*, \boldsymbol{u}^*_{(k)}) = \min\left\{1,\, \frac{p(\boldsymbol{\theta}^*)\, \widehat{p}_N(\boldsymbol{x} \,|\, \boldsymbol{\theta}^*, \boldsymbol{u}_{(1)}, \ldots, \boldsymbol{u}^*_{(k)}, \ldots, \boldsymbol{u}_{(G)})\, q(\boldsymbol{\theta} \,|\, \boldsymbol{\theta}^*)}{p(\boldsymbol{\theta})\, \widehat{p}_N(\boldsymbol{x} \,|\, \boldsymbol{\theta}, \boldsymbol{u}_{(1)}, \ldots, \boldsymbol{u}_{(k)}, \ldots, \boldsymbol{u}_{(G)})\, q(\boldsymbol{\theta}^* \,|\, \boldsymbol{\theta})}\right\}.$$

Refer to Tran et al. [2016] for the detailed guidelines of setting the algorithm parameter G and choosing the number of particles N. By updating only one block $\boldsymbol{u}^*_{(k)}$ at each iteration, the variation in the Metropolis - Hastings acceptance ratio is reduced. As a result, the block-wise PMMH algorithm helps the chain to mix well even if highly variable likelihood estimates are used and thus the number of particles N required at every iteration can be decreased significantly. Tran et al. [2016] have also proved that the block-wise PMMH algorithm only requires $O(T^{3/2})$ operations at each MCMC iteration as in the CPM approach. By comparing with the CPM method, the block-wise PMMH algorithm has the following advantages. First, it provides a more direct way to control the correlation between the log-likelihood estimators at the current and proposed values of the parameters. Secondly, blocking is shown to be a natural way to perform the PMMH algorithm in panel data models and subsampling problems.

Chapter 4

Application

In this chapter, we apply the pseudo-marginal Metropolis-Hastings (PMMH) algorithm to the Bayesian estimation of Gaussian copula model with discrete and mixed margins in high dimensions. The numerical studies confirm the competence of the PMMH algorithm in this new application area. The theories and ideas in this chapter are borrowed from Gunawan et al. [2016a].

4.1 Gaussian Copula with Discrete Margins

A J-dimensional copula is a multivariate distribution function defined on $[0,1]^J$, whose univariate marginals are uniformly distributed on $[0,1]$. Sklar [1959] states that the joint cumulative distribution function of random variables $X_1, X_2, \ldots, X_J$ with marginal distributions $F_1(x_1), F_2(x_2), \ldots, F_J(x_J)$ can be written as

$$F(x_1, x_2, \ldots, x_J) = C(F_1(x_1), F_2(x_2), \ldots, F_J(x_J)).$$

In this section, we focus on the Gaussian copula model with discrete margins. Suppose $\boldsymbol{X} := (X_1, X_2, \ldots, X_J)$ is a set of discrete random variables and we say that $\boldsymbol{X}$ is generated by a Gaussian copula based on the latent vector $\boldsymbol{Z} := (Z_1, Z_2, \ldots, Z_J) \sim \mathbb{N}_J(\boldsymbol{0}, \Sigma)$, if

$$p(\mathrm{d}\boldsymbol{X}) = \int_{\sqrt{\sigma_{11}}\Phi^{-1}(F_1(x_1^-))}^{\sqrt{\sigma_{11}}\Phi^{-1}(F_1(x_1))} \cdots \int_{\sqrt{\sigma_{JJ}}\Phi^{-1}(F_J(x_J^-))}^{\sqrt{\sigma_{JJ}}\Phi^{-1}(F_J(x_J))} \phi_J(\boldsymbol{z} \,|\, \Sigma)\, \mathrm{d}\boldsymbol{z} \tag{4.1.1}$$

where $\phi_J(\boldsymbol{z} \,|\, \Sigma)$ denotes the density function of vector $\boldsymbol{Z}$, σ_{jj} is the j-th diagonal element of the covariance matrix Σ, $\Phi^{-1}(\cdot)$ is the inverse of standard normal cumulative distribution function in one dimension and $F_j(x_j^-) = \lim_{s \uparrow x_j} F_j(s)$.

The likelihood function (4.1.1) needs to be evaluated for all the observations in each iteration when implementing the standard Metropolis - Hastings algorithm.

However, by introducing the *difference operator notation* from Gunawan et al. [2016b], we can see that the evaluation of (4.1.1) is computationally intractable

$$\begin{aligned} p(\mathrm{d}\boldsymbol{X}) &= \int_{\sqrt{\sigma_{11}}\Phi^{-1}(F_1(x_1^-))}^{\sqrt{\sigma_{11}}\Phi^{-1}(F_1(x_1))} \cdots \int_{\sqrt{\sigma_{JJ}}\Phi^{-1}(F_J(x_J^-))}^{\sqrt{\sigma_{JJ}}\Phi^{-1}(F_J(x_J))} \phi_J(\boldsymbol{z} \mid \Sigma)\, \mathrm{d}\boldsymbol{z} \\ &= \triangle_{a_1}^{b_1} \triangle_{a_2}^{b_2} \cdots \triangle_{a_J}^{b_J} g_{\boldsymbol{z}}(. \mid \mathbf{0}, \Sigma) \end{aligned} \tag{4.1.2}$$

where a_i and b_i denote the lower and upper limits in each integral level respectively. Also, $g_{\boldsymbol{z}}(. \mid \mathbf{0}, \Sigma)$ is the J-dimensional normal cumulative distribution function with zero mean vector and covariance matrix Σ, which can be computed exactly in many mathematical softwares. From (4.1.2), we can say that evaluating (4.1.1) requires computing 2^J terms and each of which is also computationally expensive. See Appendix C.3 for the full explanation of the difference operator notation $\triangle$.

Estimating the full covariance matrix is equivalent to estimating $J(J-1)/2$ parameters. In order to reduce the number of parameters to be estimated in high-dimensional Gaussian copula model, we follow Murray et al. [2013] and apply a factor copula model by decomposing the covariance matrix Σ into the following factor form

$$\Sigma := \mathrm{B}\mathrm{B}^\top + \mathrm{I}_J,$$

with $\mathrm{B} = (b_{ij})$ for $i = 1, 2, \ldots, J$ and $j = 1, 2, \ldots, k$, is the J by k loading matrix where k denotes the number of factors used in the factor copula model. Moreover, I_J is a J-dimensional identity matrix. To identify the loading matrix B it is necessary to constrain it. We follow Geweke and Zhou [1996], Lopes and West [2004], Ghosh and Dunson [2009] and Murray et al. [2013] to assume that B is a full rank block lower triangular matrix with strictly positive diagonal elements.

We apply the PMMH algorithm to the Bayesian estimation of Gaussian factor copula model and use the approach proposed by Genz [1992] to unbiasedly estimate the likelihood function in (4.1.1). Extensive comparisons amongst the several algorithms in the literature indicate that the algorithm proposed by Genz [1992] is the most accurate method for estimating a high-dimensional normal probability. Genz algorithm uses a randomised lattice rule and we prove below why it produces an unbiased estimate.

The Genz algorithm uses the point set $\{x_n + \sigma\}$ for $n = 0, 1, ..., N-1$, where $\{x\} = x - \lfloor x \rfloor$ is the fractional part of x (it is a component-wise operation if x is a vector), where σ is a random number which is uniformly distributed in $[0,1]^J$ and

$x_0, ..., x_{N-1}$ is a lattice rule. Suppose that we wish to estimate

$$\rho = \int f(x)\,\mathrm{d}x$$

by the approximation

$$\hat{\rho} = \frac{1}{N}\sum_{n=0}^{N-1} f(\{x_n + \sigma\}).$$

The approximation of the integral is unbiased, since

$$\begin{aligned}
\mathbb{E}(\hat{\rho}) &= \mathbb{E}\left(\frac{1}{N}\sum_{n=0}^{N-1} f(\{x_n + \sigma\})\right) \\
&= \frac{1}{N}\sum_{n=0}^{N-1} \mathbb{E}[f(\{x_n + \sigma\})] \\
&= \frac{1}{N}\sum_{n=0}^{N-1} \mathbb{E}[f(\sigma)] \\
&= \mathbb{E}[f(\sigma)] \\
&= \int f(x)\,\mathrm{d}x = \rho,
\end{aligned}$$

where we used the linearity of the expected value and the fact that if σ is uniformly distributed in $[0,1]^J$, then so is $\{x_n + \sigma\}$. Therefore, the estimate produced by the Genz algorithm is unbiased.

Prior to demonstrating some simulation studies, we first describe the data generation steps and the common specifications of the proposal distribution and prior distribution used in the PMMH algorithm.

Data Generation

Set the true values of the loading matrix B, then form the covariance matrix:

$$\Sigma = \mathrm{B}\mathrm{B}^\top + \mathrm{I}_J$$

where J denotes the dimension of the copula factor model. For a single factor model, the loading matrix B becomes a vector of length J with the first element being strictly positive:

$$\mathrm{B} = (\beta_1, \beta_2, \ldots, \beta_J)^\top.$$

1. Simulate J-variate $(Y_1, \ldots, Y_J)^\top$ from the J-dimensional normal distribution with zero mean vector and correlation matrix $corr(\Sigma)$, where $corr(\Sigma)$ denotes the correlation matrix converted from the covariance matrix Σ.
2. By calculating $U_1 = \widetilde{F}(Y_1), \ldots, U_J = \widetilde{F}(Y_J)$, we obtain $(U_1, \ldots, U_J)^\top$, where $\widetilde{F}(\cdot)$ is the standard normal cumulative distribution function in one dimension.
3. Transform $X_i = F_i^{-1}(U_i)$ for $i = 1, \ldots, J$, where the resulting $(X_1, \ldots, X_J)^\top$ are random variables from the Gaussian copula factor model with margins $F_i(\cdot)$ for $i = 1, \ldots, J$.

Repeat the above procedure T times for generating T observations.

Proposal Distribution

We use two steps for setting the proposal distribution used in the PMMH algorithm in order to obtain a better convergence and mixing of the generated Markov chain. We employ the random-walk Metropolis sampler whose proposal distribution has the form:

$$q(\boldsymbol{\beta}^* \mid \boldsymbol{\beta}) = \mathbb{N}_p(\boldsymbol{\beta}, c^2\, \widetilde{\Sigma})$$

where $p = \dim(\boldsymbol{\beta})$ is the number of parameters to be estimated and $\boldsymbol{\beta} = \mathrm{vec}(B)$ is the vector formed by the columns of loading matrix B.

1. Let $\widetilde{\Sigma} = \mathrm{I}_p$ and the scale factor c is set around $0.1/p$ to $0.5/p$ which depends on the dimensionality of the proposal. We run a preliminary MCMC simulation and get the empirical covariance structure $\overline{\Sigma}$ of the posterior distribution of parameters after Burin-in stage.
2. Run MCMC simulation again with $\widetilde{\Sigma} = \overline{\Sigma}$ and the scale factor c is set around $\sqrt{0.1}$ to $\sqrt{0.5}$ which is dependent on the problem.

Prior Distribution

The prior distribution of parameters $\boldsymbol{\beta}$ is specified according to the constraints on the loading matrix $\mathrm{B} = (b_{ij})$ for $i = 1, 2, \ldots, J$ and $j = 1, 2, \ldots, k$. For $i > j$, we use independent standard normal priors $p(b_{ij}) \sim \mathbb{N}(0, 1)$; for $i = j$, we use $p(b_{jj}) \propto \mathbb{N}(0, 1)\mathbb{I}\{b_{jj} > 0\}$, where $\mathbb{I}\{\cdot\}$ denotes an indicator function.

4.1.1 Simulation - 10 *Poisson Margins*

In this simulation study, we use a one-factor model ($k = 1$) with $J = 10$ Poisson margins and set the true values of the loading matrix B at

$$\mathrm{B} = (\beta_1, \ldots, \beta_{10})^\top = (0.806, 0.985, 0.997, 0.473, 0.650, 0.300, 0.530, 0.492, 0.205, 0.713)^\top.$$

We generated $T = 1000$ observations by the Gaussian copula factor model with covariance matrix Σ which is formed by $\mathrm{BB}^\top + \mathrm{I}_{10}$. The parameters of 10 Poisson margins are selected as below.

$$\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_{10})^\top = (1.0, 3.0, 1.75, 2.5, 6.75, 5.45, 3.86, 4.15, 1.25, 8.0)^\top.$$

In the simulation, the parameters of the marginal distributions are set at their true values because we desire to focus on how well the joint density is estimated. The starting values for all the coordinates are arbitrarily set at 2.00.

Results and Comments:

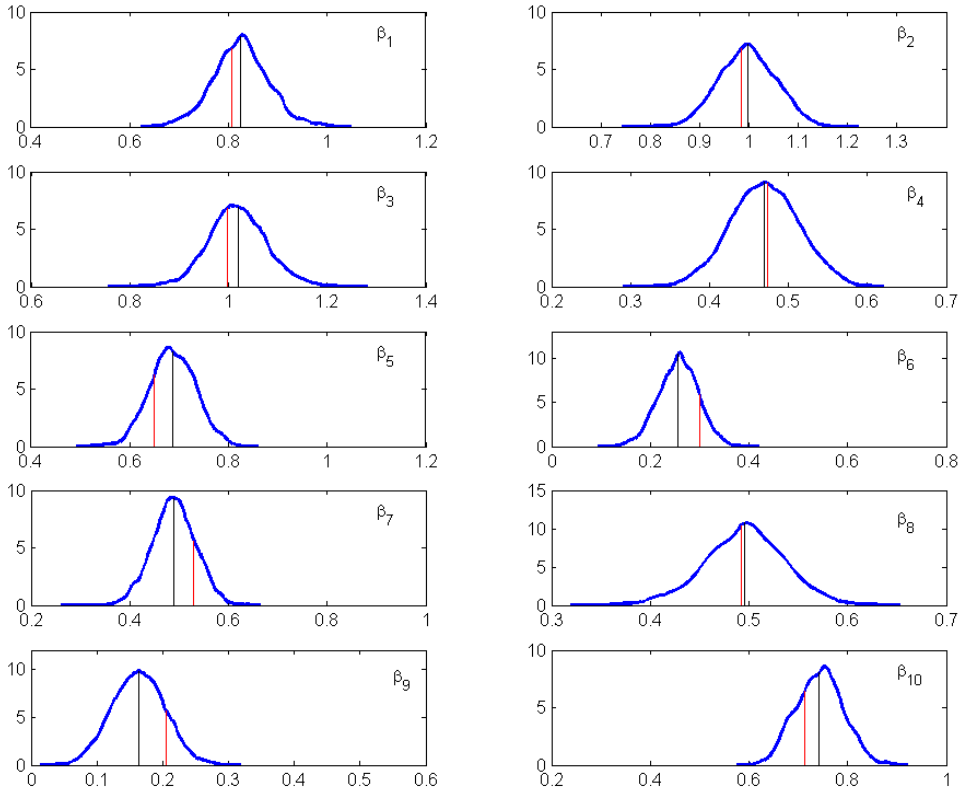
Table 4.1: Posterior mean estimates with posterior standard deviations reported in brackets for the one-factor copula model with $J = 10$ Poisson margins.

Parameter	True Value	Estimate
β_1	0.806	0.824 (0.0551)
β_2	0.985	0.998 (0.0565)
β_3	0.997	1.019 (0.0582)
β_4	0.473	0.470 (0.0433)
β_5	0.650	0.688 (0.0460)
β_6	0.300	0.257 (0.0402)
β_7	0.530	0.489 (0.0430)
β_8	0.492	0.496 (0.0402)
β_9	0.205	0.164 (0.0389)
β_{10}	0.713	0.743 (0.0473)

Points N	$\widehat{\mathbb{V}\mathrm{ar}}[\log \widehat{p}_N(\boldsymbol{x} \mid \boldsymbol{\theta})]$
30	0.054

Iterations	Acc. Prob.
15000	0.29

Figure 4.1: Posterior density estimations for all the coordinates of one-factor copula model with $J = 10$ Poisson margins (the true values are indicated by red vertical lines and the posterior mean estimates are indicated by black vertical lines).



From the table and figure above, we can see that all the estimated values are very close to their corresponding true values. The posterior density estimates are produced based on the kernel density estimation which is performed by MATLAB® built-in function `ksdensity`. We used 30 particles to ensure that the estimated variance of log-likelihood estimator below 1 for an optimal tradeoff between the efficiency of the Markov chain and the computational cost, as suggested by Pitt et al. [2012] and Doucet et al. [2015]. Each generated Markov chain consists of 15000 iterations with first 5000 iterates being discarded as Burin-in stage. The simulation achieves an overall acceptance probability of 0.29 which is considered as optimal in high dimensions.

Figure 4.2: Trace plots for all the coordinates of one-factor copula model with $J = 10$ Poisson margins.

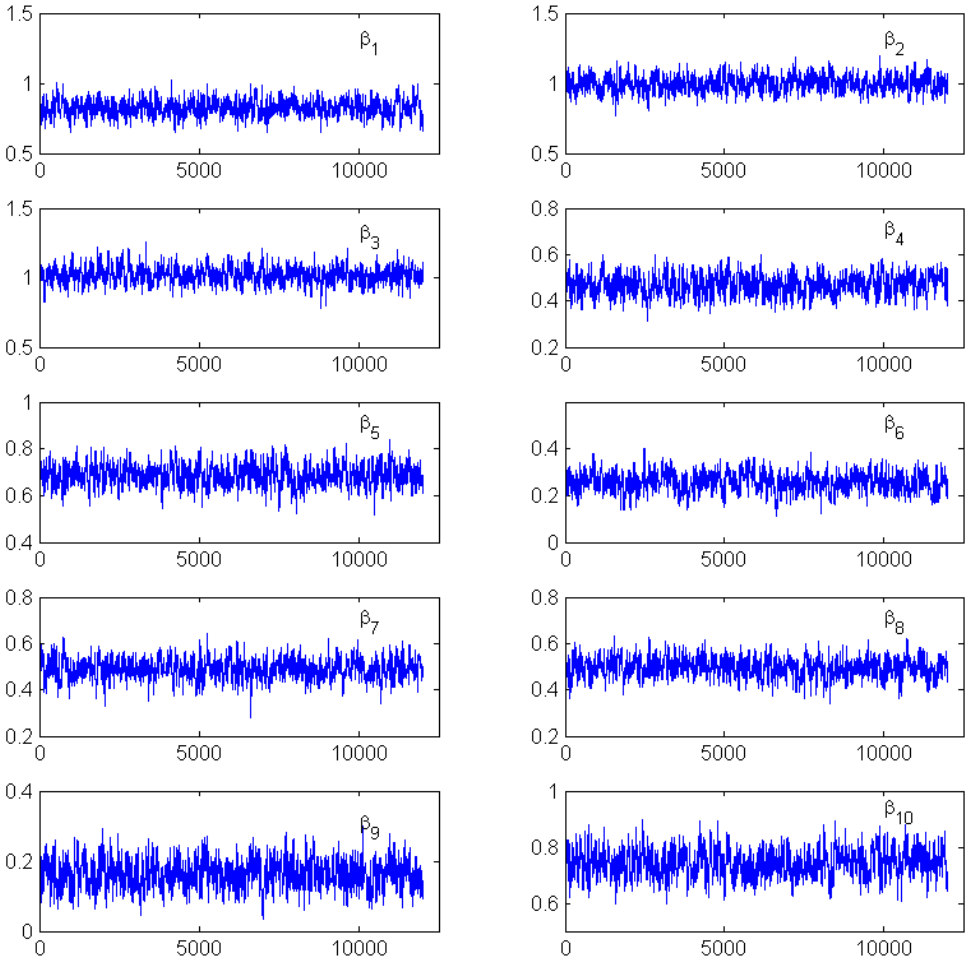
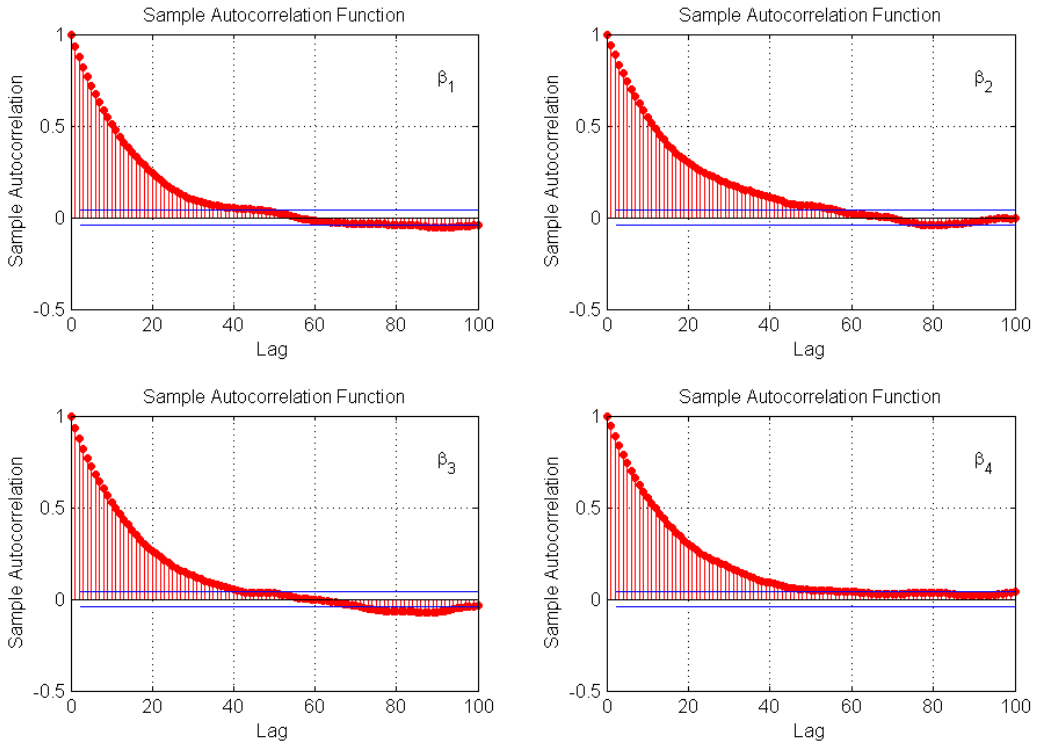


Figure 4.3: Sample autocorrelation functions for the first 4 coordinates of one-factor copula model with $J = 10$ Poisson margins.



The plot of sample autocorrelation function is only reported for the first 4 coordinates, and the rest are all similar. For all the coordinates, the sample autocorrelations are cut off at around lag 40 to lag 50. The traces plots and sample autocorrelation functions above indicate a good mixing of the generated Markov chain in 10 dimensions.

4.1.2 Simulation - 15 *Bernoulli Margins*

In the second numerical experiment, we apply a one-factor copula model ($k = 1$) with $J = 15$ Bernoulli margins and set the true values of the loading matrix B at

$$\mathrm{B} = (\beta_1, \ldots, \beta_{15})^\top = 2 \times \mathbf{1}_{15},$$

where $\mathbf{1}_{15}$ denotes a 15-dimensional column vector of 1s. We generated $T = 2000$ observations in this example by Gaussian copula one-factor model with covariance matrix Σ formed by $\mathrm{BB}^\top + \mathrm{I}_{15}$. The selected parameters of 15 Bernoulli marginal distributions are reported below.

$$\begin{aligned}\boldsymbol{p} &= (\mathrm{par}_1, \ldots, \mathrm{par}_{15})^\top \\ &= (0.40, 0.60, 0.60, 0.20, 0.30, 0.40, 0.20, 0.40, 0.35, 0.45, 0.90, 0.82, 0.25, 0.53, 0.71)^\top.\end{aligned}$$

As before, we set the parameters of the marginal distributions at their true values in the simulation, since we only want to focus on how well the joint density is estimated. The starting values of all the coordinates are arbitrarily chosen at 1.00 for the PMMH algorithm.

Results and Comments:

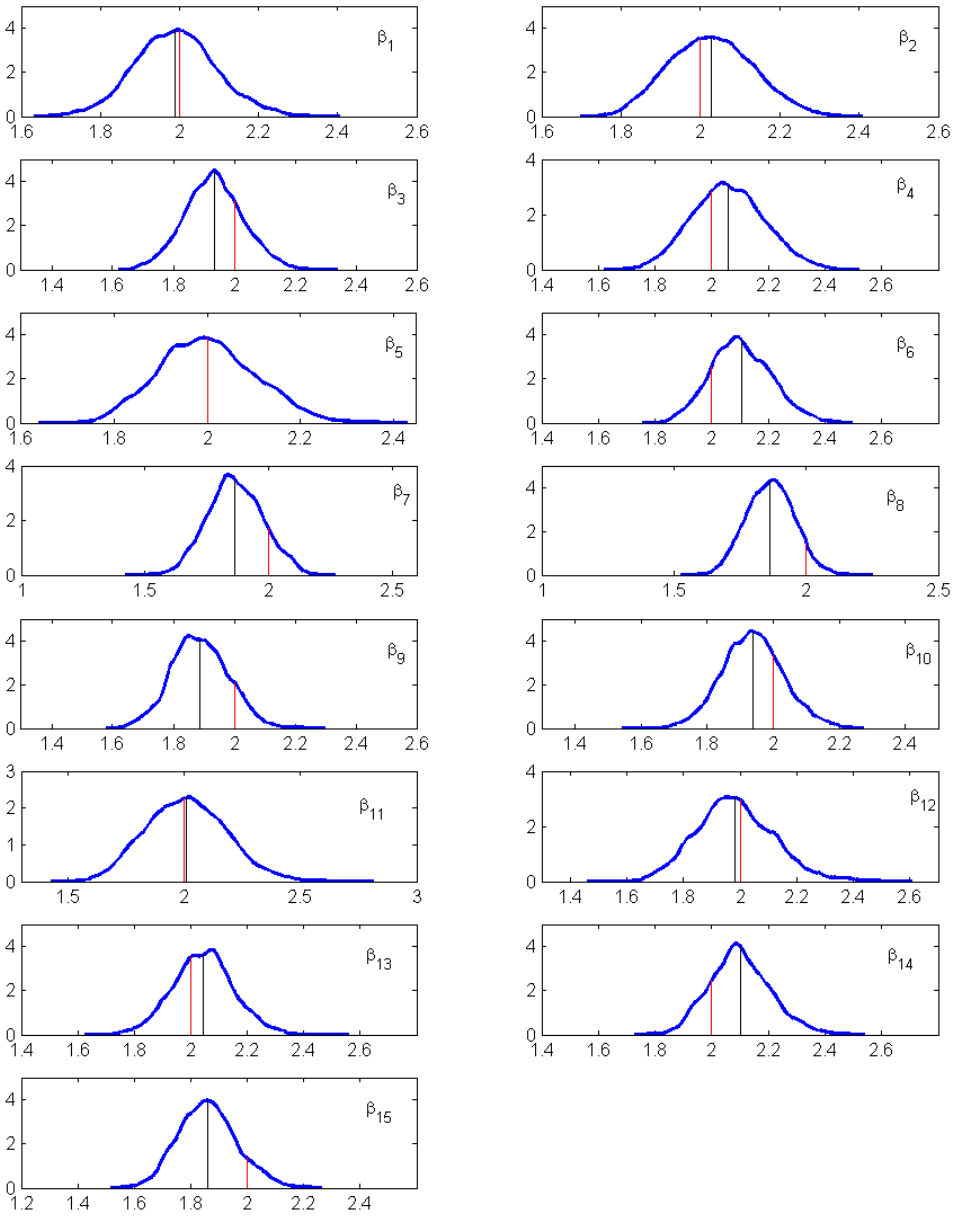
Table 4.2: Posterior mean estimates with posterior standard deviations reported in brackets for the one-factor copula model with $J = 15$ Bernoulli margins.

Parameter	True Value	Estimate
β_1	2.0	1.99 (0.103)
β_2	2.0	2.03 (0.104)
β_3	2.0	1.93 (0.095)
β_4	2.0	2.06 (0.126)
β_5	2.0	2.00 (0.103)
β_6	2.0	2.10 (0.105)
β_7	2.0	1.87 (0.110)
β_8	2.0	1.86 (0.089)
β_9	2.0	1.89 (0.093)
β_{10}	2.0	1.94 (0.091)
β_{11}	2.0	2.01 (0.171)
β_{12}	2.0	1.98 (0.137)
β_{13}	2.0	2.04 (0.108)
β_{14}	2.0	2.10 (0.107)
β_{15}	2.0	1.86 (0.103)

Points N	$\widehat{\mathbb{V}\mathrm{ar}}[\log \widehat{p}_N(\boldsymbol{x} \mid \boldsymbol{\theta})]$
300	0.116

Iterations	Acc. Prob.
20000	0.2871

Figure 4.4: Posterior density estimations for all the coordinates of one-factor copula model with $J = 15$ Bernoulli margins (the true values are indicated by red vertical lines and the posterior mean estimates are indicated by black vertical lines).



From the table and figure above, we can see that all the posterior mean estimates are very close to their corresponding true values. The posterior density estimates are created by MATLAB® built-in function `ksdensity`. However, the accuracy of the posterior mean estimates is not as good as in the case with Poisson margins since the fact that Bernoulli random variables contain less information makes the estimation more challenging. In this experiment, we need to use more particles than in the previous example to ensure the variance of log-likelihood estimator below 1 due to more observations being used. Each generated Markov chain consists of 20000 iterations with first 5000 iterates being used as Burn-in stage. The simulation achieves an overall acceptance probability of 0.2871 which is regarded as around optimal in high dimensions.

Figure 4.5: Trace plots for all the coordinates of one-factor copula model with $J = 15$ Bernoulli margins.

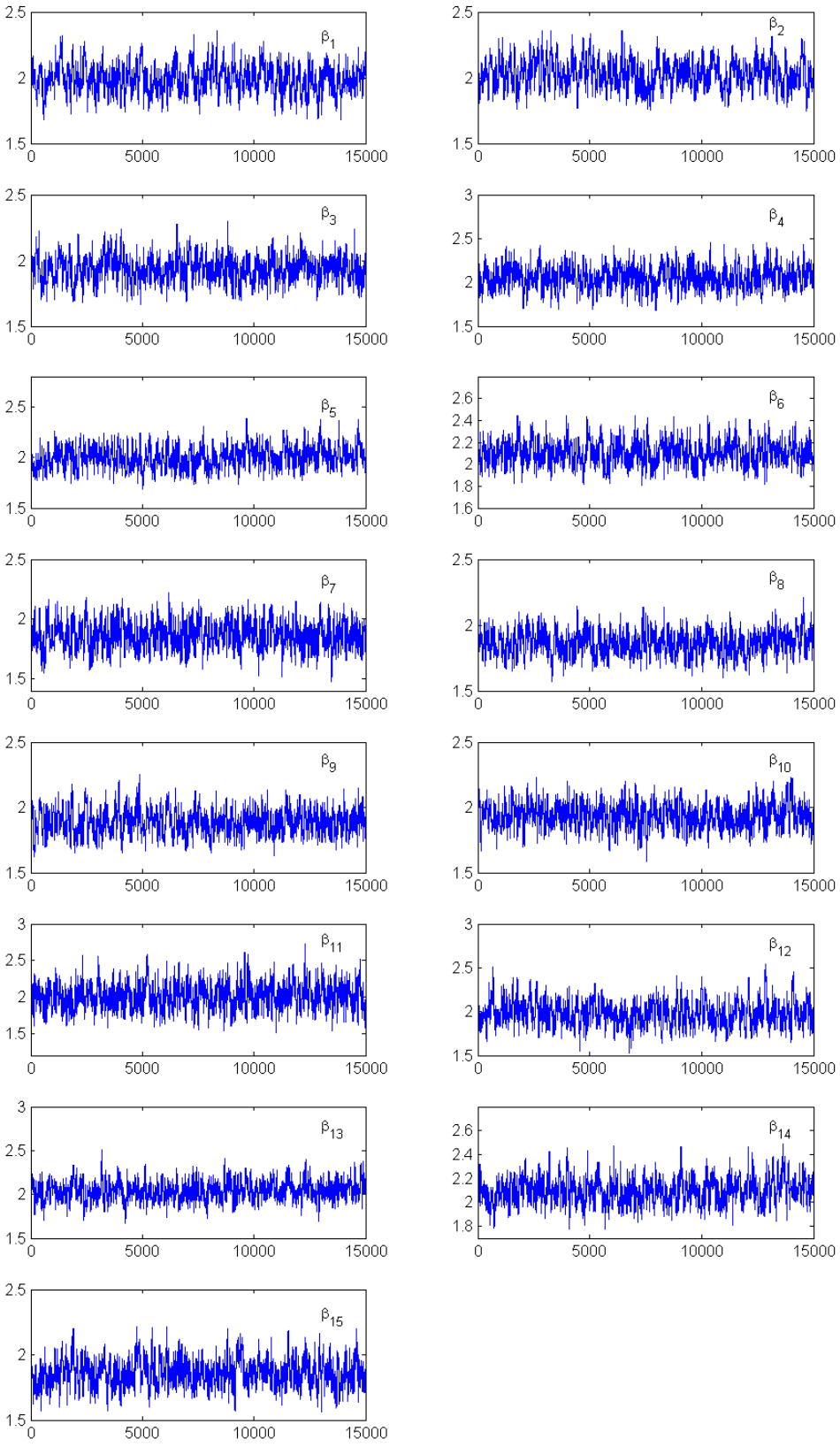
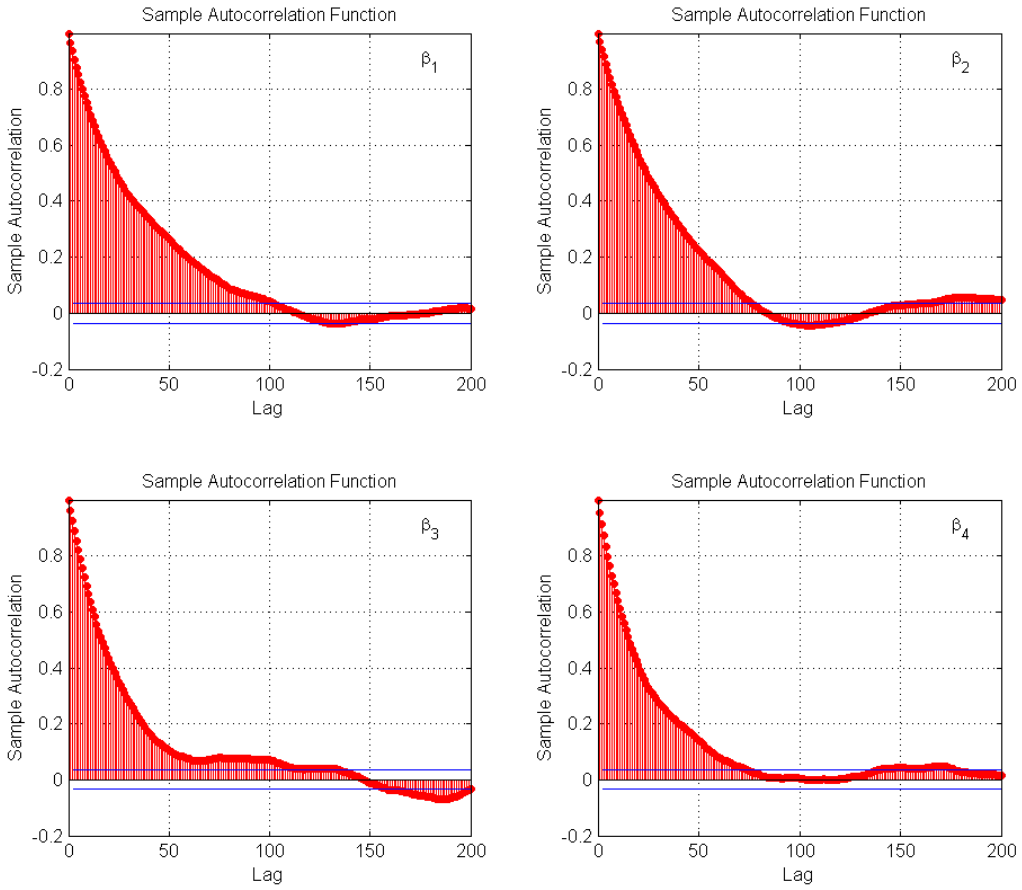


Figure 4.6: Sample autocorrelation functions for the first 4 coordinates of one-factor copula model with $J = 15$ Bernoulli margins.



The sample autocorrelation function is only plotted for the first 4 coordinates, and the rest are all similar. From the trace plots and sample autocorrelation functions above, we can see that the mixing of generated Markov chains becomes poorer but still acceptable as dimensionality of the estimation increases. For all the coordinates, the sample autocorrelations are cut off at around lag 80 to lag 100.

4.2 Gaussian Copula with Mixed Margins

In the pervious section, we discussed the Gaussian copula model with discrete margins. Moreover, the generalisation to the mixed margins is straightforward. More specifically, the vector $\boldsymbol{X}$ has both discrete and continuous marginal distributions. Without loss of generality, we suppose that $X_1, \ldots, X_r$ are discrete random variables and $X_{r+1}, \ldots, X_J$ are continuous. We further suppose that $\boldsymbol{Z} = \left(\boldsymbol{Z}_D^\top, \boldsymbol{Z}_C^\top\right)^\top \sim \mathbb{N}_J(\boldsymbol{0}, \Sigma)$ with Σ being partitioned as follows

$$\Sigma := \begin{bmatrix} \Sigma_{DD} & \Sigma_{DC} \\ \Sigma_{CD} & \Sigma_{CC} \end{bmatrix}$$

such that

$$\begin{aligned} \boldsymbol{\mu}_{D\,|\,C} &:= \mathbb{E}(\boldsymbol{Z}_D \,|\, \boldsymbol{Z}_C) \\ &= \Sigma_{DC}\Sigma_{CC}^{-1}\boldsymbol{Z}_C \\ \Sigma_{D\,|\,C} &:= \mathbb{V}\mathrm{ar}(\boldsymbol{Z}_D \,|\, \boldsymbol{Z}_C) \\ &= \Sigma_{DD} - \Sigma_{DC}\Sigma_{CC}^{-1}\Sigma_{CD} \end{aligned}$$

By following Smith and Khaled [2012], the distribution function of the Gaussian copula with mixed margins can be written as

$$p(\mathrm{d}\boldsymbol{x}) = p(\mathrm{d}\boldsymbol{x}_D \,|\, \boldsymbol{x}_C)\, p(\mathrm{d}\boldsymbol{x}_C) \tag{4.2.1}$$

where, analogously to the discrete copula case,

$$p(\mathrm{d}\boldsymbol{x}_D \,|\, \boldsymbol{x}_C) = \int_{\sqrt{\sigma_{11}}\Phi^{-1}(F_1(x_1^-))}^{\sqrt{\sigma_{11}}\Phi^{-1}(F_1(x_1))} \cdots \int_{\sqrt{\sigma_{rr}}\Phi^{-1}(F_r(x_r^-))}^{\sqrt{\sigma_{rr}}\Phi^{-1}(F_r(x_r))} \phi_r(\boldsymbol{z}_D \,|\, \boldsymbol{\mu}_{D\,|\,C},\, \Sigma_{D\,|\,C})\, \mathrm{d}\boldsymbol{z}_D \tag{4.2.2}$$

where $F_j(x_j^-) = \lim_{s \uparrow x_j} F_j(s)$, $\Phi^{-1}(\cdot)$ is the inverse of standard normal cumulative distribution function in one dimension and $\phi_r(\boldsymbol{z} \,|\, \boldsymbol{\mu}, \Sigma)$ denotes an r-variate normal density function with mean vector $\boldsymbol{\mu}$ and covariance matrix Σ. Also, we have

$$p(\mathrm{d}\boldsymbol{x}_C) = \phi_{J-r}(\boldsymbol{z}_C \,|\, \boldsymbol{0}, \Sigma_{CC}) \times \prod_{j=r+1}^{J} \frac{\sqrt{\sigma_{jj}}\, p_j(x_j)}{\phi(\Phi^{-1}(F_j(x_j)))} \tag{4.2.3}$$

where $\phi(\cdot)$ denotes the density of univariate standard normal distribution, $p_j(\cdot)$ for $j = r+1, \ldots, J$ is the density function of j-th random variable and $\boldsymbol{z}_C = (z_{r+1}, \ldots, z_J)^\top$ with $z_j = \sqrt{\sigma_{jj}}\, \Phi^{-1}(F_j(x_j))$ for $j = r+1, \ldots, J$.

The likelihood function (4.2.1) is also computationally intractable. However, it can be unbiasedly estimated as in the case with discrete margins, since (4.2.2) can be estimated unbiasedly by employing the Genz algorithm and (4.2.3) can be computed exactly.

4.2.1 Simulation - 10 *Mixed Margins*

In this simulation example, we also use a one-factor copula model ($k = 1$) but with $J = 10$ mixed margins which are specified below.

- The first 6 variables are Poisson distributed with parameters:

$$\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_6)^\top = (1.0, 1.5, 2.0, 2.5, 3.0, 3.5)^\top.$$

- Variable 7 and variable 8 are distributed according to $\mathsf{Exp}(1.5)$ and $\mathsf{Exp}(0.5)$ respectively, where Exp represents an exponential distribution.
- Variable 9 and variable 10 are $\mathbb{N}(0, 1)$ and $\mathbb{N}(2, 0.5)$ distributed respectively, where $\mathbb{N}(\mu, \sigma^2)$ is the normal distribution with mean μ and variance σ^2.

Thus, this is a mixture of 6 discrete margins and 4 continuous margins and we set the true values of the loading matrix B at

$$\mathrm{B} = (\beta_1, \ldots, \beta_{10})^\top = 0.5 \times \mathbf{1}_{10},$$

where $\mathbf{1}_{10}$ denotes a 10-dimensional column vector of 1s. We generated $T = 1000$ observations for this numerical experiment by Gaussian copula one-factor model with covariance matrix Σ formed by $\mathrm{BB}^\top + \mathrm{I}_{10}$.

As in the previous examples, the parameters of the marginal distributions are set at their true values since we wish to focus on how well the joint density is estimated. The starting points for all the coordinates are arbitrarily selected at 1.00.

Results and Comments:

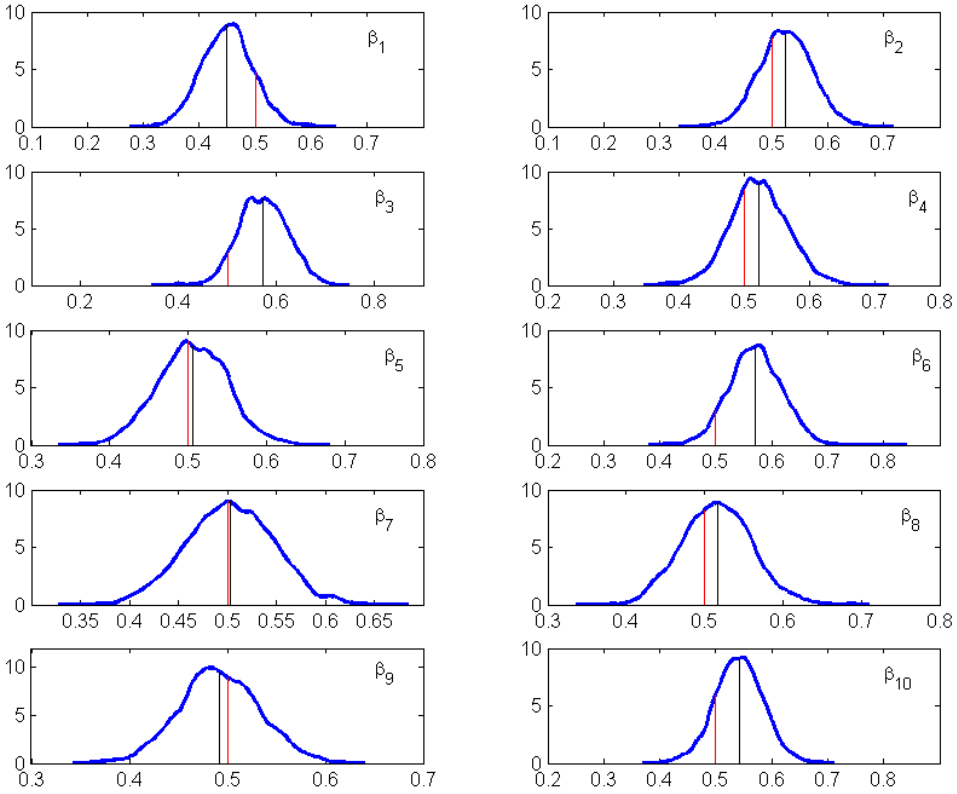
Table 4.3: Posterior mean estimates with posterior standard deviations reported in brackets for the one-factor copula model with $J = 10$ mixed margins.

Parameter	True Value	Estimate
β_1	0.5	0.45 (0.045)
β_2	0.5	0.53 (0.047)
β_3	0.5	0.57 (0.049)
β_4	0.5	0.52 (0.044)
β_5	0.5	0.51 (0.044)
β_6	0.5	0.57 (0.046)
β_7	0.5	0.50 (0.044)
β_8	0.5	0.52 (0.044)
β_9	0.5	0.49 (0.041)
β_{10}	0.5	0.54 (0.043)

Points N	$\widehat{\mathbb{V}\mathrm{ar}}[\log \widehat{p}_N(\boldsymbol{x} \mid \boldsymbol{\theta})]$
30	0.0155

Iterations	Acc. Prob.
20000	0.29

Figure 4.7: Posterior density estimations for all the coordinates of one-factor copula model with $J = 10$ mixed margins (the true values are indicated by red vertical lines and the posterior mean estimates are indicated by black vertical lines).



From the table and figure above, we can see that all the posterior mean estimates are very close to their associated true values. The posterior density estimates are obtained by MATLAB® built-in function `ksdensity`. We used 30 points to construct an unbiased estimator of likelihood function with estimated variance below 1. Each generated Markov chain contains 20000 iterations with first 5000 iterates being considered as Burin-in period. The simulation achieves an overall acceptance rate of 0.29 which is regarded as optimal in high dimensions by Roberts et al. [1997].

Figure 4.8: Trace plots for all the coordinates of one-factor copula model with $J = 10$ mixed margins.

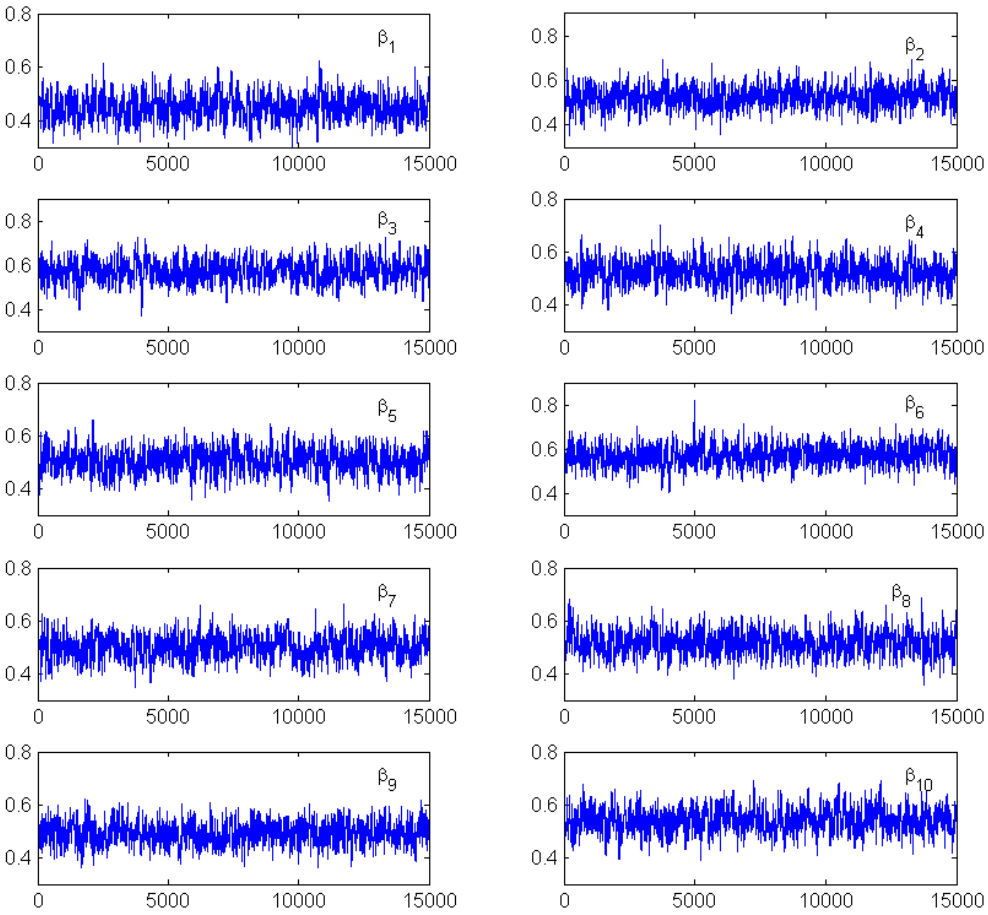
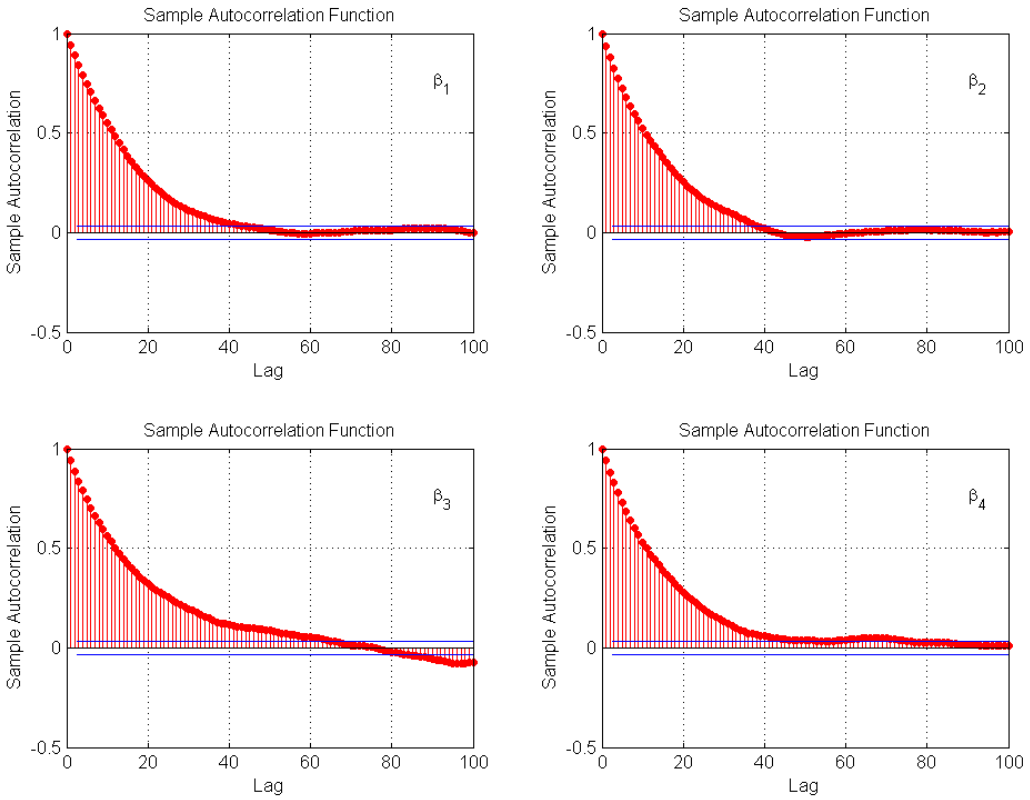


Figure 4.9: Sample autocorrelation functions for the first 4 coordinates of one-factor copula model with $J = 10$ mixed margins.



The sample autocorrelation function is only plotted for the first 4 coordinates, and the rest are all similar. For all of the coordinates, the sample autocorrelations are cut off at around lag 40 to lag 60. The trace plots and sample autocorrelation functions above confirm that all the generated Markov chains have good mixing in a relatively high dimensions.

Chapter 5

Conclusion and Further Work

Intractable likelihood function makes the Metropolis-Hastings (MH) algorithm infeasible. The pseudo-marginal Metropolis-Hastings (PMMH) approach overcomes this problem by working with an unbiased and non-negative estimator of the likelihood. Pitt et al. [2012] and Doucet et al. [2015] suggest that the number of particles has to be selected such that the variance of the log-likelihood estimator should be around 1 in order to obtain an optimal trade-off between the efficiency of the generated Markov chain and the computational cost. We apply the PMMH method to Bayesian estimation of high-dimensional copula model with discrete and mixed margins. Estimating copula model with discrete margins is challenging, because evaluation of the likelihood function requires computing 2^J terms (J denotes the number of discrete margins) and each of which is also computationally expensive. The numerical experiments confirm the competence of the PMMH method in this new application area.

Further studies include two parts. First, we are interested in applying the PMMH method to estimations of some Archimedean copulas since non-Gaussian copula is more popular in practice, especially in financial industry. Second, it would be interesting to implement the recently proposed correlated pseudo-marginal method (Deligiannidis et al. [2016]) by considering both of pseudo-random numbers and randomised quasi-random numbers.

References

Christophe Andrieu and Gareth O Roberts. The pseudo-marginal approach for efficient Monte Carlo computations. *The Annals of Statistics*, pages 697–725, 2009.

Christophe Andrieu and Matti Vihola. Convergence properties of pseudo-marginal Markov chain Monte Carlo algorithms. *The Annals of Applied Probability*, 25(2): 1030–1077, 2015.

Christophe Andrieu, Arnaud Doucet, and Roman Holenstein. Particle Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(3):269–342, 2010.

Mark A Beaumont. Estimation of population growth or decline in genetically monitored populations. *Genetics*, 164(3):1139–1160, 2003.

JE Besag and JC York. Bayesian restoration of images. *Analysis of Statistical Information (T. Matsunawa, ed.)*, pages 491–507, 1989.

George Deligiannidis, Arnaud Doucet, Michael K Pitt, and Robert Kohn. The correlated pseudo-marginal method. *arXiv preprint arXiv:1511.04992*, 2016.

Arnaud Doucet, MK Pitt, George Deligiannidis, and Robert Kohn. Efficient implementation of Markov chain Monte Carlo when using an unbiased likelihood estimator. *Biometrika*, page asu075, 2015.

Paul H Garthwaite, Yanan Fan, and Scott A Sisson. Adaptive optimal scaling of Metropolis-Hastings algorithms using the Robbins-Monro process. *Communications in Statistics-Theory and Methods*, (just-accepted), 2015.

Andrew Gelman, John B Carlin, Hal S Stern, and Donald B Rubin. *Bayesian data analysis*, volume 2. Chapman & Hall/CRC Boca Raton, FL, USA, 2014.

Stuart Geman and Donald Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6):721–741, 1984.

Alan Genz. Numerical computation of multivariate normal probabilities. *Journal of computational and graphical statistics*, 1(2):141–149, 1992.

John Geweke and Guofu Zhou. Measuring the pricing error of the arbitrage pricing theory. *Review of Financial Studies*, 9(2):557–587, 1996.

Joyee Ghosh and David B Dunson. Default prior distributions and efficient posterior computation in Bayesian factor analysis. *Journal of Computational and Graphical Statistics*, 18(2):306–320, 2009.

D Gunawan, M-N Tran, K Suzuki, J Dick, and R Kohn. Computationally efficient Bayesian estimation of high dimensional copulas with discrete and mixed margins. *arXiv preprint arXiv:1608.06174*, 2016a.

David Gunawan, Mohamad A Khaled, and Robert Kohn. Mixed marginal copula modeling. *arXiv preprint arXiv:1605.09101*, 2016b.

W Keith Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

L Lin, KF Liu, and J Sloan. A noisy Monte Carlo algorithm. *Physical Review D*, 61(7):074505, 2000.

Hedibert Freitas Lopes and Mike West. Bayesian model assessment in factor analysis. *Statistica Sinica*, pages 41–67, 2004.

Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

Jared S Murray, David B Dunson, Lawrence Carin, and Joseph E Lucas. Bayesian Gaussian copula factor models for mixed data. *Journal of the American Statistical Association*, 108(502):656–665, 2013.

Michael Pitt, David Chan, and Robert Kohn. Efficient Bayesian inference for Gaussian copula regression models. *Biometrika*, 93(3):537–554, 2006.

Michael K Pitt, Ralph dos Santos Silva, Paolo Giordani, and Robert Kohn. On some properties of Markov chain Monte Carlo simulation methods based on the particle filter. *Journal of Econometrics*, 171(2):134–151, 2012.

Gareth O Roberts, Andrew Gelman, Walter R Gilks, et al. Weak convergence and optimal scaling of random walk Metropolis algorithms. *The annals of applied probability*, 7(1):110–120, 1997.

Gareth O Roberts, Jeffrey S Rosenthal, et al. Optimal scaling for various Metropolis-Hastings algorithms. *Statistical science*, 16(4):351–367, 2001.

Sheldon M Ross. *Introduction to probability models*. Academic press, 2014.

Pavel V Shevchenko. *Modelling operational risk using Bayesian inference*. Springer Science & Business Media, 2011.

M Sklar. *Fonctions de répartition à n dimensions et leurs marges*. Université Paris 8, 1959.

Michael S Smith and Mohamad A Khaled. Estimation of copula models with discrete margins via Bayesian data augmentation. *Journal of the American Statistical Association*, 107(497):290–303, 2012.

M-N Tran, R Kohn, M Quiroz, and M Villani. Block-wise pseudo-marginal Metropolis-Hastings. *arXiv preprint arXiv:1603.02485*, 2016.

Pravin K Trivedi and David M Zimmer. *Copula modeling: an introduction for practitioners*. Now Publishers Inc, 2007.

Appendix A

Markov Chain

In this appendix, we deliver some basic definitions and theorems of Markov chain, which are essential knowledge to the thesis work. We focus on the discrete-time Markov chain and the following materials can be found in Ross [2014, Chapter 4].

Definition A.1 (Stochastic Process)**.** *A stochastic process $\mathcal{X} = \{X(t) : t \in T\}$ is an infinite collection of random variables defined on the same sample space, indexed by an integer or a real number.*

Thus, for each fixed $t \in T$, $X(t)$ is a random variable. The index t is usually interpreted as *time*, so $X(t)$ is the state of the process at time t. The set T is called the *index set* of the process. When T is a countable set, $\mathcal{X}$ is said to be a *discrete-time* process, and we use the notation

$$\{X_n,\, n = 0, 1, 2, \ldots\}.$$

When T is an interval on the real line, $\mathcal{X}$ is said to be a *continuous-time* process, denoted

$$\{X(t),\, t \geqslant 0\}.$$

Definition A.2. *A discrete-time Markov chain is*

1. *a discrete-time stochastic process $\{X_n,\, n = 0, 1, 2, \ldots\}$,*
2. *that takes on a finite or countable number of possible values (discrete state space S_X) called states and*
3. *satisfying the Markov property: for any $i, j, i_0, i_1, \ldots, i_{n-1} \in S_X$,*

$$\mathbb{P}(X_{n+1} = j \,|\, X_n = i, X_{n-1} = i_{n-1}, \ldots, X_1 = i_1, X_0 = i_0) = \mathbb{P}(X_{n+1} = j \,|\, X_n = i).$$

The Markov property indicates that given the present state (time n), further information about the past (times $0, 1, \ldots, n-1$) is irrelevant for comprehending the future (time $n+1$). Here, we assume the Markov chain is homogeneous which means the *transition probabilities* do not depend on the time index n,

$$\mathbb{P}(X_{n+1} = j \,|\, X_n = i) = P_{ij}, \qquad \forall i, j \in S_X.$$

The transition probability P_{ij} represents the probability that the process will, when in state i, next make the transition to state j.

Theorem A.1 (Chapman-Kolmogorov Equations)**.** *For all $n, m \geqslant 0$ and $i, j \geqslant 0$, we have*

$$P^{[n+m]} = \sum_{k \in S_X} P_{ik}^{[n]} P_{kj}^{[m]}.$$

With $P^{[n]}$ the matrix of n-step transition probabilities $P_{ij}^{[n]}$, this is

$$P^{[n+m]} = P^{[n]} P^{[m]}.$$

In particular, $P^{[2]} = P^{[1]} P^{[1]} = PP = P^2$ and by induction we have

$$P^{[n]} = P^n.$$

Therefore, Chapman-Kolmogorov Equations suggest that the n-step transition matrix is the n-th power of the one-step transition matrix. Note that $P^{[0]} = P^0 = \mathrm{I}$, where I denotes the identity matrix.

Definition A.3 (Accessible State)**.** *State j is said to be accessible from state i, denoted by $i \to j$, if*

$$[P^n]_{ij} = P_{ij}^{[n]} > 0$$

for some $n \geqslant 0$.

Definition A.4 (Communicating States)**.** *Two states i and j communicate, denoted $i \leftrightarrow j$, if i is accessible from j and j is accessible from i ($i \to j$ and $j \to i$).*

Definition A.5 (Communication Class)**.** *A class $\mathcal{C}$ is a non-empty set of states such that for each state $i \in \mathcal{C}$, i communicates with all $j \in \mathcal{C}$ and does not communicate with any $i \notin \mathcal{C}$.*

Definition A.6 (Irreducible Chain)**.** *A Markov chain is said to be irreducible if there is only one class, that is, if all states communicate with each other.*

For any state, let p_i denotes the probability that, starting in state i, the process will ever re-enter the state i.

Definition A.7 (States Classification)**.** *State i is said to be*

1. *recurrent if $p_i = 1$, or*
2. *transient if $p_i < 1$.*

Proposition A.1. *State i is recurrent if*

$$\sum_{n=0}^{\infty} P_{ii}^{[n]} = \sum_{n=0}^{\infty} [P^n]_{ii} = \infty$$

and transient if

$$\sum_{n=0}^{\infty} P_{ii}^{[n]} = \sum_{n=0}^{\infty} [P^n]_{ii} < \infty.$$

Theorem A.2. *In a Markov chain, either all states in a given class are recurrent or all are transient.*

Definition A.8 (Aperiodic State)**.** *The period of a state i, denoted by $d(i)$, is the greatest common divisor of those values of n for which $P_{ii}^{[n]} > 0$. If the period of a state is 1, then the state is aperiodic.*

Theorem A.3. *In a Markov chain, all states in a given class have the same period.*

Let T_{ij} be the number of transitions until a chain starting in state i enters state j for the first time. So T_{ii} is the number of transitions until a chain starting in state i returns to state i.

- if state i is transient, then $\mathbb{E}(T_{ii}) = \infty$, as the chain will never be back with positive probability.
- if state i is recurrent, then $\mathbb{E}(T_{ii}) < \infty$ or $\mathbb{E}(T_{ii}) = \infty$.

Definition A.9. *A state i of a Markov chain is positive-recurrent if it is recurrent and $\mathbb{E}(T_{ii}) < \infty$. It is null-recurrent if it is recurrent and $\mathbb{E}(T_{ii}) = \infty$.*

Definition A.10 (Ergodic Chain)**.** *A state is said to be ergodic if it is positive-recurrent and aperiodic. A class of ergodic states is an ergodic class. An irreducible Markov chain consisting of one ergodic class is an ergodic chain.*

Definition A.11 (Invariant Distribution)**.** *Suppose X is a Markov chain with state space S_X and transition probability matrix P. If $\boldsymbol{\pi} = (\pi_j,\, j \in S_X)$ is a distribution over S_X (that is, $\boldsymbol{\pi}$ is a row vector with $|S_X|$ components such that $\sum_j \pi_j = 1$ and $\pi_j \geqslant 0$ for all $j \in S_X$) and satisfies*

$$\boldsymbol{\pi} P = \boldsymbol{\pi},$$

that is,

$$\pi_j = \sum_{i \in S_X} \pi_i \, P_{ij}, \quad \forall\, j \in S_X,$$

where π_j is the dot product between $\boldsymbol{\pi}$ and the j-th column of P, then $\boldsymbol{\pi}$ is an invariant distribution of the Markov chain X.

Theorem A.4. *For an ergodic Markov chain, there exists a unique invariant distribution $\boldsymbol{\pi}$ such that*

$$\lim_{n \to \infty} P_{ij}^{[n]} = \pi_j$$

for all i, $j \in S_X$ and is independent of state i.

Definition A.12 (Reversible Transition)**.** *A Markov transition probability matrix P satisfying the reversible property*

$$\pi_i P_{ij} = \pi_j P_{ji}, \quad \forall i, j \in S_X,$$

for some distribution $\boldsymbol{\pi}$ is called a reversible transition with respect to the distribution $\boldsymbol{\pi}$.

Proposition A.2. *If a Markov transition probability matrix P is a reversible transition with respect to some probability distribution $\boldsymbol{\pi}$, then $\boldsymbol{\pi}$ is the invariant distribution of the Markov chain with Markov transition matrix P.*

Appendix B

Proof of Theorem 2.2.1

In this appendix, we provide a detailed proof of theorem 2.2.1, which also explains why the Metropolis - Hastings algorithm works. The following proof is adapted from Gelman et al. [2014, Chapter 11.2].

The proof consists of two parts: (1) simulated sequence is a Markov chain with a unique invariant distribution, (2) the invariant distribution needs to be equal to $\pi(\boldsymbol{\theta} \mid \boldsymbol{x})$ which is our target density. The first part holds if the Markov chain is irreducible, aperiodic and not transient. The latter two conditions hold for a random walk on any proper distribution. Moreover, the irreducibility holds if the random walk has positive probability of eventually reaching any state from any other state. This is true for all sensible proposal densities we could use. It remains to be shown that the invariant distribution is indeed $\pi(\boldsymbol{\theta} \mid \boldsymbol{x})$.

In Metropolis - Hastings algorithm, the proposal density $q(\cdot \mid \cdot)$ is not required to be symmetric. To correct this asymmetry in the proposal distribution, the acceptance probability is defined as

$$\alpha(\boldsymbol{\theta}^{(i-1)}, \boldsymbol{\theta}^*) = \min\left\{1,\ \frac{\pi(\boldsymbol{\theta}^* \mid \boldsymbol{x})\, q(\boldsymbol{\theta}^{(i-1)} \mid \boldsymbol{\theta}^*)}{\pi(\boldsymbol{\theta}^{(i-1)} \mid \boldsymbol{x})\, q(\boldsymbol{\theta}^* \mid \boldsymbol{\theta}^{(i-1)})}\right\}.$$

Now, in order to prove that the invariant density is the required target density, we consider starting the chain at step $i-1$ and two points $\boldsymbol{\theta}_a$ and $\boldsymbol{\theta}_b$ such that let's say $\pi(\boldsymbol{\theta}_b \mid \boldsymbol{x})\, q(\boldsymbol{\theta}_a \mid \boldsymbol{\theta}_b) \geqslant \pi(\boldsymbol{\theta}_a \mid \boldsymbol{x})\, q(\boldsymbol{\theta}_b \mid \boldsymbol{\theta}_a)$. Then,

$$\begin{aligned}
&\mathbb{P}(\text{to be at } \boldsymbol{\theta}_a \text{ and to move to } \boldsymbol{\theta}_b \text{ over steps } (i-1) \to (i)) \\
&= \mathbb{P}(\boldsymbol{\theta}^{(i-1)} = \boldsymbol{\theta}_a,\ \boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}_b) \\
&= \pi(\boldsymbol{\theta}_a \mid \boldsymbol{x})\, q(\boldsymbol{\theta}_b \mid \boldsymbol{\theta}_a)\ \min\left\{1,\ \frac{\pi(\boldsymbol{\theta}_b \mid \boldsymbol{x})\, q(\boldsymbol{\theta}_a \mid \boldsymbol{\theta}_b)}{\pi(\boldsymbol{\theta}_a \mid \boldsymbol{x})\, q(\boldsymbol{\theta}_b \mid \boldsymbol{\theta}_a)}\right\} \\
&= \pi(\boldsymbol{\theta}_a \mid \boldsymbol{x})\, q(\boldsymbol{\theta}_b \mid \boldsymbol{\theta}_a).
\end{aligned}$$

Similarly, we have

$$
\begin{aligned}
&\mathbb{P}(\text{to be at } \boldsymbol{\theta}_b \text{ and to move to } \boldsymbol{\theta}_a \text{ over steps } (i-1) \to (i)) \\
&= \mathbb{P}(\boldsymbol{\theta}^{(i-1)} = \boldsymbol{\theta}_b,\, \boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}_a) \\
&= \pi(\boldsymbol{\theta}_b \,|\, \boldsymbol{x})\, q(\boldsymbol{\theta}_a \,|\, \boldsymbol{\theta}_b)\, \min\left\{1,\, \frac{\pi(\boldsymbol{\theta}_a \,|\, \boldsymbol{x})\, q(\boldsymbol{\theta}_b \,|\, \boldsymbol{\theta}_a)}{\pi(\boldsymbol{\theta}_b \,|\, \boldsymbol{x})\, q(\boldsymbol{\theta}_a \,|\, \boldsymbol{\theta}_b)}\right\} \\
&= \pi(\boldsymbol{\theta}_b \,|\, \boldsymbol{x})\, q(\boldsymbol{\theta}_a \,|\, \boldsymbol{\theta}_b)\, \frac{\pi(\boldsymbol{\theta}_a \,|\, \boldsymbol{x})\, q(\boldsymbol{\theta}_b \,|\, \boldsymbol{\theta}_a)}{\pi(\boldsymbol{\theta}_b \,|\, \boldsymbol{x})\, q(\boldsymbol{\theta}_a \,|\, \boldsymbol{\theta}_b)} \\
&= \pi(\boldsymbol{\theta}_a \,|\, \boldsymbol{x})\, q(\boldsymbol{\theta}_b \,|\, \boldsymbol{\theta}_a).
\end{aligned}
$$

Hence, we get the following equality

$$
\mathbb{P}(\boldsymbol{\theta}^{(i-1)} = \boldsymbol{\theta}_a,\, \boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}_b) = \mathbb{P}(\boldsymbol{\theta}^{(i-1)} = \boldsymbol{\theta}_b,\, \boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}_a)
$$

which means the joint distribution of two consecutive samples is symmetric. Therefore, $\boldsymbol{\theta}^{(i-1)}$ and $\boldsymbol{\theta}^{(i)}$ have the same marginal distributions and thus $\pi(\boldsymbol{\theta} \,|\, \boldsymbol{x})$ is the invariant distribution of the Markov chain $\left\{\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}, \ldots, \boldsymbol{\theta}^{(M)}\right\}$.

Appendix C

Copula Model

In this appendix, we state some basic knowledge related to copula model, which is helpful for the reader to understand this thesis. Moreover, we provide several common examples of copula models. Finally, we give a detailed explanation for the difference operator notation used throughout the thesis. The following materials are collected from Shevchenko [2011, Chapter 7], Trivedi and Zimmer [2007, Chapter 2] and Gunawan et al. [2016b].

Copula models are popular in high-dimensional statistical applications as they allow us to easily model the dependence structure between multiple random variables and estimate the distribution of random vectors. A copula is a multivariate distribution function whose univariate marginals are uniformly distributed on $[0, 1]$. A J-dimensional copula (or J-copula for short) is a function $C : [0, 1]^J \to [0, 1]$ which satisfies the following conditions:

1. $C(1, \ldots, 1, a_n, 1, \ldots, 1) = a_n$ for every $n \leqslant J$ and all a_n in $[0, 1]$;
2. $C(a_1, a_2, \ldots, a_J) = 0$ if $a_n = 0$ for any $n \leqslant J$;
3. C is J-increasing, that is, for each hyper-rectangle $B = \prod_{i=1}^{J}[x_i, y_i] \subseteq [0, 1]^J$ the C-volume of B is non-negative.

Condition 3 says that the C-volume of any J-dimensional interval is non-negative. For instance, in the bivariate case $J = 2$, these three conditions become:

1. $C(u, 1) = C(1, u) = u$ for all $u \in [0, 1]$;
2. $C(0, u) = C(u, 0) = 0$ for all $u \in [0, 1]$;
3. for all pairs $(u_1, u_2), (v_1, v_2) \in [0, 1] \times [0, 1]$ with $u_1 \leqslant v_1$ and $u_2 \leqslant v_2$:

$$C(v_1, v_2) - C(v_1, u_2) - C(u_1, v_2) + C(u_1, u_2) \geqslant 0.$$

The following well-known theorem links the multivariate distribution to its marginals.

Theorem C.1 (Sklar's Theorem)**.** *Given a copula function $C(u_1, \ldots, u_J)$, the joint cumulative distribution function of random variables $Y_1, \ldots, Y_J$ with marginal distributions $F_1(y_1), \ldots, F_J(y_J)$ can be constructed as*

$$F(y_1, \ldots, y_J) = C(F_1(y_1), \ldots, F_J(y_J)).$$

The theorem indicates that one can always find a unique copula C for a joint distribution with given continuous marginals. Note that in the case of discrete distributions, this copula may not be unique. By talking derivatives, the joint density can be written as

$$f(y_1, \ldots, y_J) = c(F_1(y_1), \ldots, F_J(y_J)) \prod_{i=1}^{J} f_i(y_i),$$

where

$$c(u_1, u_2, \ldots, u_J) = \frac{\partial^J}{\partial u_1 \partial u_2 \cdots \partial u_J} C(u_1, u_2, \ldots, u_J)$$

is the density of the copula. This relation clearly shows that the contribution to the joint density comes from two parts: one that comes from the copula and is "responsible" for the dependence ($c(u_1, u_2, \ldots, u_J)$) and another one which takes into account marginal information only ($\prod_{i=1}^{J} f_i(y_i)$). It is also clear that the independence implies that the corresponding copula is $C(u_1, u_2, \ldots, u_J) = u_1 u_2 \cdots u_J$.

There are many different copulas discussed in the literature and we provide some common examples below.

C.1 Gaussian Copula

The J-dimensional Gaussian copula is obtained by transformation of the multivariate normal distribution

$$C(u_1, \ldots, u_J) = F_N^{\Sigma}(F_N^{-1}(u_1), \ldots, F_N^{-1}(u_J))$$

and its density is

$$c(u_1, \ldots, u_J) = \frac{f_N^{\Sigma}(F_N^{-1}(u_1), \ldots, F_N^{-1}(u_J))}{\prod_{i=1}^{J} f_N(F_N^{-1}(u_i))}.$$

Here, $F_N(\cdot)$ and $f_N(\cdot)$ are the standard normal distribution and its density function respectively; F_N^{Σ} and f_N^{Σ} are the standard multivariate normal cumulative distribution function and its density respectively with zero means, unit variances and correlation matrix Σ. This is "The formula that killed Wall Street". When $\Sigma = \mathrm{I}_J$, we get $C(u_1, \ldots, u_J) = u_1 \cdots u_J$ as expected.

C.2 Archimedean Copulas

Non-Gaussian copulas are much more important in practice, since Gaussian copulas do not allow us to model reasonably well the tail dependence, that is, joint extreme events have virtually a zero probability. Especially in financial applications, it is very important to be able to model dependence in the tails. The J-dimensional Archimedean copulas can be written as

$$C(u_1, \ldots, u_J) = \phi^{-1}(\phi(u_1) + \cdots + \phi(u_J)),$$

where ϕ is a continuous and strictly decreasing function from $[0, 1]$ in $[0, \infty)$ such that $\phi(1) = 0$ and is called *generator*. Two important members in this family are Clayton and Gumbel copulas.

C.2.1 Clayton Copula

Clayton copula has loads of lower tail dependence and no upper tail dependence. It is given by

$$C(u_1, \ldots, u_J) = \left(1 - J + \sum_{i=1}^{J} (u_i)^{-\rho}\right)^{-\frac{1}{\rho}}$$

and its density is

$$c(u_1, \ldots, u_J) = \left(1 - J + \sum_{i=1}^{J} (u_i)^{-\rho}\right)^{-J-\frac{1}{\rho}} \prod_{i=1}^{J} \left((u_i)^{-\rho-1}\{(i-1)\rho + 1\}\right),$$

where $\rho > 0$ is a dependence parameter. The generator and inverse generator for the Clayton copula are given by

$$\phi_C(t) = (t^{-\rho} - 1),$$
$$\phi_C^{-1}(s) = (1 + s)^{-\frac{1}{\rho}}.$$

C.2.2 Gumbel Copula

The Gumbel copula is much more flexible in modelling dependence in the upper tails. For an arbitrary dimension J, it is defined as

$$C(u_1, \ldots, u_J) = \exp\left\{ -\left(\sum_{i=1}^{J} (-\log u_i)^{\rho} \right)^{\frac{1}{\rho}} \right\}$$

where $\rho \geqslant 1$ is a parameter controls the strength of dependence and the density of the copula is

$$c(u_1, \ldots, u_J) = \frac{\partial^J}{\partial u_1 \cdots \partial u_J} C(u_1, \ldots, u_J).$$

In the bivariate case, the explicit expression for the density of Gumbel copula is given by

$$\begin{aligned} c(u_1, u_2) &= \frac{\partial^2}{\partial u_1 \, \partial u_2} C(u_1, u_2) \\ &= C(u_1, u_2)\, u_1^{-1} u_2^{-1} \left[\sum_{i=1}^{2} (-\log u_i)^{\rho} \right]^{2\left(\frac{1}{\rho}-1\right)} (\log u_1 \log u_2)^{\rho-1} \\ &\qquad \times \left[1 + (\rho - 1) \left[\sum_{i=1}^{2} (-\log u_i)^{\rho} \right]^{-\frac{1}{\rho}} \right]. \end{aligned}$$

The generator and inverse generator for the Gumbel copula are given by

$$\begin{aligned} \phi_G(t) &= (-\log t)^{\rho}, \\ \phi_G^{-1}(s) &= \exp\left(-s^{1/\rho}\right). \end{aligned}$$

The benefit of using the Archimedean copulas is that they allow for simple description of the J-dim dependence by using a function of one argument only (the generator). However, it is seen immediately that the Archimedean copula is symmetric in its arguments and this limits its applicability for modelling dependencies that are not symmetric in their arguments. The so-called *Liouville* copula is an extension of the Archimedean copula and can be used also to model dependencies that are not symmetric in their arguments.

C.3 Difference Operator Notations

In this thesis, we focus on the high-dimensional Gaussian copula with discrete and mixed margins. Their estimation procedures are computationally challenging since evaluation of the likelihood function requires computing 2^J terms, where J denotes the number of discrete margins. In order to illustrate this difficulty, we introduce the following difference operator which is borrowed from Gunawan et al. [2016b].

The difference operator notation $\triangle_a^b g_x(.)$ has two components:

1. When the operator $\triangle_a^b$ is applied to a function, the indices a and b are used to make the domain of the function clear.
2. A dot is marked at the position of the variables that are being differenced.

We provide some examples below to demonstrate the use of the difference operator notation.

- Consider a function $g(x)$ where x is a scalar. Then $\triangle_a^b g_x(.)$ defines

$$\triangle_a^b g_x(.) := g(b) - g(a)$$

- Consider a function $g(x, y)$ where both x and y are scalars. By $\triangle_a^b g_{x,y}(., z)$ we mean that the differencing is only applied to x while the second argument is fixed at $y = z$, that is

$$\triangle_a^b g_{x,y}(., z) := g(b, z) - g(a, z)$$

- Consider a function $g(\boldsymbol{x})$ where $\boldsymbol{x}$ is two-dimensional. By $\triangle_{\boldsymbol{a}}^{\boldsymbol{b}} g_{\boldsymbol{x}}(.)$, we mean

$$\begin{aligned}
\triangle_{\boldsymbol{a}}^{\boldsymbol{b}} g_{\boldsymbol{x}}(.) &= \triangle_{a_1}^{b_1} \triangle_{a_2}^{b_2} g_{\boldsymbol{x}}(.) \\
&= \triangle_{a_1}^{b_1} (g_{x_1,x_2}(., b_2) - g_{x_1,x_2}(., a_2)) \\
&= \triangle_{a_1}^{b_1} (g_{x_1,x_2}(., b_2)) - \triangle_{a_1}^{b_1} (g_{x_1,x_2}(., a_2)) \\
&= g(b_1, b_2) - g(a_1, b_2) - g(b_1, a_2) + g(a_1, a_2)
\end{aligned}$$

Appendix D

MATLAB® Code

This appendix contains all the programs we wrote for the numerical experiments. For instance, the simulation study of 10 Poisson margins contains

- `Copula_Data_Poisson.m`
- `PMMH_Copula_Poisson.m`

which implement

- the data generation steps
- the estimation by PMMH algorithm

respectively. Finally, we attach the program of the likelihood estimator (`qsilatmvnv.m`) which is borrowed from Genz [1992].

D.1 Simulation-10 Poisson Margins

D.1.1 Copula_Data_Poisson.m

```
% Simulated example of Bayesian estimation of high dimensional Gaussian
% Copula model with discrete margins by the PMMH algorithm, written by
% Xuebin Zheng (z3369961) for the Master Project (MATH5925).
% Supervisor: Associate Professor Josef Dick
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% << Data Generation Script (Poisson Margins) >>
clear all;
format long;
% Input the number of observations 'n' we want to generate.
n = 1000;
% Input the J by k loading matrix B where J is the dimension of the
% Copula and k denotes the number of factors in the model.
B = [0.806, 0.985, 0.997, 0.473, 0.650, 0.300, 0.530, 0.492, ...
```

```
        0.205, 0.713]';
J = size(B, 1);
% We use J 'Poisson margins' each has parameter specified below.
margin_par = [1.0, 3.0, 1.75, 2.5, 6.75, 5.45, 3.86, 4.15, 1.25, 8.0]';
% Now, we generate n observations by using Gaussian Copula.
mu = zeros(1, J);
% The loading matrix B is used to form the covariance matrix 'sigma'.
sigma = B * B' + eye(J);
Z = mvnrnd(mu, corrcov(sigma), n);
U = normcdf(Z, 0, 1);
% Pre-allocate the resulting data matrix 'X' which has dimension n by J.
% The resulting data matrix 'X' contains all of the simulated data with
% each row corresponds to one data point of J variables.
X = zeros(n, J);
for i = 1 : J
    X(:, i) = poissinv(U(:, i), margin_par(i, 1));
end
% Calculate the theoretical rank correlations of the data.
tauTheoretical = 2.* asin(corrcov(sigma)) ./ pi;
% Calculate the sample rank correlations of the data.
tauSample = corr(X, 'type', 'Kendall');
% Make sure the simulated data preserve correct dependence structure.

save('Copula_Data_Poisson.mat', 'X', 'tauTheoretical', 'tauSample');
```

D.1.2 PMMH_Copula_Poisson.m

```
% Simulated example of Bayesian estimation of high dimensional Gaussian
% Copula model with discrete margins by the PMMH algorithm, written by
% Xuebin Zheng (z3369961) for the Master Project (MATH5925).
% Supervisor: Associate Professor Josef Dick
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% << PMMH Algorithm for Bayesian Copula Model (Poisson Margins) >>
clear all;
format long;
load('Copula_Data_Poisson.mat');
load('cov_sim.mat');
```

```
[n, J] = size(X);
% Input the J by k loading matrix B where J is the dimension of the
% Copula and k denotes the number of factors in the model.
% B = [0.806, 0.985, 0.997, 0.473, 0.650, 0.300, 0.530, 0.492, ...
%        0.205, 0.713]';
% We use J 'Poisson margins' each has parameter specified below.
margin_par = [1.0, 3.0, 1.75, 2.5, 6.75, 5.45, 3.86, 4.15, 1.25, 8.0]';
% Prepare the upper and lower bounds used in estimating the
% likelihood function. The limits 'U' and 'L' are n by J matrices.
U = X;
L = X - 1;
for i = 1 : J
    U(:, i) = poisscdf(U(:, i), margin_par(i, 1));
    U(:, i) = norminv(U(:, i), 0, 1);
    L(:, i) = poisscdf(L(:, i), margin_par(i, 1));
    L(:, i) = norminv(L(:, i), 0, 1);
end
% Set the length of MCMC run 'nMCMC'.
nMCMC = 15000;
% Set the number of particles we want to use for estimating
% the likelihood.
nP = 30;
% Pre-allocate the output matrix (nMCMC by J).
beta = zeros(nMCMC, J);
% set initial values for the J parameters.
betaInit = 2 * ones(1, J);
beta(1, :) = betaInit;
% Calculate likelihood estimation for the first iteration.
sigma = betaInit' * betaInit + eye(J);
sigma_diag = diag(sigma);
U_current = zeros(n, J);
L_current = zeros(n, J);
U_prop = zeros(n, J);
L_prop = zeros(n, J);
parfor i = 1 : J
    U_current(:, i) = U(:, i) * sqrt(sigma_diag(i));
    L_current(:, i) = L(:, i) * sqrt(sigma_diag(i));
end
```

```
log_likeli_current = zeros(n, 1);
parfor j = 1 : n
    log_likeli_current(j, 1) = log(qsilatmvnv(nP, sigma, ...
            L_current(j, :)', U_current(j, :)'));
end
sum_log_likeli_current = sum(log_likeli_current);
% Calculate the posterior for the first iteration.
log_prior_current = log(mvnpdf(betaInit, zeros(1, J), eye(J)));
posterior_current = sum_log_likeli_current + log_prior_current;
% Define the variable 'count', which is used to calculate the average
% acceptance probability.
count = 0;
% Start MCMC run from the second iteration.
for ii = 2 : nMCMC
    % Specify the proposal distribution.
    beta_prop = mvnrnd(beta(ii - 1, :), 0.5 * cov_sim, 1);
    % Compute the proposed log-likelihood estimate.
    sigma_prop = beta_prop' * beta_prop + eye(J);
    sigma_diag_prop = diag(sigma_prop);
    parfor i = 1 : J
        U_prop(:, i) = U(:, i) * sqrt(sigma_diag_prop(i));
        L_prop(:, i) = L(:, i) * sqrt(sigma_diag_prop(i));
    end
    log_likeli_prop = zeros(n, 1);
    parfor j = 1 : n
        log_likeli_prop(j, 1) = log(qsilatmvnv(nP, sigma_prop, ...
                L_prop(j, :)', U_prop(j, :)'));
    end
    sum_log_likeli_prop = sum(log_likeli_prop);
    % Calculate the proposed value of posterior.
    log_prior_prop = log(mvnpdf(beta_prop, zeros(1, J), eye(J)));
    posterior_prop = sum_log_likeli_prop + log_prior_prop;
    % Calculate the acceptance probability.
    acc = min(1, exp(posterior_prop - posterior_current));
    % Decide to accept or reject the proposed values.
    u = rand(1);
    if u < acc
        beta(ii, :) = beta_prop;
```

```
        posterior_current = posterior_prop;
        count = count + 1;
    else
        beta(ii, :) = beta(ii - 1, :);
    end

end

% Compute the average acceptance probability.
AccProb = count / (nMCMC - 1);
% Posterior Mean with first 5000 iterations are used as burn-in.
PostMean = zeros(1, J);
parfor h = 1 : J
    PostMean(1, h) = mean(beta(5001:nMCMC, h));
end

save('PMMH_Copula_Poisson_cov_sim.mat', 'AccProb', 'PostMean', 'beta');

display(AccProb)
display(PostMean)
```

D.2 Simulation-15 Bernoulli Margins

D.2.1 Copula_Data_Bernoulli.m

```
% Simulated example of Bayesian estimation of high dimensional Gaussian
% Copula model with discrete margins by the PMMH algorithm, written by
% Xuebin Zheng (z3369961) for the Master Project (MATH5925).
% Supervisor: Associate Professor Josef Dick
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% << Data Generation Script (Bernoulli Margins) >>
clear all;
format long;
% Input the number of observations 'n' we want to generate.
n = 2000;
% Input the J by k loading matrix B where J is the dimension of the
% Copula and k denotes the number of factors in the model.
```

```
B = 2 * ones(15, 1);
J = size(B, 1);
% We use J 'Bernoulli margins' each has parameter specified below.
par_1 = 0.40; par_2 = 0.60; par_3 = 0.60; par_4 = 0.20; par_5 = 0.30;
par_6 = 0.40; par_7 = 0.20; par_8 = 0.40; par_9 = 0.35; par_10 = 0.45;
par_11 = 0.90; par_12 = 0.82; par_13 = 0.25; par_14 = 0.53;par_15 = 0.71;
margin_par = [par_1, par_2, par_3, par_4, par_5, par_6, par_7, par_8, ...
    par_9, par_10, par_11, par_12, par_13, par_14, par_15]';
% Now, we generate n observations by using Gaussian Copula.
mu = zeros(1, J);
% The loading matrix B is used to form the covariance matrix 'sigma'.
sigma = B * B' + eye(J);
Z = mvnrnd(mu, corrcov(sigma), n);
U = normcdf(Z, 0, 1);
% Pre-allocate the resulting data matrix 'X' which has dimension n by J.
% The resulting data matrix 'X' contains all of the simulated data with
% each row corresponds to one data point of J variables.
X = zeros(n, J);
for i = 1 : J
    X(:, i) = bsxfun(@le, U(:, i), margin_par(i, 1));
end
% Calculate the theoretical rank correlations of the data.
tauTheoretical = 2.* asin(corrcov(sigma)) ./ pi;
% Calculate the sample rank correlations of the data.
tauSample = corr(X, 'type', 'Kendall');
% Make sure the simulated data preserve correct dependence structure.

save('Copula_Data_Bernoulli.mat', 'X', 'tauTheoretical', 'tauSample');
```

D.2.2 PMMH_Copula_Bernoulli.m

```
% Simulated example of Bayesian estimation of high dimensional Gaussian
% Copula model with discrete margins by the PMMH algorithm, written by
% Xuebin Zheng (z3369961) for the Master Project (MATH5925).
% Supervisor: Associate Professor Josef Dick
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% << PMMH Algorithm for Bayesian Copula Model (Bernoulli Margins) >>
```

```
clear all;
format long;
load('Copula_Data_Bernoulli.mat');
load('cov_sim.mat');
[n, J] = size(X);
% Input the J by k loading matrix B where J is the dimension of the
% Copula and k denotes the number of factors in the model.
% B = 2 * ones(15, 1);
% We use J 'Bernoulli margins' each has parameter specified below.
par_1 = 0.40; par_2 = 0.60; par_3 = 0.60; par_4 = 0.20; par_5 = 0.30;
par_6 = 0.40; par_7 = 0.20; par_8 = 0.40; par_9 = 0.35; par_10 = 0.45;
par_11 = 0.90; par_12 = 0.82; par_13 = 0.25; par_14 = 0.53;par_15 = 0.71;
% Prepare the upper and lower bounds used in estimating the
% likelihood function. The limits 'U' and 'L' are n by J matrices.
U = zeros(n, J);
L = zeros(n, J);
for i = 1 : n
    if X(i, 1) == 0
        U(i, 1) = norminv(1 - par_1);
        L(i, 1) = -Inf;
    else
        U(i, 1) = Inf;
        L(i, 1) = norminv(1 - par_1);
    end

    if X(i, 2) == 0
        U(i, 2) = norminv(1 - par_2);
        L(i, 2) = -Inf;
    else
        U(i, 2) = Inf;
        L(i, 2) = norminv(1 - par_2);
    end

    if X(i, 3) == 0
        U(i, 3) = norminv(1 - par_3);
        L(i, 3) = -Inf;
    else
        U(i, 3) = Inf;
```

```
    L(i, 3) = norminv(1 - par_3);
end

if X(i, 4) == 0
    U(i, 4) = norminv(1 - par_4);
    L(i, 4) = -Inf;
else
    U(i, 4) = Inf;
    L(i, 4) = norminv(1 - par_4);
end

if X(i, 5) == 0
    U(i, 5) = norminv(1 - par_5);
    L(i, 5) = -Inf;
else
    U(i, 5) = Inf;
    L(i, 5) = norminv(1 - par_5);
end

if X(i, 6) == 0
    U(i, 6) = norminv(1 - par_6);
    L(i, 6) = -Inf;
else
    U(i, 6) = Inf;
    L(i, 6) = norminv(1 - par_6);
end

if X(i, 7) == 0
    U(i, 7) = norminv(1 - par_7);
    L(i, 7) = -Inf;
else
    U(i, 7) = Inf;
    L(i, 7) = norminv(1 - par_7);
end

if X(i, 8) == 0
    U(i, 8) = norminv(1 - par_8);
    L(i, 8) = -Inf;
```

```
else
    U(i, 8) = Inf;
    L(i, 8) = norminv(1 - par_8);
end

if X(i, 9) == 0
    U(i, 9) = norminv(1 - par_9);
    L(i, 9) = -Inf;
else
    U(i, 9) = Inf;
    L(i, 9) = norminv(1 - par_9);
end

if X(i, 10) == 0
    U(i, 10) = norminv(1 - par_10);
    L(i, 10) = -Inf;
else
    U(i, 10) = Inf;
    L(i, 10) = norminv(1 - par_10);
end

if X(i, 11) == 0
    U(i, 11) = norminv(1 - par_11);
    L(i, 11) = -Inf;
else
    U(i, 11) = Inf;
    L(i, 11) = norminv(1 - par_11);
end

if X(i, 12) == 0
    U(i, 12) = norminv(1 - par_12);
    L(i, 12) = -Inf;
else
    U(i, 12) = Inf;
    L(i, 12) = norminv(1 - par_12);
end

if X(i, 13) == 0
```

```
        U(i, 13) = norminv(1 - par_13);
        L(i, 13) = -Inf;
    else
        U(i, 13) = Inf;
        L(i, 13) = norminv(1 - par_13);
    end

    if X(i, 14) == 0
        U(i, 14) = norminv(1 - par_14);
        L(i, 14) = -Inf;
    else
        U(i, 14) = Inf;
        L(i, 14) = norminv(1 - par_14);
    end

    if X(i, 15) == 0
        U(i, 15) = norminv(1 - par_15);
        L(i, 15) = -Inf;
    else
        U(i, 15) = Inf;
        L(i, 15) = norminv(1 - par_15);
    end
end
% Set the length of MCMC run 'nMCMC'.
nMCMC = 20000;
% Set the number of particles we want to use for estimating
% the likelihood.
nP = 100;
% Pre-allocate the output matrix (nMCMC by J).
beta = zeros(nMCMC, J);
% set initial values for the J parameters.
betaInit = 1 * ones(1, J);
beta(1, :) = betaInit;
% Calculate likelihood estimation for the first iteration.
sigma = betaInit' * betaInit + eye(J);
sigma_diag = diag(sigma);
U_current = zeros(n, J);
L_current = zeros(n, J);
```

```
U_prop = zeros(n, J);
L_prop = zeros(n, J);
parfor i = 1 : J
    U_current(:, i) = U(:, i) * sqrt(sigma_diag(i));
    L_current(:, i) = L(:, i) * sqrt(sigma_diag(i));
end
log_likeli_current = zeros(n, 1);
parfor j = 1 : n
    log_likeli_current(j, 1) = log(qsilatmvnv(nP, sigma, ...
            L_current(j, :)', U_current(j, :)'));
end
sum_log_likeli_current = sum(log_likeli_current);
% Calculate the posterior for the first iteration.
log_prior_current = log(mvnpdf(betaInit, zeros(1, J), eye(J)));
posterior_current = sum_log_likeli_current + log_prior_current;
% Define the variable 'count', which is used to calculate the average
% acceptance probability.
count = 0;
% Start MCMC run from the second iteration.
for ii = 2 : nMCMC
    % Specify the proposal distribution.
    beta_prop = mvnrnd(beta(ii - 1, :), 0.3 * cov_sim, 1);
    % Compute the proposed log-likelihood estimate.
    sigma_prop = beta_prop' * beta_prop + eye(J);
    sigma_diag_prop = diag(sigma_prop);
    parfor i = 1 : J
        U_prop(:, i) = U(:, i) * sqrt(sigma_diag_prop(i));
        L_prop(:, i) = L(:, i) * sqrt(sigma_diag_prop(i));
    end
    log_likeli_prop = zeros(n, 1);
    parfor j = 1 : n
        log_likeli_prop(j, 1) = log(qsilatmvnv(nP, sigma_prop, ...
                L_prop(j, :)', U_prop(j, :)'));
    end
    sum_log_likeli_prop = sum(log_likeli_prop);
    % Calculate the proposed value of posterior.
    log_prior_prop = log(mvnpdf(beta_prop, zeros(1, J), eye(J)));
    posterior_prop = sum_log_likeli_prop + log_prior_prop;
```

```
    % Calculate the acceptance probability.
    acc = min(1, exp(posterior_prop - posterior_current));
    % Decide to accept or reject the proposed values.
    u = rand(1);
    if u < acc
        beta(ii, :) = beta_prop;
        posterior_current = posterior_prop;
        count = count + 1;
    else
        beta(ii, :) = beta(ii - 1, :);
    end

end

% Compute the average acceptance probability.
AccProb = count / (nMCMC - 1);
% Posterior Mean with first 5000 iterations are used as burn-in.
PostMean = zeros(1, J);
parfor h = 1 : J
    PostMean(1, h) = mean(beta(5001:nMCMC, h));
end

save('PMMH_Copula_Bernoulli_big_var.mat', 'AccProb', 'PostMean', 'beta');

display(AccProb);
display(PostMean);
```

D.3 Simulation-10 Mixed Margins

D.3.1 Copula_Data_Mixed.m

```
% Simulated example of Bayesian estimation of high dimensional Gaussian
% Copula model with 10 mixed margins by the PMMH algorithm, written by
% Xuebin Zheng (z3369961) for the Master Project (MATH5925).
% Supervisor: Associate Professor Josef Dick
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% << Data Generation Script (Mixed Margins) >>
```

```
% Gaussian Copula model with 10 mixed margins:
% Margin_1: Poisson(par_1), Margin_2: Poisson(par_2),
% Margin_3: Poisson(par_3), Margin_4: Poisson(par_4),
% Margin_5: Poisson(par_5), Margin_6: Poisson(par_6),
% Margin_7: Exponential(par_7), Margin_8: Exponential(par_8),
% Margin_9: Normal(par_9, par_10), Margin_10: Normal(par_11, par_12)
clear all;
format long;
% Input the number of observations 'n' we want to generate.
n = 1000;
% Input the 10 by k loading matrix B where 10 is the dimension of
% the Copula
% and k denotes the number of factors in the model.
% Here, B is a column vector if only one factor is used.
B = 0.5 * ones(10, 1);
J = size(B, 1);
% We use 10 'Mixed margins' with parameters specified below.
par_1 = 1.0; par_2 = 1.5; par_3 = 2.0; par_4 = 2.5; par_5 = 3.0;
par_6 = 3.5; par_7 = 1.5; par_8 = 0.5; par_9 = 0; par_10 = 1;
par_11 = 2; par_12 = 0.5;
margin_par = [par_1, par_2, par_3, par_4, par_5, par_6, par_7, par_8, ...
    par_9, par_10, par_11, par_12]';
% Now, we generate n observations by using Gaussian Copula.
mu = zeros(1, J);
% The loading matrix B is used to form the covariance matrix 'sigma'.
sigma = B * B' + eye(J);
Z = mvnrnd(mu, corrcov(sigma), n);
U = normcdf(Z, 0, 1);
% Pre-allocate the resulting data matrix 'X' which has dimension n by J.
% The resulting data matrix 'X' contains all of the simulated data with
% each row corresponds to one data point of J variables.
X = zeros(n, J);
for i = 1 : 6
    X(:, i) = poissinv(U(:, i), margin_par(i));
end
X(:, 7) = expinv(U(:, 7), 1 / par_7);
X(:, 8) = expinv(U(:, 8), 1 / par_8);
X(:, 9) = norminv(U(:, 9), par_9, sqrt(par_10));
```

```
X(:, 10) = norminv(U(:, 10), par_11, sqrt(par_12));
% Calculate the theoretical rank correlations of the data.
tauTheoretical = 2 .* asin(corrcov(sigma)) ./ pi;
% Calculate the sample rank correlations of the data.
tauSample = corr(X, 'type', 'Kendall');
% Make sure the simulated data preserve correct dependence structure.

save('Copula_Data_Mixed.mat', 'X', 'tauTheoretical', 'tauSample');
```

D.3.2 PMMH_Copula_Mixed.m

```
% Simulated example of Bayesian estimation of high dimensional Gaussian
% Copula model with 10 mixed margins by the PMMH algorithm, written by
% Xuebin Zheng (z3369961) for the Master Project (MATH5925).
% Supervisor: Associate Professor Josef Dick
% ------------------------------------------------------------------------
% ------------------------------------------------------------------------
% << PMMH Algorithm for Bayesian Copula Model (10 Mixed Margins) >>
tic
clear all;
format long;
load('Copula_Data_Mixed.mat');
load('cov_sim.mat');
% The first 6 margins are discrete and the rest (4) are continuous.
% Margin_1: Poisson(par_1), Margin_2: Poisson(par_2),
% Margin_3: Poisson(par_3), Margin_4: Poisson(par_4),
% Margin_5: Poisson(par_5), Margin_6: Poisson(par_6),
% Margin_7: Exponential(par_7), Margin_8: Exponential(par_8),
% Margin_9: Normal(par_9, par_10), Margin_10: Normal(par_11, par_12)
par_1 = 1.0; par_2 = 1.5; par_3 = 2.0; par_4 = 2.5; par_5 = 3.0;
par_6 = 3.5; par_7 = 1.5; par_8 = 0.5; par_9 = 0; par_10 = 1;
par_11 = 2; par_12 = 0.5;
margin_par = [par_1, par_2, par_3, par_4, par_5, par_6, par_7, par_8, ...
    par_9, par_10, par_11, par_12]';
X_D = X(:, 1:6);
X_C = X(:, 7:10);
[n, J_D] = size(X_D);
[J_C] = size(X_C, 2);
```

```
% Calculate the lower and upper bounds for the first 6 Poisson margins
% used in the integration.
U = zeros(n, J_D);
L = zeros(n, J_D);
U(:, 1:6) = X(:, 1:6);
L(:, 1:6) = X(:, 1:6) - 1;
for i = 1 : 6
    U(:, i) = poisscdf(U(:, i), margin_par(i));
    U(:, i) = norminv(U(:, i), 0, 1);
    L(:, i) = poisscdf(L(:, i), margin_par(i));
    L(:, i) = norminv(L(:, i), 0, 1);
end
% Set the length of MCMC run 'nMCMC'.
nMCMC = 20000;
% Set the number of particles we want to use for estimating
% the likelihood.
nP = 30;
% Pre-allocate the output matrix of size nMCMC by 10.
beta = zeros(nMCMC, 10);
% set initial values.
betaInit = 1 * ones(1, 10);
beta(1, :) = betaInit;
sigma = betaInit' * betaInit + eye(10);
sigma_diag = diag(sigma);
% Partition the Sigma matrix.
sigma_DD = sigma(1:6, 1:6);
sigma_CD = sigma(7:10, 1:6);
sigma_DC = sigma(1:6, 7:10);
sigma_CC = sigma(7:10, 7:10);
% Calculate the conditional covariance matrix.
cond_sigma_DC = sigma_DD - sigma_DC * ...
        (sigma_CC \ eye(size(sigma_CC, 1))) * sigma_CD;
% Calculate the lower and upper bounds for the first iteration.
U_current = zeros(n, J_D);
L_current = zeros(n, J_D);
U_prop = zeros(n, J_D);
L_prop = zeros(n, J_D);
for i = 1 : 6
```

```
    U_current(:, i) = U(:, i) * sqrt(sigma_diag(i));
    L_current(:, i) = L(:, i) * sqrt(sigma_diag(i));
end
% Z_C is n by 4 matrix, but we initially define Z_C as n by 10 matrix
% only for programming purpose.
Z_C = zeros(n, 10);
for i = 7 : 8
    Z_C(:, i) = expcdf(X(:, i), 1 / margin_par(i, 1));
    Z_C(:, i) = sqrt(sigma_diag(i)) * norminv(Z_C(:, i), 0, 1);
end
Z_C(:, 9) = normcdf(X(:, 9), par_9, sqrt(par_10));
Z_C(:, 9) = sqrt(sigma_diag(9)) * norminv(Z_C(:, 9), 0, 1);
Z_C(:, 10) = normcdf(X(:, 10), par_11, sqrt(par_12));
Z_C(:, 10) = sqrt(sigma_diag(10)) * norminv(Z_C(:, 10), 0, 1);
Z_C = Z_C(:, 7:10);
% Calculate the conditional mean 'cond_mu_DC' is a 6 by n matrix.
cond_mu_DC = zeros(6, n);
for i = 1 : n
    cond_mu_DC(:, i) = sigma_DC * (sigma_CC \ eye(size(sigma_CC, 1))) ...
        * Z_C(i, :)';
end
% Calculate the log-likelihood for the first iteration.
mu = zeros(1, 4);
log_likeli = zeros(n, 1);
log_likeli_C = zeros(n, 1);
parfor h = 1 : n
    log_likeli(h, 1) = log(qsilatmvnv(nP, cond_sigma_DC, ...
        L_current(h, :)' - cond_mu_DC(:, h), U_current(h, :)'- ...
        cond_mu_DC(:, h)));
end
for ii = 1 : n
    log_likeli_C(ii, 1) = log(mvnpdf(Z_C(ii, :), mu, sigma_CC)) + ...
        log(sqrt(sigma_diag(7)) * exppdf(X(ii, 7), 1 / par_7) / ...
        normpdf(norminv(expcdf(X(ii, 7), 1 / par_7), 0, 1), 0, 1)) + ...
        log(sqrt(sigma_diag(8)) * exppdf(X(ii, 8), 1 / par_8) / ...
        normpdf(norminv(expcdf(X(ii, 8), 1 / par_8), 0, 1), 0, 1)) + ...
        log(sqrt(sigma_diag(9)) * normpdf(X(ii, 9), par_9, ...
        sqrt(par_10)) / normpdf(norminv(normcdf(X(ii, 9), par_9, ...
```

```
        sqrt(par_10)), 0, 1), 0, 1)) + ...
        log(sqrt(sigma_diag(10)) * normpdf(X(ii, 10), par_11, ...
        sqrt(par_12)) / normpdf(norminv(normcdf(X(ii, 10), par_11, ...
        sqrt(par_12)), 0, 1), 0, 1));
end
sum_log_likeli_current = sum(log_likeli + log_likeli_C);
% Calculate the posterior for the first iteration.
log_prior_current = log(mvnpdf(betaInit, zeros(1, 10), eye(10)));
posterior_current = sum_log_likeli_current + log_prior_current;
% Define the variable 'count', which is used to calculate the average
% acceptance probability.
count = 0;
% Start MCMC run from the second iteration.
for N = 2 : nMCMC
    % Specify the proposal distribution.
    beta_prop = mvnrnd(beta(N - 1, :), 0.5 * cov_sim, 1);
    % Compute the proposed log-likelihood estimate.
    sigma = beta_prop' * beta_prop + eye(10);
    sigma_diag = diag(sigma);
    % Partition the Sigma matrix.
    sigma_DD = sigma(1:6, 1:6);
    sigma_CD = sigma(7:10, 1:6);
    sigma_DC = sigma(1:6, 7:10);
    sigma_CC = sigma(7:10, 7:10);
    % Calculate the conditional covariance matrix.
    cond_sigma_DC = sigma_DD - sigma_DC * ...
            (sigma_CC \ eye(size(sigma_CC, 1))) * sigma_CD;
    for i = 1 : 6
        U_prop(:, i) = U(:, i) * sqrt(sigma_diag(i));
        L_prop(:, i) = L(:, i) * sqrt(sigma_diag(i));
    end
    % Z_C is n by 4 matrix, but we initially define Z_C as n by 10 matrix
    % only for programming purpose.
    Z_C = zeros(n, 10);
    for i = 7 : 8
        Z_C(:, i) = expcdf(X(:, i), 1 / margin_par(i, 1));
        Z_C(:, i) = sqrt(sigma_diag(i)) * norminv(Z_C(:, i), 0, 1);
    end
```

```
Z_C(:, 9) = normcdf(X(:, 9), par_9, sqrt(par_10));
Z_C(:, 9) = sqrt(sigma_diag(9)) * norminv(Z_C(:, 9), 0, 1);
Z_C(:, 10) = normcdf(X(:, 10), par_11, sqrt(par_12));
Z_C(:, 10) = sqrt(sigma_diag(10)) * norminv(Z_C(:, 10), 0, 1);
Z_C = Z_C(:, 7:10);
% Calculate the conditional mean 'cond_mu_DC' is a 6 by n matrix.
cond_mu_DC = zeros(6, n);
for i = 1 : n
    cond_mu_DC(:, i) = sigma_DC * ...
            (sigma_CC \ eye(size(sigma_CC, 1))) * Z_C(i, :)';
end
% Calculate the log-likelihood estimates.
parfor h = 1 : n
    log_likeli(h, 1) = log(qsilatmvnv(nP, cond_sigma_DC, ...
        L_prop(h, :)' - cond_mu_DC(:, h), U_prop(h, :)' - ...
        cond_mu_DC(:, h)));
end
for ii = 1 : n
    log_likeli_C(ii, 1) = log(mvnpdf(Z_C(ii, :), mu, sigma_CC)) + ...
     log(sqrt(sigma_diag(7)) * exppdf(X(ii, 7), 1 / par_7) / ...
     normpdf(norminv(expcdf(X(ii, 7), 1 / par_7), 0, 1), 0, 1)) + ...
     log(sqrt(sigma_diag(8)) * exppdf(X(ii, 8), 1 / par_8) / ...
     normpdf(norminv(expcdf(X(ii, 8), 1 / par_8), 0, 1), 0, 1)) + ...
     log(sqrt(sigma_diag(9)) * normpdf(X(ii, 9), par_9, ...
     sqrt(par_10)) / normpdf(norminv(normcdf(X(ii, 9), par_9, ...
     sqrt(par_10)), 0, 1), 0, 1)) + ...
     log(sqrt(sigma_diag(10)) * normpdf(X(ii, 10), par_11, ...
     sqrt(par_12)) / normpdf(norminv(normcdf(X(ii, 10), par_11, ...
     sqrt(par_12)), 0, 1), 0, 1));
end
sum_log_likeli_prop = sum(log_likeli + log_likeli_C);
% Calculate the posterior.
log_prior_prop = log(mvnpdf(beta_prop, zeros(1, 10), eye(10)));
posterior_prop = sum_log_likeli_prop + log_prior_prop;
% Calculate the acceptance probability.
acc = min(1, exp(posterior_prop - posterior_current));
% Decide to accept or reject the proposed values.
u = rand(1);
```

```
    if u < acc
        beta(N, :) = beta_prop;
        posterior_current = posterior_prop;
        count = count + 1;
    else
        beta(N, :) = beta(N - 1, :);
    end

end

% Compute the average acceptance probability.
AccProb = count / (nMCMC - 1);
% Posterior Mean with first 5000 iterations are used as burn-in.
PostMean = zeros(1, 10);
parfor s = 1 : 10
    PostMean(1, s) = mean(beta(5001:nMCMC, s));
end
toc
save('PMMH_Copula_Mixed_cov_sim.mat', 'AccProb', 'PostMean', 'beta');

display(AccProb);
display(PostMean);
```

D.4 qsilatmvnv.m

```
function [ p, e ] = qsilatmvnv( m, r, a, b )
%
%  [ P E ] = QSILATMVNV( M, R, A, B )
%    uses a randomized lattice rule with m points to estimate an
%    MVN probability for positive definite covariance matrix r,
%     with lower integration limit column vector a and upper
%     integration limit column vector b.
%   Probability p is output with error estimate e.
%    Example use:
%     r = [4 3 2 1;3 5 -1 1;2 -1 4 2;1 1 2 5];
%     a = -inf*[1 1 1 1 ]'; b = [ 1 2 3 4 ]';
%     [ p e ] = qsilatmvnv( 5000, r, a, b ); disp([ p e ])
```

```
%
%   This function uses a modification of an algorithm in the paper
%      "Numerical Computation of Multivariate Normal Probabilities",
%      A. Genz, J. of Comp. Graph. Stat., 1(1992), pp. 141-149.
%
%  The primary references for the numerical integration are
%  "Randomization of Number Theoretic Methods for Multiple Integration",
%     R. Cranley & T.N.L. Patterson, SIAM J Numer Anal, 13(1976),
%   pp. 904-14. and
%    "Fast Component-by-Component Construction, a Reprise for Different
%     Kernels", D. Nuyens & R. Cools. In H. Niederreiter and D. Talay,
%     editors, Monte-Carlo and Quasi-Monte Carlo Methods 2004,
%     Springer-Verlag, 2006, 371-385.
%
%Alan Genz is the author of this function and following Matlab functions.
%          Alan Genz, WSU Math, PO Box 643113, Pullman, WA 99164-3113
%          Email : alangenz@wsu.edu
%
%
% Copyright (C) 2013, Alan Genz,  All rights reserved.
%
% Redistribution and use in source and binary forms, with or without
% modification, are permitted provided the following conditions are met:
%   1. Redistributions of source code must retain the above copyright
%      notice, this list of conditions and the following disclaimer.
%   2. Redistributions in binary form must reproduce the above copyright
%      notice, this list of conditions and the following disclaimer in
%      the documentation and/or other materials provided with the
%      distribution.
%   3. The contributor name(s) may not be used to endorse or promote
%      products derived from this software without specific prior
%      written permission.
% THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
% "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
% LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
% FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
% COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
% INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
```

```
% BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
% OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
% ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR
% TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF USE
% OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
%
% Initialization
%
[ ch as bs ] = chlrdl( r, a, b ); ct = ch(1,1); ai = as(1); bi = bs(1);
if ai > -9*ct, if ai < 9*ct, c = Phi(ai/ct); else, c=1; end,else c=0;end
if bi > -9*ct, if bi < 9*ct, d = Phi(bi/ct); else, d=1; end,else d=0;end
[n, n] = size(r); ns = 10; [ q pr ] = fstrnk( m/ns, n-1 ); q = q/pr;
y = zeros(n-1,pr); on = ones(1,pr); ci = c; dci = d - ci; p = 0; e = 0;
%
% Randomization loop for ns samples
%
for S = 1 : ns, c = ci; dc = dci; vp = dc;
  %
  % Compute randomized MVN integrand with tent periodization
  %
  for i = 2 : n, xi = abs( 2*mod( q(i-1)*[1:pr] + rand, 1 ) - 1 );
    y(i-1,:) = Phinv( c + xi.*dc ); s = ch(i,1:i-1)*y(1:i-1,:);
    ct = ch(i,i); ai = as(i) - s; bi = bs(i) - s; c = on; d = c;
    c(find( ai < -9*ct )) = 0; d(find( bi < -9*ct )) = 0;
    tl = find( abs(ai) < 9*ct ); c(tl) = Phi(ai(tl)/ct);
    tl = find( abs(bi) < 9*ct ); d(tl) = Phi(bi(tl)/ct);
    dc = d - c; vp = vp.*dc;
  end, d = ( mean(vp) - p )/S; p = p + d; e = ( S - 2 )*e/S + d^2;
end, e = 3*sqrt(e);
% error estimate is 3 x standard error with ns samples.
%
% end qsilatmvnv
%
%
%  Standard statistical normal distribution functions
%
function p =   Phi(z), p =  erfc( -z/sqrt(2) )/2;
function z = Phinv(p), z = norminv( p );
```

```
%
function [ c, ap, bp ] = chlrdl( r, a, b )
%
% Computes scaled permuted lower Cholesky factor c for r which may be
% singular, also scaling and permuting integration limit vectors a and b.
%
ep = 1e-10; % singularity tolerance;
%
[n,n] = size(r); c = r; ap = a; bp = b; dc = sqrt(max(diag(c),0));
for i = 1 : n, d = dc(i);
  if d > 0, ap(i) = ap(i)/d; bp(i) = bp(i)/d;
    c(:,i) = c(:,i)/d; c(i,:) = c(i,:)/d;
  end
end, y = zeros(n,1); sqtp = sqrt(2*pi);
for k = 1 : n, epk = ep*k; im = k; ck = 0; vm = 1 + epk;
  for i = k : n
    if c(i,i) > ep, ci = sqrt(c(i,i)); s = c(i,1:k-1)*y(1:k-1);
      ai = ( ap(i) - s )/ci; bi = ( bp(i) - s )/ci;
      dna = 0; dsa = 0; dnb = 0; dsb = 1;
      if ai > -9, dna = exp(-ai^2/2)/sqtp; dsa = Phi(ai); end
      if bi <  9, dnb = exp(-bi^2/2)/sqtp; dsb = Phi(bi); end,p=dsb-dsa;
      if p > epk, mn = dna - dnb; v = 0;
        if     ai >  -9 & bi <  9; v =  ai*dna - bi*dnb;
        elseif ai <= -9 & bi <  9, v =         - bi*dnb;
        elseif ai >  -9 & bi >= 9, v =  ai*dna;
        end, mn = mn/p; v = 1 + v/p - mn^2;
      else, mn = ( ai + bi )/2;
        if ai < -9, mn = bi; elseif bi > 9, mn = ai; end
      end, if v < vm, im = i; vm = v; y(k) = mn; ck = ci; end
    end
  end
  if im > k, c(im,im) = c(k,k);
    t = c(im,1:k-1); c(im,1:k-1) = c(k,1:k-1); c(k,1:k-1) = t;
    t = c(im+1:n,im); c(im+1:n,im) = c(im+1:n,k); c(im+1:n,k) = t;
    t = c(k+1:im-1,k); c(k+1:im-1,k) = c(im,k+1:im-1)';c(im,k+1:im-1)=t';
    tv = ap(im); ap(im) = ap(k); ap(k) = tv;
    tv = bp(im); bp(im) = bp(k); bp(k) = tv;
  end, c(k,k+1:n) = 0;
```

```
  if ck < epk, c(k:n,k) = 0; y(k) = 0;
  else, c(k,k) = ck; c(k+1:n,k) = c(k+1:n,k)/ck;
    for i = k+1 : n, c(i,k+1:i) = c(i,k+1:i) - c(i,k)*c(k+1:i,k)'; end
  end
end
%
% end chlrdr
%
%
function [ z, n ] = fstrnk( ni, sm, om, gm, bt )
%
% Reference:
%   "Fast Component-by-Component Construction, a Reprise for Different
%     Kernels", D. Nuyens and R. Cools. In H. Niederreiter and D. Talay,
%     editors, Monte-Carlo and Quasi-Monte Carlo Methods 2004,
%     Springer-Verlag, 2006, 371-385.
% Modifications to original by A. Genz, 05/07
% Typical Use:
%  om = @(x)x.^2-x+1/6; n = 99991; s = 100; gam = 0.9.^[1:s];
%  z = fastrank( n, s, om, gam, 1 + gam/3 ); disp([z'; e])
%
n = fix(ni); if ~isprime(n), pt = primes(n); n = pt(length(pt)); end
if nargin < 3, om = @(x)x.^2-x+1/6;
  bt = ones(1,sm); gm = [ 1 (4/5).^[0:sm-2] ];
end, q = 1; w = 1; z = [1:sm]'; m = ( n - 1 )/2; g = prmrot(n);
perm = [1:m]; for j = 1 : m-1, perm(j+1) = mod( g*perm(j), n ); end
perm = min( n - perm, perm ); c = om(perm/n); fc = fft(c);
for s = 2 : sm, q = q.*( bt(s-1) + gm(s-1)*c([w:-1:1 m:-1:w+1]) );
  [ es w ] = min( real( ifft( fc.*fft(q) ) ) ); z(s) = perm(w);
end
%
% end fstrnk
%
function r = prmrot(pin)
%
% find primitive root for prime p, might fail for large primes (p > 32e7)
%
p = pin; if ~isprime(p), pt = primes(p); p = pt(length(pt)); end
```

```
pm = p - 1; fp = unique(factor(pm)); n = length(fp); r = 2; k = 1;
while k <= n; d = pm/fp(k); rd = r;
  for i = 2 : d, rd = mod( rd*r, p ); end % computes r^d mod p
  if rd == 1, r = r + 1; k = 0; end, k = k + 1;
end
%
% prmrot
%
```