PARTICLE GIBBS METHODS IN STOCHASTIC VOLATILITY MODELS

by

Chen Gong

B.S. in Statistics, Shandong University, China, 2012

M.S. in Statistics, The George Washington University, 2014

Submitted to the Graduate Faculty of

the Kenneth P. Dietrich School of Arts and Sciences in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2019

UNIVERSITY OF PITTSBURGH

KENNETH P. DIETRICH SCHOOL OF ARTS AND SCIENCES

This dissertation was presented

by

Chen Gong

It was defended on

Aug 2nd, 2019

and approved by

David S. Stoffer, Ph.D., Department of Statistics

Yu Cheng, Ph.D., Department of Statistics

Kehui Chen, Ph.D., Department of Statistics

Robert T. Krafty, Ph.D., Department of Biostatistics

Dissertation Director: David S. Stoffer, Ph.D., Department of Statistics

PARTICLE GIBBS METHODS IN STOCHASTIC VOLATILITY MODELS

Chen Gong, PhD

University of Pittsburgh, 2019

The Stochastic Volatility (SV) model and the Multivariate Stochastic Volatility (MSV) model are powerful tools for modeling the volatility of stock price data. A lot of research has been done in the past 40 years. One popular method is to represent the SV as a hidden Markov model and employ the Bayesian inference through the Markov Chain Monte Carlo (MCMC) method. Sampling the hidden states in the SV model from the full conditional distribution is the key part in MCMC. Several particle methods had been developed to approximate the hidden states, such as the Sequential Monte Carlo (SMC) and Particle MCMC. However, these methods suffer from the path degeneracy problem. The Particle Gibbs with Ancestor Sampling (PGAS) was introduced to deal with the path degeneracy in 2014 and it made the MCMC algorithm efficient. However, we believe that the efficiency of MCMC still can be improved by choosing a suitable prior distribution for the parameters in the SV model. In this thesis, we explore the potential reason, which leads to the low efficiency problem, and propose a new method to deal with the SV model through the PGAS algorithm. In our proposed method, we employ the bivariate normal distribution as prior distribution for the parameters in the state equation. The negative correlation between the two parameters can be explained by setting a negative correlation coefficient in the prior distribution. Consequently, the efficiency of the algorithm is improved significantly after sampling parameters out jointly through the Random Walk Metropolis Hastings (RWMH). However, sometimes it is difficult to find a good proposal distribution for RWMH in practice. Thus, we apply the adaptive MCMC to our proposed method to further improve the algorithm. Moreover, we extend our proposed method to the MSV model and get a high efficiency algorithm. We provide theoretical details of particle method to justify the validity of the proposed methods. Numerical experiments,

including simulation studies and applications to the S&P 500 index data and some banks' stock data, are presented to demonstrate the good performances of the proposed methods.

Keywords: Stochastic Volatility, Hidden Markov model, Particle Gibbs with Ancestor Sampling, Efficient Markov Chain Monte Carlo, Multivariate Stochastic Volatility model.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

1.0 INTRODUCTION

In the financial time series, the Stochastic Volatility (SV) model is widely used to model the stock price data (Shumway and Stoffer, 2017). In order to get the estimation for the parameters in the SV model, a lot of research has been done in the past 40 years. One popular method is to use the Bayesian inference. Treating the SV model as a hidden Markov model, researcher can perform the Bayesian analysis by using the Markov Chain Monte Carlo (MCMC) method. These methods can be found in Jacquier et al. (1994) and Taylor (1994). However, the slow convergence problem was reported in Chib and Greenberg (1996). The reason is that sampled hidden states from the full conditional distribution are highly correlated when we use the MCMC procedure. Kim et al. (1998) developed a more efficient MCMC algorithm by applying a mixture simulator and integrating out the log-volatilities.

In Del Moral (1996), the Particle Filter was introduced to sample the hidden states together from the conditional distribution. However, the particle filters suffer from the path degeneracy problem, which is mentioned in Doucet et al. (2000). The path degeneracy makes the approximation of joint density distribution unreliable for long period time series. In order to avoid the path degeneracy, which leads all but one of the importance weights decreases to zero, several resampling methods were introduced in late 1990s, which can be found in Liu and Chen (1998) and Doucet et al. (2000). In addition, the Forward Filter Backward Simulator (FFBSi) and the Forward Filter Backward Smoother (FFBSm) were introduced in Godsill et al. (2004) and Doucet et al. (2000) to handle path degeneracy.

In Andrieu et al. (2010), the Particle Markov Chain Monte Carlo (PMCMC) method was introduced, which combines the MCMC and the Sequential Monte Carlo (SMC). The PMCMC became an important and powerful tool for solving the State Space Model (SSM) in the Bayesian analysis framework. The Particle Gibbs (PG) sampler was also introduced as a PMCMC method in

Andrieu et al. (2010). However, the PG sampler still suffers from the path degeneracy and it can work well only for short time series. One way to solve this problem is involving the Backward Simulation sweep into the PG sampler, which is called Particle Gibbs with Backward Simulation (PGBS) Whiteley et al. (2010). On the other hand, one can use the Particle Gibbs with Ancestor Sampling (PGAS) method to deal with the path degeneracy (Lindsten et al., 2014).

The PGAS can be used to perform a Bayesian analysis on the SV models through the MCMC and it can improve the mixing of the Markov kernel by Ancestor Sampling (AS). However, we notice that there are still some slow convergence problems in the original sampling procedure. One potential reason is the high correlation among model parameters. In order to improve the efficiency of the algorithm, we propose a new method for the SV models by employing a bivariate normal distribution as prior. A Random Walk Metropolis Hastings (RWMH) algorithm is used to implement the new method. The new method reduces the parameter inefficiencies significantly. Then, we introduce the Adaptive Markov Chain Monte Carlo (Adaptive MCMC) to the proposed method to overcome the difficulty of choosing good proposal distribution in RWMH. Moreover, we extend our new method to the Multivariate Stochastic Volatility (MSV) model.

The thesis is organized as follows. Chapter 2 first reviews the State Space model with the linear setting and non-linear non-Gaussian setting. Then, we move on to some particle methods, including the SMC method, PG sampler and PGAS sampler. Chapter 3 reviews the SV models and MSV model and introduces some approaches to solve the SV models. A case study is presented at the end of Chapter 3, which illustrates the motivation of our research. In Chapter 4, we propose a new method to solve the slow convergence problem in the original method, which is introduced in Chapter 3. Then, the Adaptive MCMC method is used to further improve efficiency of the algorithm. At the end of Chapter 4, we extend the new method to the MSV model. Finally, we present the simulation results and applications of our proposed methods in Chapter 5.

2.0 PARTICLE MARKOV CHAIN MONTE CARLO

In this chapter, we first introduce the State Space Model (SSM) and Nonlinear State Space Model (NLSS), which are commonly used to model time series and dynamic systems. Then, we move to the Sequential Monte Carlo (SMC), which can sequentially sample the sequence from the posterior densities. In the last section, we review the Particle Gibbs with Ancestor Sampling (PGAS) and apply it to the nonlinear state space model.

2.1 STATE SPACE MODEL

2.1.1 Linear Gaussian State Space Model

The State Space Model (SSM), which is also known as a Hidden Markov Model (HMM), was presented in Kalman (1960) and Kalman and Bucy (1961). The State Space Model is a powerful tool in time series analysis and dynamic systems modeling.

Firstly, we consider the linear Gaussian State Space Models or Dynamic Linear Model (DLM)

$$X_t = \Phi X_{t-1} + W_t \tag{2.1}$$

$$Y_t = A_t X_t + V_t \tag{2.2}$$

where $W_t \sim iidN_p(0, Q)$, $V_t \sim iidN_q(0, R)$, $X_0 \sim N_p(\mu_0, \Sigma_0)$ and $t = 1, \cdots, T$. The equation (2.1), also known as the state equation, shows that the $p \times 1$ state vector X_t can be generated by the past $p \times 1$ state vector X_{t-1}. The equation (2.2) is called the observation equation and the observed Y_t has a linear relationship with X_t and noise V_t as (2.2). Although the states sequence X_t cannot be observed directly, we can observe Y_t from the real world.

From a density point of view, the hidden Markov state process $\{X_t, t \in \mathbb{N}_0\}$ is characterized by the initial density $X_0 \sim N_p(\mu_0, \Sigma_0)$ and the transition probability density

$$X_t \mid X_{t-1} \sim N_p(\Phi X_{t-1}, Q). \tag{2.3}$$

Then, the observations are conditionally independent given the states

$$Y_t \mid X_t \sim N_q(A_t X_t, R). \tag{2.4}$$

We are interested in getting the estimation for the underlying unobserved states X_t, given the observation $Y_{1:s} = \{Y_1, \cdots, Y_s\}$. When $s = t$, we call it filtering. When $s < t$, we call it forecasting or prediction. When $s > t$, we call it smoothing. The Kalman Filter and Smoother can give the solution to these problems. In addition, the Newton-Raphson and EM (expectation-maximization) algorithm (Shumway and Stoffer, 1982) can find the Maximum Likelihood Estimator (MLE) for parameters. The details about the Linear Gaussian State Space Mode can be found at Shumway and Stoffer (2017).

2.1.2 Non-Linear Non-Gaussian State Space Model

In practice, the linear and Gaussian assumptions do not always hold for State Space Model. In this case, we call it Non-Linear Non-Gaussian State Space Model. The model can be expressed as

$$x_t \sim f_\theta(x_t \mid x_{t-1}) \tag{2.5}$$

$$y_t \sim g_\theta(y_t \mid x_t), \tag{2.6}$$

where $t = 1, \cdots, T$, $x_0 \sim \mu_\theta(x_0)$ and $\theta \in \Theta$ is a static parameter or a vector of static parameters. Typically, researcher's interest to this model is to infer the latent states given the observations from the system. Then, we can perform the Bayesian inference on the model. Similar to the Linear Gaussian State Space Model, for $p_\theta(x_t \mid y_{1:s})$, when $s = t$, we call it filter. When $s < t$, we call it prediction. And when $s > t, (t \leq T)$, we call it smoother.

In particular, for a known parameter θ, what we are interested in is performing Bayesian inference on the joint posterior density $p_\theta(x_{1:T} \mid y_{1:T})$. The complete data likelihood can be written as,

$$p_\theta(x_{1:T}, y_{1:T}) = \mu_\theta(x_0) \prod_{t=1}^{T} g_\theta(y_t \mid x_t) \prod_{t=1}^{T} f_\theta(x_t \mid x_{t-1}). \tag{2.7}$$

By the Bayes' rule, we have

$$p_\theta(x_{1:T} \mid y_{1:T}) \propto p_\theta(x_{1:T}, y_{1:T}). \tag{2.8}$$

If the θ is unknown, we can assign a prior distribution $\pi(\theta)$. Then, the Bayesian inference will depend on the joint density

$$p_\theta(\theta, x_{1:T} \mid y_{1:T}) \propto p_\theta(x_{1:T}, y_{1:T})\pi(\theta). \tag{2.9}$$

Then, we can easily apply the Gibbs sampler to get the posterior distribution of θ. Specifically, first initialize $\theta^{(0)} \in \Theta$ and then iterate the following steps:

(i) Draw $x_{1:T}^{[r]} \sim p(x_{1:T} \mid \theta^{[r-1]}, y_{1:T})$.

(ii) Draw $\theta^{[r]} \sim p(\theta \mid x_{1:T}^{[r]}, y_{1:T})$.

A Markov chain $\{\theta^{[r]}, x_{1:T}^{[r]}\}_{r \geq 1}$ will be generated by the Gibbs sampler and the stationary distribution is $p_\theta(\theta, x_{1:T} \mid y_{1:T})$. Then, the subchain $\{\theta^{[r]}\}_{r \geq 1}$ will have the stationary distribution $p(\theta \mid y_{1:T})$. In Step (i), we need to compute the joint smoothing density $p_\theta(x_{1:T}, y_{1:T})$. Thus, we need to address this intermediate problem before applying the Gibbs sampler.

2.2 SEQUENTIAL MONTE CARLO

For the Non-Linear Non-Gaussian State Space Model, we can not find the close form expression for $p_\theta(x_{1:T} \mid y_{1:T})$ and $p_\theta(\theta, x_{1:T} \mid y_{1:T})$ easily in most cases, which makes it hard to inference in practice. In order to overcome this difficulty, people usually use some approximation methods. Especially, Monte Carlo method plays a very important role, which provides a systematic framework to calculate the Bayesian posterior probabilities. In this section, we will introduce the Sequential Monte Carlo method, which is also known as the Particle Filter (PF).

The particle filter is based on the sequential important sampling and resampling techniques. The samples $\{x^i_{1:t}\}^N_{i=1}$ are called particles. In the importance sampler, we can generate N independent particles $\{x^i_{1:t}\}^N_{i=1}$ from the proposal density $q_t(x_{1:t} \mid y_{1:t})$. Then, we can assign the important weights to these particles. The weights are given by

$$w^i_t = \frac{p(x^i_{1:t} \mid y_{1:t})}{q_t(x^i_{1:t} \mid y_{1:t})}, \tag{2.10}$$

where $i = 1, \cdots, N$. Then, we can normalize the weights to let them sum to one. That is,

$$W^i_t = \frac{w^i_t}{\sum_{j=1}^N w^j_t}. \tag{2.11}$$

Employing the sequential method, we can decompose the proposal density as

$$\begin{aligned} q_t(x^i_{1:t} \mid y_{1:t}) &= q_t(x^i_t \mid x^i_{1:t-1}, y_{1:t}) q_{t-1}(x^i_{1:t-1} \mid y_{1:t}) \\ &= q_t(x^i_t \mid x^i_{t-1}, y_t) q_{t-1}(x^i_{1:t-1} \mid y_{1:t-1}). \end{aligned} \tag{2.12}$$

This implies that the important weights are given by

$$\begin{aligned} w^i_t = \frac{p(x^i_{1:t} \mid y_{1:t})}{q_t(x^i_{1:t} \mid y_{1:t})} &\propto \frac{p(x^i_t \mid x^i_{t-1}, y_t)}{q_t(x^i_t \mid x^i_{t-1}, y_t)} \frac{p(x^i_{1:t-1} \mid y_{1:t-1})}{q_{t-1}(x^i_{1:t-1} \mid y_{1:t-1})} \\ &\propto \frac{p(x^i_t, y_t \mid x^i_{t-1})}{q_t(x^i_t \mid x^i_{t-1}, y_t)} \frac{p(x^i_{1:t-1} \mid y_{1:t-1})}{q_{t-1}(x^i_{1:t-1} \mid y_{1:t-1})} \\ &\propto \frac{g(y_t \mid x^i_t) f(x^i_t \mid x^i_{t-1})}{q_t(x^i_t \mid x^i_{t-1}, y_t)} \frac{p(x^i_{1:t-1} \mid y_{1:t-1})}{q_{t-1}(x^i_{1:t-1} \mid y_{1:t-1})}. \end{aligned} \tag{2.13}$$

Then, we define the weight function as

$$\alpha^i_t(x^i_{1:t}) = \frac{g(y_t \mid x^i_t) f(x^i_t \mid x^i_{t-1})}{q_t(x^i_t \mid x^i_{t-1}, y_t)}. \tag{2.14}$$

The important weight can be expressed as

$$w_t^i = \alpha_t^i w_{t-1}^i, \tag{2.15}$$

which is a sequential updating formula for the important weight. The $\{x_{1:t}^i, W_t^i\}_{i=1}^N$ are called a weighted particle system, where $W_t^i = \frac{\alpha_t^i}{\sum_{j=1}^N \alpha_t^j}$. However, this procedure has a serious drawback. The weights updating in (2.15) are unstable. The variance of the importance weights increases as the time index increases with a typical exponential rate (Cappé and Ryden, 2005). This will lead to the effect that all but one of weights decreases to zero. Eventually, only one of the particles will be emphasized by the procedure. This will result in a bad performance of the sampler.

In order to overcome this drawback, resampling techniques are the key ingredients of Sequential Monte Carlo methods. Denote $\pi(x_{1:t}) = p(x_{1:t} \mid y_{1:t})$. Then, its Importance Sampling (IS) approximation $\hat{\pi}(dx_{1:t}) := \sum_{i=1}^N W_t^i \delta_{x_{1:t}^i}(dx_{1:t})$, which is an empirical point mass distribution on $\mathcal{X}^t$, is based on the weighted samples from $q_t(x_{1:t} \mid y_{1:t})$. Intuitively, we can just sample from the IS approximation $\hat{\pi}(x_{1:t})$ to get the approximate samples from $\pi(x_{1:t})$. If we are going to get N samples from $\pi(x_{1:t})$, we just need to resample it N times independently from the approximated distribution, $\hat{\pi}(x_{1:t})$, and get the particles $\tilde{x}_{1:t}^i$. Although the resampling procedure will introduce some extra noise, it can remove the low-weight particles with high probability, which is an important advantage of the resampling technique.

Through resampling technique, the equally weighted particles system will become $\{\tilde{x}_{1:t}^i, \frac{1}{N}\}$, which can be used in the Particle Filter algorithm. Then, we can introduce the Particle Filter in Algorithm 1. Through the Particle Filter, we can get the approximation of $p_\theta(x_{1:T} \mid y_{1:T})$. This Particle Filter is a foundation of the Particle Gibbs sampler, which will be discussed in Section 2.3.1.

Algorithm 1 Particle Filter: Each Step is for $i = 1, \cdots, N$

1: Draw $x_1^i \sim q_1(x_1 \mid y_1)$.

2: Calculate the weight function $\alpha_1^i(x_1^i)$ and get the normalized weight $W_1^i = \frac{\alpha_1^i}{\sum_{j=1}^N \alpha_1^j}$.

3: **For** $t \geq 2$ **do**

4: Resample $\tilde{x}_{t-1}^i$ with $P(\tilde{x}_{t-1}^i = x_{t-1}^j) = W_{t-1}^j$.

5: Draw $x_t^i \sim q_t(x_t \mid \tilde{x}_{t-1}^i, y_t)$ and set $x_{1:t}^i = \{\tilde{x}_{1:t-1}^i, x_t^i\}$.

6: Calculate $\alpha_t^i(x_t^i)$.

7: Compute normalized weight $W_t^i = \frac{\alpha_t^i}{\sum_{j=1}^N \alpha_t^j}$.

8: **End for**

2.3 PARTICLE GIBBS WITH ANCESTOR SAMPLING

In this section, we will review the Particle Gibbs (PG) sampler and the Particle Gibbs with Ancestor Sampling (PGAS) method. At first, we will introduce the Particle Gibbs, which was proposed by Andrieu et al. (Andrieu et al., 2010). The PG sampler is a special case of PGAS method. Therefore, PG is a good start point for introducing PGAS.

2.3.1 Particle Gibbs

Before we review the Particle Gibbs sampler, we should introduce the ancestor indices to the particle system. Assume that the weighted particle system $\{x_{t-1}^i, W_{t-1}^i\}_{i=1}^N$ is targeting to the density $\pi_\theta(x_{1:t-1}) = p_\theta(x_{1:t-1} \mid y_{1:t-1})$. Then, the empirical point mass approximation of the target distribution is given by

$$\hat{\pi}_\theta(dx_{1:t-1}) = \sum_{i=1}^{N} W_{t-1}^i \delta_{x_{1:t-1}^i}(dx_{1:t-1}). \tag{2.16}$$

Then, the propagation of particle system to time t can be done by sequential important sampling and resampling from the proposed kernel

$$M_{t,\theta}(x_t, a_t) = W_{t-1}^{a_t} \pi(x_t \mid x_{1:t-1}^{a_t}), \tag{2.17}$$

on product space $\{1, \cdots, N\} \times \mathcal{X}$, where $t = 2, \cdots, T$. Here, we define a_t^i as the index of the ancestor at time $t-1$ of particle x_t^i. The weighted particle system will become $\{x_t^i, W_t^i, a_t^i\}_{i=1}^N$. Then, the ancestral path of particle x_t^i will be $x_{1:t}^i$, which is defined recursively as

$$x_{1:t}^i := (x_{1:t-1}^{a_t^i}, x_t^i). \tag{2.18}$$

For the whole ancestral path $x_{1:T}^k$ of particle x_T^k, we denote it as

$$x_{1:T}^k = x_{1:T}^{b_{1:T}} := (x_1^{b_1}, \cdots, x_T^{b_T}), \tag{2.19}$$

where $b_T = k$ and $b_t = a_{t+1}^{b_{t+1}}$. The ancestor indices method is a very important concept to the Particle Gibbs sampler, and it allows us to track the resampled particle trajectory. In Figure 2.1, we give an example to ancestor indices.

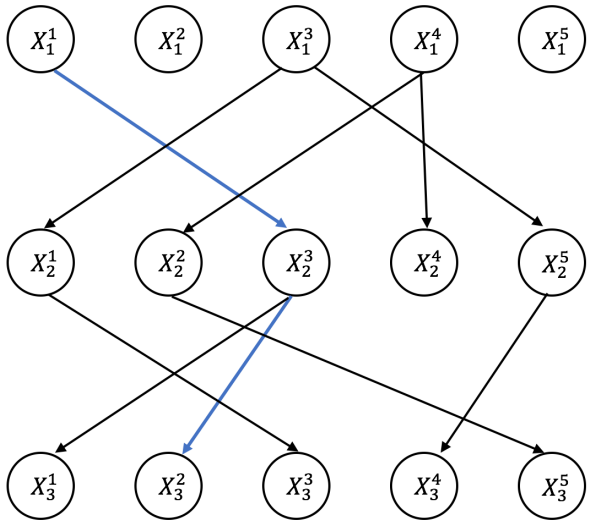

Figure 2.1: *Example of ancestor indices for particle $N = 5$ and time $T = 3$. The ancestral path of x_3^2 is shown as the blue path. Its ancestor indices are $a_2^3 = 1$ and $a_3^2 = 3$. Thus, the path is $x_{1:3}^2 = (x_1^{a_2^{a_3^2}}, x_2^{a_3^2}, x_3^2) = (x_1^1, x_2^3, x_3^2)$.*

Now from the weighted particle system $\{x_{t-1}^i, W_{t-1}^i, a_{t-1}^i\}_{i=1}^N$, we can generate the particles up to time t by independently sampling N times from the kernel in (2.17). That is, sampling from

$$\{x_t^i, a_t^i\} \sim M_{t,\theta}(x_t, a_t), \tag{2.20}$$

for $i = 1, \ldots, N$. Then, the new N particles will connect to the ancestral trajectory and we get the $x_{1:t}^i$ as in (2.18). The importance weights $\{w_t^i\}_{i=1}^N$ are given by $\alpha_t^i(x_{1:t}^i)$ in (2.14) and we normalized them by $W_t^i = w_t^i / \sum_{j=1}^N w_t^j$. This gives us a new weighted particle system $\{x_t^i, W_t^i, a_t^i\}_{i=1}^N$ targeting the density $\pi_\theta(x_{1:t})$. At this point, we accommodated the ancestor indices to the particle filter and are ready to introduce the Particle Gibbs algorithm.

Extending the procedure in Section 2.1.2 to Particle Gibbs sampler, we have the following framework:

(i) Draw $\theta^{[r]} \sim p(\theta \mid x_{1:T}^{[r-1]}, y_{1:T})$.

(ii) Draw $x_{1:T}^{[r]} \sim p(x_{1:T} \mid \theta^{[r]}, y_{1:T})$.

(iii) Draw k with $P(k = i) = W_T^i$ and then set $x'_{1:T} = x_{1:T}^k$.

Note that the framework above gives an approximation of joint density $p_\theta(\theta, x_{1:T} \mid y_{1:T})$ in (2.9). Denote the particles and ancestor indices generated by the Particle Filter at time t as

$$\mathbf{x}_t = \{x_t^1, \cdots, x_t^N\}, \quad \mathbf{a}_t = \{a_t^1, \cdots, a_t^N\}. \tag{2.21}$$

The density of particles and ancestor indices generated by Particle Filter is given by

$$\psi_\theta(\mathbf{x}_{1:T}, \mathbf{a}_{2:T}) = \prod_{i=1}^{N} \pi_\theta(x_1^i) \prod_{t=2}^{T} \prod_{i=1}^{N} M_{t,\theta}(x_t^i, a_t^i), \tag{2.22}$$

which is defined on the extended space $\mathcal{X}^{NT} \times \{1, \cdots, N\}^{N(T-1)}$.

Then, the samples $\{\theta^{[r]}, \mathbf{x}_{1:T}, \mathbf{a}_{2:T}, k\}$ generated by the framework above can be seen as coming from the space $\Omega := \Theta \times \mathcal{X}^{NT} \times \{1, \cdots, N\}^{N(T-1)+1}$. We can define the extended target density on Ω as

$$\begin{aligned} \phi(\theta, \mathbf{x}_{1:T}, \mathbf{a}_{2:T}, k) &= \phi(\theta, x_{1:T}^{b_{1:T}}, b_{1:T}) \phi(\mathbf{x}_{1:T}^{-b_{1:T}}, \mathbf{a}_{2:T}^{-b_{2:T}} \mid \theta, x_{1:T}^{b_{1:T}}, b_{1:T}) \\ &:= \frac{\pi(\theta, x_{1:T}^{b_{1:T}})}{N^T} \prod_{i=1, i \neq b_1}^{N} \pi_\theta(x_1^i) \prod_{t=2}^{T} \prod_{i=1, i \neq b_t}^{N} M_{t,\theta}(x_t^i, a_t^i), \end{aligned} \tag{2.23}$$

where $\mathbf{x}_t^{-i} = \{x_t^1, \cdots, x_t^{i-1}, x_t^{i+1}, \cdots, x_t^N\}$ and $\mathbf{x}_{1:T}^{-b_{1:T}} = \{\mathbf{x}_1^{-b_1}, \cdots, \mathbf{x}_T^{-b_T}\}$. Note that the ancestral path x_T^k is the particle trajectory $x_{1:T}^{b_{1:T}}$ in the density above.

For the framework above, we can perform Step (i) by constructing the following density in Equation (2.23),

$$\phi(\theta \mid x_{1:T}^{b_{1:T}}, b_{1:T}, y_{1:T}) = \frac{p(\theta, x_{1:T} \mid y_{1:T})}{\int p(\theta, x_{1:T} \mid y_{1:T}) d\theta}. \tag{2.24}$$

And Step (ii) is given by

$$\phi(\mathbf{x}_{1:T}^{-b_{1:T}}, \mathbf{a}_{2:T}^{-b_{2:T}} \mid \theta, x_{1:T}^{b_{1:T}}, b_{1:T}) = \prod_{i=1, i \neq b_1}^{N} \pi_\theta(x_1^i) \prod_{t=2}^{T} \prod_{i=1, i \neq b_t}^{N} M_{t,\theta}(x_t^i, a_t^i). \tag{2.25}$$

In order to accomplish this step, we need to do some modification on the Particle Filter in Algorithm 1. In Equation (2.24) , we specified an ancestral path of particles, which leads to exclude the indices $b_{1:T}$ in Equation (2.25). Thus, the trajectory $x_{1:T}^{b_{1:T}}$, which is the reference trajectory, is not changed every time. We can summarize this step as a conditional Particle Filter in Algorithm 2.

Algorithm 2 Conditional Particle Filter (conditioned on $x'_{1:T}$)

Draw $x_1^i \sim \pi_\theta(x_1)$ for $i = 1, \cdots, N-1$.
Set $x_1^N = x'_1$.
Calculate the weight function $w_1^i = \alpha_1^i(x_1^i)$ for $i = 1, \cdots, N$, and get the normalized weight $W_1^i = w_1^i / \sum_{j=1}^N w_1^j$.
for $t = 2 : T$ **do**
    Draw $\{x_t^i, a_t^i\} \sim M_{t,\theta}(a_t, x_t)$ for $i = 1, \cdots, N-1$.
    Set $a_t^N = N$ and $x_t^N = x'_t$.
    Set $x_{1:t}^i = \{x_{1:t-1}^{a_t^i}, x_t^i\}$ for $i = 1, \cdots, N$.
    Calculate $w_t^i = \alpha_t^i(x_{1:t}^i)$ for $i = 1, \cdots, N$.
    Compute normalized weight $W_t^i = w_t^i / \sum_{j=1}^N w_t^j$ for $i = 1, \cdots, N$.
end for

For Step (iii), we have

$$\phi(k \mid \theta, \mathbf{x}_{1:T}, \mathbf{a}_{2:T}) \propto \phi(\theta, \mathbf{x}_{1:T}, \mathbf{a}_{2:T}, k) \propto W_T^k. \tag{2.26}$$

The detailed proof can be found at Andrieu et al. (2010). Now we summarize the Particle Gibbs sampler in Algorithm 3. Note that we fixed the reference trajectory $b_{1:T} = (N, \cdots, N)$ in the Particle Gibbs algorithm and it can be fixed to some arbitrary trajectory.

Algorithm 3 Particle Gibbs

Set the initial value $\theta^{[0]}$ and $x_{1:T}^{[0]}$ arbitrarily.
for $r \geq 1$ **do**
    Draw $\theta^{[r]} \sim \pi_T(\theta \mid x_{1:T}^{[r-1]})$.
    Run a conditional Particle Filter (Algorithm 2) targeting $\pi_T^{\theta^{[r]}}(x_{1:T})$, conditioned on $x_{1:T}^{[r-1]}$.
    Sample k with $P(k = i) = W_T^i$.
    Set $x_{1:T}^{[r]} = x_{1:T}^k$.
end for

The Particle Gibbs algorithm allows us to build efficient high dimensional proposal distribution by the Sequential Monte Carlo methods and makes the Bayesian inference more reliable. However,

the Particle Gibbs still has a serious drawback. The path degeneracy occurs in high dimensional problems and the mixing of the Markov kernel can be very poor. One way to address this problem is adding a backward simulation sweep to the Particle Gibbs algorithm, which leads to the Particle Gibbs with Backward Simulation (PGBS) method. On the other hand, we can add an ancestor sampling step to the Particle Gibbs algorithm and we get the Particle Gibbs with ancestor Sampling (PGAS) algorithm.

2.3.2 Ancestor Sampling

Recall that the Particle Gibbs sampler fixes a reference trajectory in Particle Filter. However, Particle Gibbs still suffers from the path degeneracy problem, which limits Particle Gibbs's application. The Particle Gibbs with Ancestor Sampling (PGAS) proposed by Lindsten et al. (Lindsten et al., 2014) gives a significant improvement to mixing and alleviates the path degeneracy problem. Without the backward sweep, the PGAS achieves the same effect as PGBS.

Based on the extended target density (2.23) on $\Omega = \Theta \times \mathcal{X}^{NT} \times \{1, \cdots, N\}^{N(T-1)+1}$, we introduce the following sweep:

(i) Draw $\theta' \sim p(\theta \mid x_{1:T}^{b_{1:T}}, b_{1:T})$

(ii) Draw $\mathbf{x}_1^{\prime,-b_1} \sim p(\mathbf{x}_1^{-b_1} \mid \theta, x_{1:T}^{b_{1:T}}, b_{1:T})$ and then
draw $\{\mathbf{x}_t^{\prime,-b_t}, \mathbf{a}_t^{\prime,-b_t}\} \sim \phi(\mathbf{x}_t^{-b_t}, \mathbf{a}_t^{-b_t} \mid \theta', \mathbf{x}_{1:t-1}^{\prime,-b_{1:t-1}}, \mathbf{a}_{2:t-1}', x_{1:T}^{b_{1:T}}, b_{t-1:T})$;
$b_{t-1}' \sim \phi(b_{t-1} \mid \theta', \mathbf{x}_{1:t-1}^{\prime,-b_{1:t-1}}, \mathbf{a}_{2:t-1}', x_{1:T}^{b_{1:T}}, b_{t:T})$ for $t = 2, \cdots, T$

(iii) Draw $b_T' \sim \phi(b_T \mid \theta', \mathbf{x}_{1:T}^{\prime,-b_{1:T}}, \mathbf{a}_{2:T}', x_{1:T}^{b_{1:T}})$.

In the sweep above, we randomly generated a new value for the ancestor index $a_t^{b_t}$ at each iteration of the conditional Particle Filter. This step is called Ancestor Sampling, which is a small modification of the Particle Gibbs algorithm. By the ancestor sampling step, Particle Gibbs with Ancestor Sampling can significantly improve the mix rate in the algorithm. In order to apply this step, for time $t \geq 2$, we artificially connect the reference trajectory $x_{t:T}'$ to one of the particle paths $\{x_{1:t-1}^i\}_{i=1}^N$ by sampling a_t^N with $P(a_t^N = i) \propto w_{t-1|T}^i$, where

$$w_{t-1|T}^i := w_{t-1}^i \frac{p(\{x_{1:t-1}^i, x_{t:T}'\}, y_{1:T})}{p(x_{1:t-1}^i, y_{1:t-1})} \tag{2.27}$$

for $i = 1, \cdots, N$. Based on Lemma 2 in Lindsten et al. (2014), Step (ii) and Step (iii) leave ϕ_θ invariant since it properly collapses. Furthermore, we have the theorem that the sweep above leaves $p_\theta(x_{1:T} \mid y_{1:T})$ invariant. The detailed proof can be found in Lindsten et al. (2014).

For the State Space Model, the weight function at time t is

$$w_t^i = \frac{g_\theta(y_t \mid x_t^i) f_\theta(x_t^i \mid \tilde{x}_{t-1}^i)}{q_t(x_t^i \mid \tilde{x}_{t-1}^i, y_t)} \tag{2.28}$$

where $i = 1, \cdots, N$. Note that $\{\tilde{x}_{1:t-1}^i\}_{i=1}^{N-1}$ can be generated by sampling $N-1$ times with replacement from $\{x_{1:t-1}^i\}_{i=1}^{N-1}$ with probabilities proportional to the important weights $\{w_{t-1}^i\}_{i=1}^N$ and set $\tilde{x}_{1:t-1}^N$ as $x_{1:t-1}^J$ after we sampled the ancestor indices J. Then, the ancestor weights become

$$w_{t-1|T}^i = w_{t-1}^i p(x'_{t:T}, y_{t:T} \mid x_{t-1}^i) \propto w_{t-1}^i f(x'_t \mid x_{t-1}^i) \tag{2.29}$$

for $i = 1, \cdots, N$. Similar to the Particle Gibbs sampler, we summarize the conditional Particle Filter with Ancestor Sampling for State Space Model in Algorithm 4 and then summarize the PGAS algorithm for Bayesian Learning of State Space Model in Algorithm 5. We will apply it to the SV models in the Chapter 4.

Algorithm 4 Conditional Particle Filter with Ancestor Sampling for State Space Model

Input: Conditional trajectory $x'_{1:T}$ and parameter $\boldsymbol{\theta} \in \Theta$.

Draw $x_1^i \sim \pi_\theta(x_1)$ for $i = 1, \cdots, N-1$.

Set $x_1^N = x'_1$.

Calculate the weight function $w_1^i = g_\theta(y_1 \mid x_1^i)\mu_\theta(x_1^i)/q_1(x_1^i \mid y_1)$ for $i = 1, \cdots, N$, and get the normalized weight $W_1^i = w_1^i / \sum_{j=1}^N w_1^j$.

for $t = 2$ **to** T **do**

  Draw with replacement $N-1$ times from $\{x_{1:t-1}^i\}_{i=1}^N$ with probability $\{W_{t-1}^i\}_{i=1}^N$ and get $\{\tilde{x}_{1:t-1}^i\}_{i=1}^{N-1}$.

  Draw J with

$$P(J = i) = \frac{w_{t-1}^i f_\theta(x'_t \mid x_{t-1}^i)}{\sum_{m=1}^N w_{t-1}^m f_\theta(x'_t \mid x_{t-1}^m)}, \quad i = 1, \cdots, N.$$

  Set $\tilde{x}_{1:t-1}^N = x_{1:t-1}^J$.

  Sample particles $x_t^i \sim q_t(x_t \mid \tilde{x}_{t-1}^i, y_t)$ for $i = 1, \cdots, N-1$ and set $x_t^N = x'_t$.

  Set $x_{1:t}^i = \{\tilde{x}_{1:t-1}^i, x_t^i\}$ for $i = 1, \cdots, N$.

  Calculate the weight $w_t^i = g_\theta(y_t \mid x_t^i) f_\theta(x_t^i \mid \tilde{x}_{t-1}^i)/q_t(x_t^i \mid \tilde{x}_{t-1}^i, y_t)$ for $i = 1, \cdots, N$.

  Normalize the weight $W_t^i = w_t^i / \sum_{j=1}^N w_t^j$ for $i = 1, \cdots, N$.

end for

Sample k with $P(k = i) = W_T^i$.

Return: $x_{1:T}^k$

Algorithm 5 Particle Gibbs with Ancestor Sampling for Bayesian Learning of SSM

Set the initial value of $\theta^{[0]}$ and $x_{1:T}^{[0]}$ arbitrarily.

for $r \geq 1$ **do**

  Draw $x_{1:T}^{[r]}$ by Conditional Particle Filter with Ancestor Sampling (Algorithm 4), conditioned on $x_{1:T}^{[r-1]}$.

  Draw $\theta^{[r]} \sim p_{\theta^{[r-1]}}(\theta \mid x_{1:T}^{[r-1]}, y_{1:T})$.

end for

3.0 STOCHASTIC VOLATILITY MODEL

In this Chapter, we will introduce the Stochastic Volatility (SV) models, which are widely used in the field of mathematical finance to evaluate derivative securities. The Black-Scholes model assumes that the underlying volatility is constant over the life of the derivative (Black and Scholes, 1973). Although the volatility is never constant long term, it can be relatively constant in a very short term. SV models treat the volatility as a random process, which is one approach to resolving a shortcoming of the Black-Scholes model. In Section 3.1, we will review the SV models and one approach to solve them. In Section 3.2, we will apply the SV models to real world data and discuss the problem we are facing.

3.1 EMPIRICAL VERSION OF STOCHASTIC VOLATILITY MODEL

The SV model builds on the stock price and its volatility. Introducing the latent stochastic process on the volatility, we can build a model for stock return price by stochastic differential equations. In order to estimate the parameters in the model, we can transform the continuous-time SV model into the discrete-time SV model. Then, Bayesian inference by MCMC will become available to estimate the parameters.

3.1.1 Model Introduction

From the assumptions in Wiggins (1987), the stock price S and the instantaneous standard deviation σ satisfy the following stochastic differential equations:

$$dS(t) = \mu S(t)dt + \sigma(t)S(t)dz_s, \tag{3.1}$$

$$d\sigma(t) = f(\sigma(t))dt + \theta\sigma(t)dz_\sigma, \tag{3.2}$$

$$\rho dt = dz_s dz_\sigma, \tag{3.3}$$

where dz_s and dz_σ are standard Wiener process and the correlation coefficient between stock returns and volatility movements is as Equation (3.3). The model above is known as continuous-time SV model (Taylor, 1994). In particular, for logarithm of the volatility, we have the most popular continuous-time model

$$\frac{dS(t)}{S(t)} = \mu dt + \sigma(t)dz_s, \tag{3.4}$$

$$d(log(\sigma(t))) = \lambda(log(\sigma(t)) - \xi)dt + \theta dz_\sigma, \tag{3.5}$$

$$\rho dt = dz_s dz_\sigma, \tag{3.6}$$

where μ, λ, ξ, θ and ρ are constant parameters. When the stock price data are observed in discrete and regular-spaced time interval, we can also work with the discrete-time SV model

$$x_t = \mu + \phi(x_{t-1} - \mu) + \omega_t \tag{3.7}$$

$$y_t = \beta \exp\{\frac{x_t}{2}\}\epsilon_t, \tag{3.8}$$

where $x_1 \sim N(\mu, \frac{\sigma_\omega^2}{1-\phi^2})$, $\omega_t \sim i.i.dN(0, \sigma_\omega^2)$, $\epsilon_t \sim i.i.dN(0,1)$ and $t = 1, \cdots, T$. The detailed derivation of this discrete-time SV model can be found at Duffie and Singleton (1990), Wiggins (1987) and Chesney and Scott (1989). In the model above, x_t is the logarithm of the volatility at time t, which follows a stationary process with $|\phi| < 1$ or random walk if $\phi = 1$. y_t is the return or relative gain, which is defined as

$$y_t = \frac{S_t - S_{t-1}}{S_{t-1}}, \tag{3.9}$$

where S_t is the stock price at time t. If the return represents a small percentage change, then we can also define return as $y_t \approx \nabla log(S_t)$. ω_t and ϵ_t are uncorrelated standard normal white noise. The parameter β can be treated as a constant scaling factor, which can also be interpreted as model instantaneous volatility. The constant μ is sometimes called a *leverage effect* and hence the model

is also called *SV model with leverage*. In order to avoid overparameterizing the SV model, we simply set $\beta = \exp(\mu/2)$, which is consistent with the method in Kim et al. (1998). Since the model has three unknown parameters, we will later on refer to this model as the *three-parameter SV model*. If we set $\mu = 0$ to further avoid overparameterization, the SV model will only have two unknown parameters and we hereafter call it *two-parameter SV model*. The detailed econometric properties are discussed in Taylor (2007), Shephard (1996) and Taylor (1994).

3.1.2 Multivariate Stochastic Volatility Model

In the real world, we can assume that certain stock price series can receive an analogous signal x_t, which is the hidden state in Hidden Markov Model. The different behaviors of their stock price are determined by some scale parameters and by the independent white noise. Thus, we need to introduce the Multivariate Stochastic Volatility (MSV) model in Asai et al. (2006),

$$x_t = \phi x_{t-1} + \omega_t \tag{3.10}$$

$$\boldsymbol{y}_t = \boldsymbol{\beta} \exp\{\frac{x_t}{2}\}\boldsymbol{\epsilon}_t, \tag{3.11}$$

where $\boldsymbol{y}_t = \begin{pmatrix} y_{1t} \\ \vdots \\ y_{pt} \end{pmatrix}$, $\boldsymbol{\beta} = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_p \end{pmatrix}$, $\boldsymbol{\epsilon}_t = \begin{pmatrix} \epsilon_{1t} \\ \vdots \\ \epsilon_{pt} \end{pmatrix}$, $\omega_t \sim i.i.d.N(0, \sigma_\omega^2)$, $\epsilon_{it} \sim i.i.d.N(0,1)$, $i = 1, \cdots, p$ and $t = 1, \cdots, T$. Note that we remove μ from the univariate SV model in order to avoid over-parameterization. Then, the conditional distribution of $\boldsymbol{y}_t \mid x_{1:t}$ is

$$\boldsymbol{y}_t \mid x_{1:t} \sim \boldsymbol{N}_p\left(\boldsymbol{0}, \boldsymbol{\Sigma} = \begin{bmatrix} \beta_1^2 \exp(x_t) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \beta_p^2 \exp(x_t) \end{bmatrix}\right). \tag{3.12}$$

In the MSV model, β_is stand for the variability of y_{it}. If the series has higher variability than others, it will have a larger β. Sometimes we can compare the variability by the estimated value of β_is.

3.1.3 Approaches to Solve the Stochastic Volatility Model

For the discrete-time SV model in (3.7)-(3.8), there are several methods to estimate the unknown parameters. For example, Quasi-likelihood method was proposed by Harvey et al. (1994), which transforms the SV model into a linear model and employs the Kalman filtering to estimate the parameters by maximizing the quasi likelihood function. Although the quasi-likelihood estimator is consistent and has asymptotic normality, it has a serious drawback when we use the normal distribution to approximate the $log(\epsilon_t^2)$.

On the other hand, one popular collection of methods to solve the discrete-time SV model is based on the Markov Chain Monte Carlo (MCMC) and Bayesian inference (Jacquier et al. (1994) and Kim et al. (1998)). Writing the model as Non-Linear Non-Gaussian State Space Model in section 2.1.2, we have

$$y_t \mid x_t, \Theta \sim N(0, \beta^2 exp\{x_t\}) \tag{3.13}$$

$$x_t \mid x_{t-1}, \Theta \sim N(\mu + \phi(x_{t-1} - \mu), \sigma_\omega^2), \tag{3.14}$$

where $x_1 \mid \Theta \sim N(\mu, \frac{\sigma_\omega^2}{1-\phi^2})$, $\Theta = \{\mu, \phi, \sigma_\omega, \beta\}$ and $t = 1, \cdots, T$. We can then get the estimated parameters by the MCMC frameworks for State Space Models. The frameworks were introduced in Chapter 2.

To fit the three-parameter SV model, where $\beta = \exp(\mu/2)$, we need to sample $\Theta = \{\mu, \phi, \sigma_\omega\}$ from the posterior distributions to complete Step 4 in Algorithm 5. Then, I will review several popular prior distributions on parameter $\{\mu, \phi, \sigma_\omega\}$.

Sampling σ_ω^2:

To sampling σ_ω^2, Kim, Shephard and Chib (Kim et al., 1998) suggested an Inverse Gamma conjugate prior distribution on σ_ω^2,

$$\sigma_\omega^2 \mid \phi, \mu \sim \mathcal{IG}(\frac{\sigma_1}{2}, \frac{S_\sigma}{2}), \tag{3.15}$$

where $\mathcal{IG}$ denotes the Inverse Gamma distribution. σ_1 and S_σ are called hyperparameters. Then, we can sample the σ_ω^2 from posterior distribution

$$\sigma_\omega^2 \mid y_{1:T}, x_{1:T}, \phi, \mu \sim \mathcal{IG}(\frac{T + \sigma_1}{2}, \frac{S_\sigma + (x_1 - \mu)^2(1 - \phi^2) + \sum_{t=1}^{T-1}((x_{t+1} - \mu) - \phi(x_t - \mu))^2}{2}). \tag{3.16}$$

Sampling ϕ:

One popular method denotes $\phi = 2\phi^* - 1$ and assumes ϕ^* is distributed as Beta$(\alpha_\phi, \beta_\phi)$,

$$\phi \mid \sigma_\omega^2, \mu \sim Beta(\alpha_\phi, \beta_\phi), \tag{3.17}$$

where α_ϕ and β_ϕ are hyperparameters. The full conditional density of ϕ is proportional to $\pi(\phi) f(x_{1:T} \mid \mu, \phi, \sigma_\omega^2)$, where

$$\log f(x_{1:T} \mid y_{1:T}, \mu, \phi, \sigma_\omega^2) \propto -\frac{(x_1-\mu)^2(1-\phi^2)}{2\sigma_\omega^2} + \frac{1}{2}\log(1-\phi^2) - \frac{\sum_{t=1}^{T-1}((x_{t+1}-\mu)-\phi(x_t-\mu))^2}{2\sigma_\omega^2}. \tag{3.18}$$

Then, ϕ can be sampled using an Acceptance-Rejection algorithm, because the function (3.18) is concave in ϕ for all values of α_ϕ and β_ϕ. Kim et al. (1998) suggests employing a first order Taylor expansion of the prior about

$$\hat{\phi} = \frac{\sum_{t=1}^{T-1}(x_{t+1}-\mu)(x_t-\mu)}{\sum_{t=1}^{T-1}(x_t-\mu)^2}, \tag{3.19}$$

and combining with $f(x_{1:T} \mid y_{1:T}, \mu, \phi, \sigma_\omega^2)$. Then, we can get a good suggestion density for the Acceptance-Rejection algorithm.

On the other hand, we can employ a Normal prior distribution on ϕ,

$$\phi \mid \sigma_\omega^2, \mu \sim N(\mu_\phi, \sigma_\phi^2), \tag{3.20}$$

where μ_ϕ and σ_ϕ^2 are hyperparameters. Note that $\beta_\phi^2 \to \infty$ gives an non-informative flat prior. Then, we can sample the ϕ from posterior distribution

$$\phi \mid y_{1:T}, x_{1:T}, \sigma_\omega^2, \mu \sim N(A, B), \tag{3.21}$$

$$A = B\left(\frac{\sum_{t=1}^{T-1}(x_{t+1}-\mu)(x_t-\mu)}{\sigma_\omega^2} + \frac{\mu_\phi}{\sigma_\phi^2}\right) \tag{3.22}$$

$$B = \left(\frac{\sum_{t=1}^{T-1}(x_t-\mu)^2 - (x_1-\mu)^2}{\sigma_\omega^2} + \frac{1}{\sigma_\phi^2}\right)^{-1} \tag{3.23}$$

Sampling μ:

Suppose μ has the non-informative prior, then we can sample μ from

$$\mu \mid y_{1:T}, x_{1:T}, \phi, \sigma_\omega^2 \sim N(\mu_1, \sigma_\mu^2), \tag{3.24}$$

where

$$\mu_1 = \sigma_\mu \left\{ \frac{1-\phi^2}{\sigma_\omega^2} x_1 + \frac{1-\phi}{\sigma_\omega^2} \sum_{t=1}^{T-1} (x_{t+1} - \phi x_t) \right\}$$

and

$$\sigma_\mu^2 = \frac{\sigma_\omega^2}{(T-1)(1-\phi)^2 + (1-\phi^2)}.$$

In conclusion, by the three separated steps above, we can generate $\Theta^{[r]} \sim p_{\Theta^{[r-1]}}(\Theta \mid x_{1:T}[r-1], y_{1:T})$. Then, we can get the estimated parameters by the posterior distribution. If we use the Mean Square Error (MSE) as risk function, which is also called squared error risk,

$$MSE := E[(\hat{\Theta} - \Theta)^2], \tag{3.25}$$

the Bayes estimator of the unknown parameter is simply the mean of the posterior distribution.

3.2 A CASE STUDY

In this section, we perform the Bayesian inference using PGAS algorithm for the Standard & Poor's 500 (S&P 500) data. From the case study, we will see the serious drawback of the original method and then we will introduce a new method to solve this problem in the next chapter.

The S&P 500 is an American stock market index. It's based on the market capitalizations of 500 large companies having common stock listed on the NYSE or NASDAQ. It is considered one of the best representations of the U.S. stock market. The S&P 500 data we used, which are shown in Figure 3.1, starts from January of 2005 and ends in October of 2011. The time interval of the data includes the Financial Crisis of 2008.

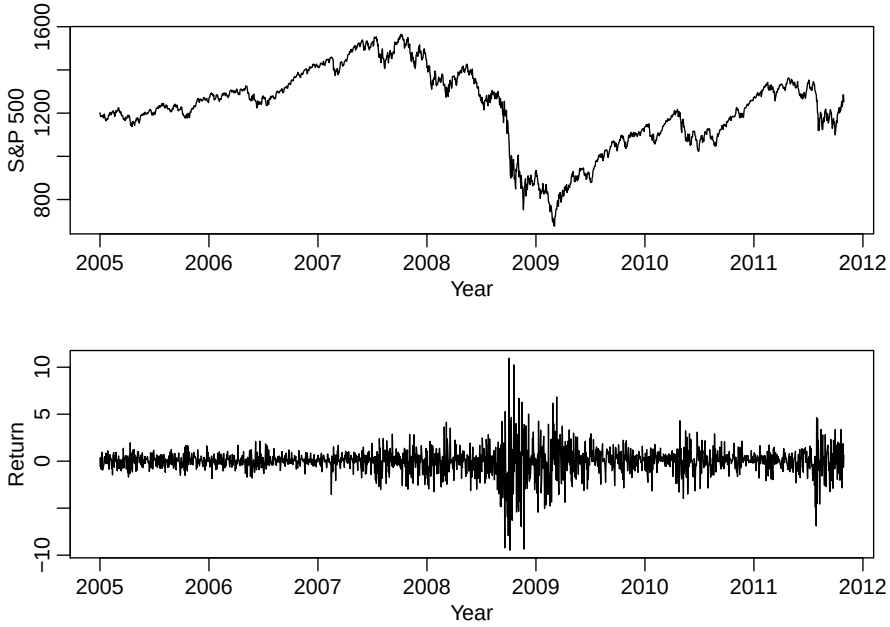

Figure 3.1: *S&P 500 daily index and daily return from 2005 to 2012. Upper: S&P 500 daily index; Lower: Daily return of S&P 500.*

There is a slow convergence problem reported in Chib and Greenberg (1996) and Kim et al. (1998), because sampled hidden states are highly correlated in the sampling procedure in Kim et al. (1998). The sampling procedure with PGAS in Section 3.1 ameliorates the slow convergence problem to some degree by sampling the whole trajectory of hidden states. However, this method

still suffers slow convergence. The reason is that ϕ and σ_ω^2 are highly correlated in the model. Either a larger σ_ω^2 or a larger ϕ can lead to a larger y_t. In such case, sampling σ_ω^2 and ϕ from the conditional distribution will produce a little movement in the draws. We can observe this slow convergence by the autocorrelations (ACF).

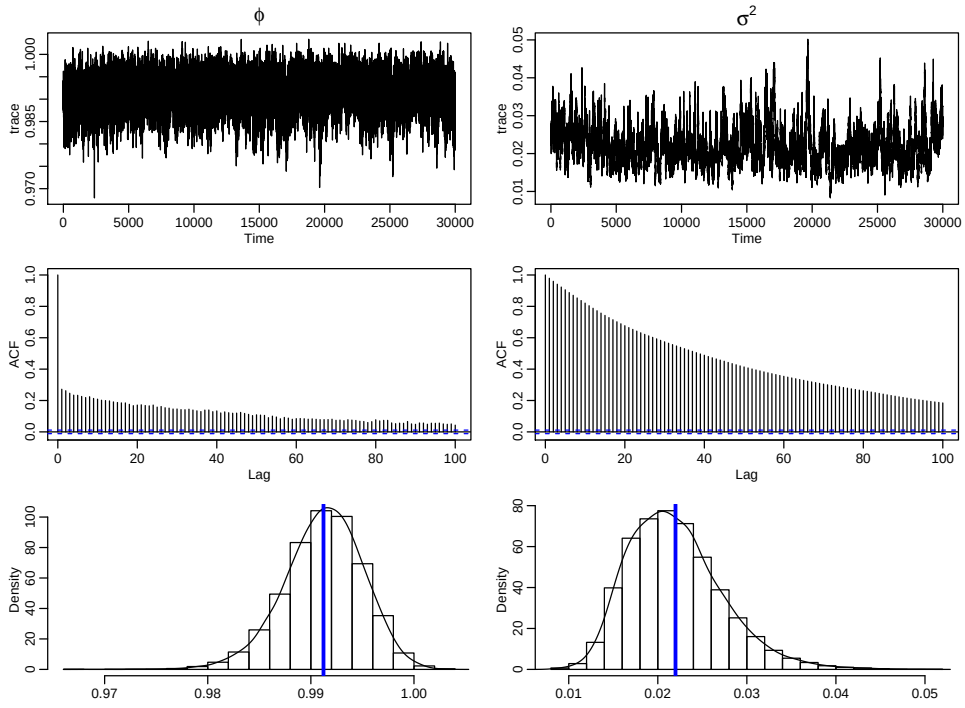

Figure 3.2: *A toy example: results for S&P 500 data using the two-parameter SV model. TOP: Trace plots of sampled parameters. MIDDLE: Autocorrelation plots (ACF) for sampled parameters. BOTTOM: Histograms of sampled parameters. The blue line shows the sample mean of parameter. In total 33,000 iterations were drawn and the first 3,000 discarded. The particles number for PGAS is* $N = 20$.

For illustrative purposes, we apply PGAS framework to the two-parameter SV model, which simply sets $\mu = 0$. That is,

$$x_t = \phi x_{t-1} + \omega_t \tag{3.26}$$

$$y_t = \exp\{\frac{x_t}{2}\}\epsilon_t, \tag{3.27}$$

where $x_1 \sim N(\mu, \frac{\sigma_\omega^2}{1-\phi^2})$, $\omega_t \sim i.i.dN(0, \sigma_\omega^2)$, $\epsilon_t \sim i.i.dN(0,1)$ and $t = 1, \cdots, T$. The unknown parameters are ϕ and σ_ω^2.

We iterate the algorithm for 33,000 iterations and burn in the first 3,000 iterations. Then, we get the results shown in Figure 3.2. We can notice that the autocorrelations decay very slowly for ϕ and σ_ω^2. In order to evaluate the mixing, we calculate the *inefficiencies*, which is defined as

$$\text{IF} := 1 + 2\sum_{i=1}^{\infty} \rho(i), \tag{3.28}$$

where $\rho(i)$ is the autocorrelation function at lag i. In this thesis, we use the *initial monotone sequence estimator* (Geyer, 1992) to estimate the inefficiencies. The estimated inefficiencies for μ and σ_ω^2 are 32.918 and 131.958, respectively. The average of inefficiencies for the two parameters is 82.438.

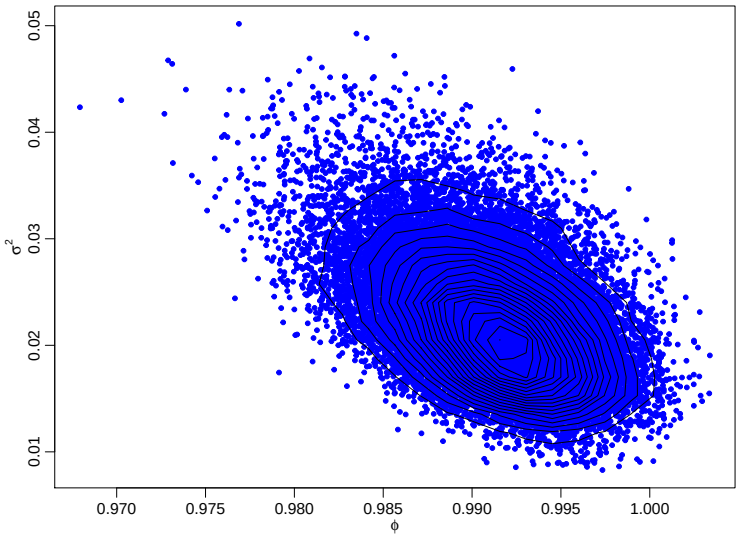

Figure 3.3: *The contour plot for sampled parameters ϕ and σ_ω^2. There is a high negative correlation between the two parameters.*

From the contour plot in Figure 3.3, there is a very clear negative correlation between ϕ and σ_ω^2 and the sample correlation is -0.493. This is the reason for slow convergence and slow decay of autocorrelations and we will illustrate the reason in next chapter before we introduce a new method. In addition, from the Figure 3.4, we can see the increased value of hidden state x_t when

the volatility y_t becomes large. In particular, large x_t corresponds to the high volatility during the 2008 financial crisis.

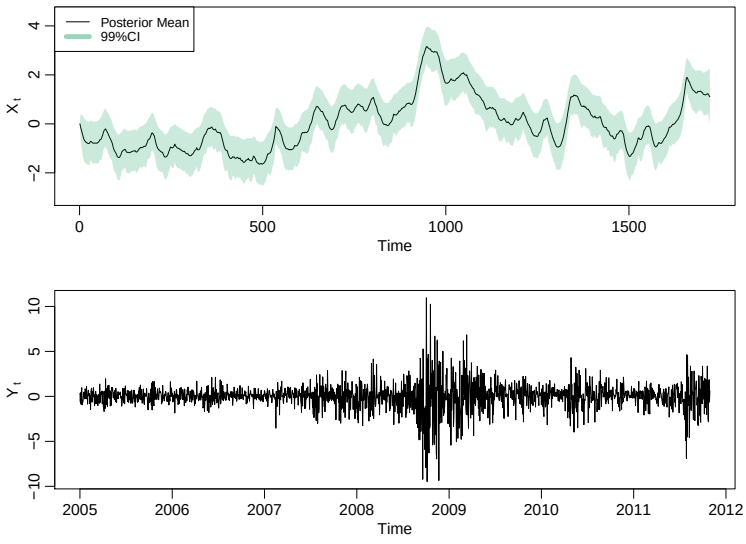

Figure 3.4: *A toy example: the posterior mean plot of x_t and daily return of S&P 500. TOP: The posterior mean plot of x_t with 99% credible intervals. BOTTOM: Daily return of S&P 500.*

4.0 PARTICLE GIBBS WITH ANCESTOR SAMPLING IN STOCHASTIC VOLATILITY MODELS

In this chapter, we are going to propose a new method to perform the Bayesian inference for SV model in (3.7) and (3.8). First of all, a new efficient method will be proposed in Section 4.1 to solve the slow decay of autocorrelations, which was discussed in Section 3.2. Then, we combine the Adaptive Markov Chain Monte Carlo algorithm with our proposed method in Section 4.2. At last, the proposed method will be extended and applied to MSV Model in Section 4.3.

4.1 PROPOSED METHOD FOR STOCHASTIC VOLATILITY MODELS

In the Section 3.2, we applied the Particle Gibbs with Ancestor Sampling to solve the SV model (3.7 - 3.8) in Section 3.1. Although the PGAS solves the slow convergence problems reported by Kim et al. (1998), we still observed slow convergence caused by the high negative correlation between ϕ and σ_ω^2. In this section, we will propose a new method to improve the convergence.

4.1.1 New Methodology for Sampling Parameters Jointly

From Section 3.1.3, to solve the SV models, people usually put normal priors on the leverage term μ and (or beta priors) on the regression parameter ϕ, and inverse gamma priors on the scale parameter σ_ω. That is, current methods treat ϕ as a regression parameter and σ_ω as a scale parameter. Unfortunately, this treatment is exactly leading to the inefficiency for this particular model.

The problem for SV models is that ϕ plays two roles in SV model. That is, ϕ behaves like a

regression parameter as well as a scale parameter. For example, the autocorrelation function (ACF) of y_t^2 is given by

$$cor(y_t^2, y_{t+h}^2) = \frac{\exp(\sigma_x^2 \phi^h) - 1}{\kappa_\epsilon \exp(\sigma_x^2) - 1}, \tag{4.1}$$

where κ_ϵ is the kurtosis of the noise, ϵ_t and $\sigma_x^2 = \sigma_\omega^2/(1-\phi^2)$. For the two-parameter SV model, $\kappa_\epsilon = 3$. The ACF values are small and the decay rate as a function of lag is less than exponential and somewhat linear. This means that if you specify values for ϕ but allow us to control σ_ω (and consequently σ_x), we can always find a good σ_ω, which makes the model ACF to look approximately the same, no matter which values of ϕ are given. This is accomplished by moving ϕ and σ_ω in opposite directions.

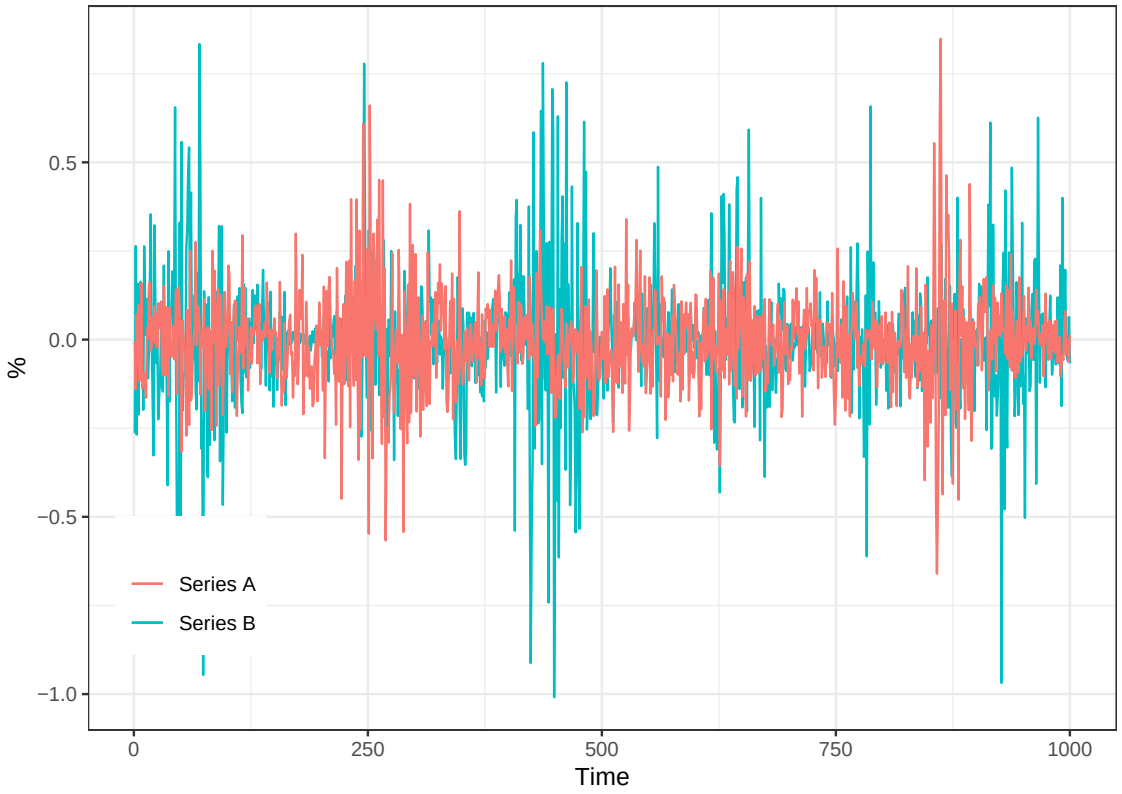

Figure 4.1: *Two data sequences (A and B) of length 1000 generated from different two-parameter SV models. Model I:* $\phi = 0.99$, $\sigma_\omega = 0.15$ *and Model II:* $\phi = 0.38$, $\sigma_\omega = 1$.

Another way of looking at the problem is to let $\xi_t = \frac{1}{\sigma_x} x_t$ and $\zeta_t = \frac{1}{2} w_t$. Then we may write

(3.27) as

$$y_t = \exp(\sigma_x \phi \xi_{t-1}) \exp(\sigma_\omega \zeta_t) \epsilon_t, \tag{4.2}$$

where ξ_{t-1} and ζ_t are independent stationary $\frac{1}{2}N(0,1)$s. It is clear from (4.2) that σ_ω and ϕ are scale parameters of the ξ_t process and σ_ω is a scale parameter of the ζ_t noise process. For any pair of σ_ω and ϕ, we can always find other pairs of σ_ω and ϕ to make the scale of y_t to look approximately the same. That is, we can keep the scale of the data approximately the same by moving σ_ω and ϕ in opposite directions.

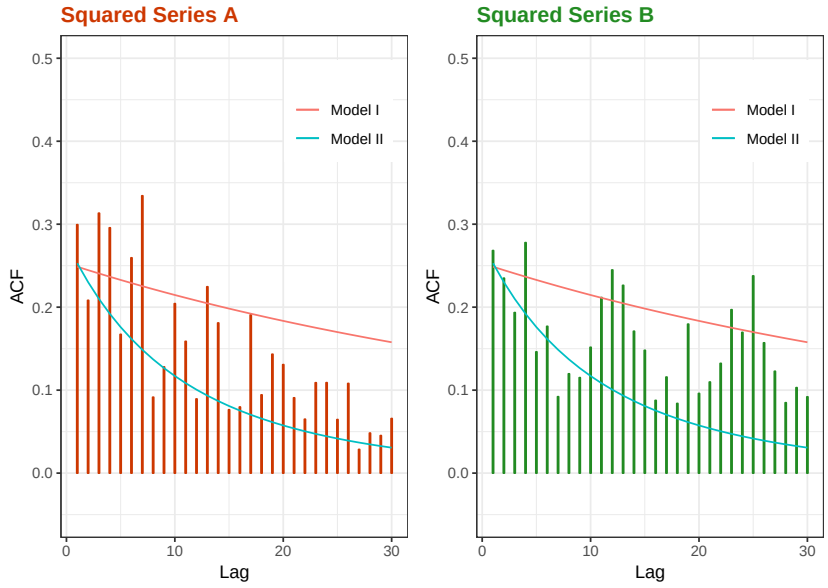

Figure 4.2: *The ACF of each generated series squared (A and B) and the theoretical ACFs of SV models I and II.*

For example, Figure 4.1 shows two data sequences of length 1000 generated from two different SV models. With $\mu = 0$ and $\beta = 0.1$, Model I employs $\sigma_\omega = 0.15$ and $\phi = 0.99$ and Model II uses $\sigma_\omega = 0.38$ and $\phi = 0.95$. Series A and Series B are corresponding to Model I and Model II, respectively. The ACF of each generated series squared (A and B) are shown as bars in Figure 4.2. The theoretical ACFs of Model I and II are displayed as lines. Note that the ACF of Model I is above the one for Model II. While the AR parameter ϕ is very different in each model, the simulated series look very much the same.

Thus, the intuition of our method is to try to treat ϕ and σ_ω as a whole since they are highly

correlated with each other. That is, rather than sample ϕ and σ_ω individually, it would be better to sample them at the same time from the posterior distribution. Compared with sampling ϕ and σ_ω separately, this small change will bring a big improvement to the convergence problem. The improvement will be shown by simulation and case studies in Chapter 5.

In order to sample ϕ and σ_ω together from the posterior distribution, we have to find a suitable joint prior distribution. Since ϕ and σ_ω can be any value in $\mathbb{R}^2$ and they are negative correlated, we can assume that they come from the bivariate normal distribution with negative correlation coefficient. That is,

$$\begin{pmatrix} \phi \\ \sigma_\omega \end{pmatrix} \sim N\left(\begin{bmatrix} \mu_\phi \\ \mu_{\sigma_\omega} \end{bmatrix}, \begin{bmatrix} \sigma_\phi^2 & \rho\sigma_\phi\sigma_q \\ \rho\sigma_\phi\sigma_q & \sigma_q^2 \end{bmatrix} \right), \tag{4.3}$$

where $\rho < 0$. The hyperparameters are μ_ϕ, μ_{σ_ω}, ρ, σ_ϕ and σ_q. Note that the high negative correlation between ϕ and σ_ω^2 can be explained by setting $\rho < 0$ in our new prior distribution.

By the definition of SV model in (3.7), we have

$$x_t \mid x_{t-1}, \mu, \phi, \sigma_\omega \sim N(\mu + \phi(x_{t-1} - \mu), \sigma_\omega^2) \tag{4.4}$$

$$x_1 \mid \mu, \phi, \sigma_\omega \sim N(\mu, \frac{\sigma_\omega^2}{1-\phi^2}), \tag{4.5}$$

where $t = 1, \cdots, T$. Then, the posterior distributions is

$$\begin{aligned}
f(\phi, \sigma_\omega \mid \mu, y_{1:T}, x_{1:T}) &\propto \pi(\phi, \sigma_\omega) L(x_1, \cdots, x_T \mid \phi, \sigma_\omega, \mu) \\
&\propto \pi(\phi, \sigma_\omega) f(x_1 \mid \phi, \sigma_\omega, \mu) \prod_{i=2}^{T} f(x_i \mid x_{i-1}, \phi, \sigma_\omega, \mu) \\
&\propto \exp\left\{ -\frac{1}{2(1-\rho^2)} \left[\frac{(\phi-\mu_\phi)^2}{\sigma_\phi^2} + \frac{(\sigma_\omega - \mu_{\sigma_\omega})^2}{\sigma_q^2} - \frac{2\rho(\phi-\mu_\phi)(\sigma_\omega - \mu_{\sigma_\omega})}{\sigma_\phi \sigma_q} \right] \right\} \\
&\cdot \frac{\sqrt{1-\phi^2}}{\sigma_\omega} \exp\left\{ -\frac{(x_1-\mu)^2}{2\sigma_\omega^2/(1-\phi^2)} \right\} \prod_{i=2}^{T} \frac{1}{\sigma_\omega} \exp\left\{ -\frac{[(x_i-\mu) - \phi(x_{i-1}-\mu)]^2}{2\sigma_\omega^2} \right\} \\
&\propto \exp\left\{ -\frac{(\phi-\mu_\phi)^2\sigma_q^2 + (\sigma_\omega - \mu_{\sigma_\omega})^2\sigma_\phi^2 - 2\rho\sigma_\phi\sigma_q(\phi-\mu_\phi)(\sigma_\omega - \mu_{\sigma_\omega})}{2(1-\rho^2)\sigma_\phi^2\sigma_q^2} \right\} \\
&\cdot \frac{\sqrt{1-\phi^2}}{\sigma_\omega^T} \exp\left\{ -\frac{(1-\phi^2)(x_1-\mu)^2 + \sum_{i=2}^{T}[(x_i-\mu) - \phi(x_{i-1}-\mu)]^2}{2\sigma_\omega^2} \right\}.
\end{aligned} \tag{4.6}$$

In this case, however, we must work with non-conjugate model in (4.6) by employing Metropolis Hastings algorithm. This is feasible because the complete data density $p(\boldsymbol{\xi}, \mu, x_{1:n}, y_{1:n})$ can be evaluated pointwise, where $\boldsymbol{\xi} = (\phi, \sigma_\omega)$.

4.1.2 Random Walk Metropolis Hastings

In order to obtain a sequence of random samples jointly from the posterior distribution in (4.6), Metropolis Hastings (MH) algorithm can be used as a Markov Chain Monte Carlo (MCMC) method. The Metropolis Hastings has an advantage in sampling from probability distribution, where direct sampling is difficult. The Random Walk Metropolis Hastings (RWMH) is a special case of the Metropolis Hastings (MH) algorithm.

Assume that π is a probability density of interest on $\mathbb{R}^n$. The Metropolis Hastings algorithm is used to produce a Markov chain with stationary density π. Let $q(\cdot \mid \xi)$ be a proposal density that is easy to sample from. The Metropolis Hastings algorithm proceeds as sweep follows:

(i) Assume the current state is $\xi^{[r]}$.

(ii) Generate proposed value for the next state of the chain. A candidate $\eta^{[r+1]}$ is sampled from $q(\cdot \mid \xi^{[r]})$.

(iii) Compute the acceptance probability $\alpha(\xi^{[r]}, \eta^{[r+1]})$ by

$$\alpha(\xi, \eta) = \begin{cases} min\Big(\frac{\pi(\eta)}{\pi(\xi)} \frac{q(\xi|\eta)}{q(z|\xi)}, 1\Big), & \text{if } \pi(\xi)q(\eta \mid \xi) > 0 \\ 1 & \text{if } \pi(\xi)q(\eta \mid \xi) = 0. \end{cases} \tag{4.7}$$

(iv) Accept the proposed value with probability α and set $\xi^{[r+1]} = \eta^{[r+1]}$, or reject and stay at the current state $\xi^{[r+1]} = \xi^{[r]}$.

This Metropolis Hastings algorithm produces a Markov chain, $\{\xi^{[r]}, t \in \mathbb{N}\}$, with transition kernel $P_q(\xi; C) := P(\xi^{[r-1]} \in C \mid \xi^{[r]} = \xi)$ given by

$$P_q(\xi; C) = \int \alpha(\xi, \eta) q(\eta \mid \xi) \mathbb{1}_A(\xi) d\eta + \mathbb{1}_A(\xi) \int q(\eta \mid \xi)[1 - \alpha(\xi, \eta)] d\eta. \tag{4.8}$$

The Random Walk Metropolis Hastings (RWMH) is based on the Metropolis Hastings (MH) algorithm. Let $\xi^{[r]}$ be constructed as

$$\eta^{[r+1]} = \xi^{[r]} + \epsilon_r, \tag{4.9}$$

where the ϵ_r are chosen to be i.i.d. with a symmetric distribution, for example, $\epsilon_r \sim$ iid $N(0, \Sigma_\epsilon)$. In this case, $q(\eta \mid \xi) = G_\epsilon(\eta - \xi) = q(\xi \mid \eta)$ and the RWMH algorithm is summarized in Algorithm 6. Theoretically, it has been shown that the ideal acceptance rate for a one-dimensional Gaussian

distribution is approximate 50%, decreasing to approximate 23.6% for a N-dimensional Gaussian target distribution (Roberts et al., 1997). Thus, we monitor and keep the acceptance rate at about 30% for the case studies in Chapter 5.

Algorithm 6 Random Walk Metropolis Hastings (RWMH)

1: Set the initial state $\xi^{[0]}$ arbitrarily.
2: **for** $r \geq 1$ **do**
3: Generate ϵ_r and let $\eta^{[r]} = \xi^{[r-1]} + \epsilon^{[r]}$.
4: Generate $U_r \sim unif(0,1)$.
5: **if** $U_r < \alpha(\xi^{[r-1]}, \eta^{[r]}) = min\Big(\frac{\pi(\eta^{[r]})}{\pi(\xi^{[r-1]})}, 1\Big)$ **then**
6: Set $\xi^{[r]} = \eta^{[r]}$.
7: **else**
8: Set $\xi^{[r]} = \xi^{[r-1]}$.
9: **end if**
10: **end for**

4.1.3 Proposed Method with PGAS Algorithm

Now, we are going to use the Random Walk Metropolis Hastings method to sample $\boldsymbol{\xi} = (\phi, \sigma_\omega)$ together from the target posterior distribution $f(\phi, \sigma_\omega \mid \mu, y_{1:T}, x_{1:T})$. The procedures are as follows:

(i) Draw $\boldsymbol{\epsilon} \sim \boldsymbol{N}_2(\mathbf{0}, \lambda\boldsymbol{\Sigma}_\epsilon)$ and set $\boldsymbol{\xi} = \boldsymbol{\xi}^{[r]} + \boldsymbol{\epsilon}$.

(ii) Compute the acceptance probability $\alpha = \frac{g(\boldsymbol{\xi})}{g(\boldsymbol{\xi}^{[r]})} \wedge 1$, where

$$g(\boldsymbol{\xi}) = \exp\left\{-\frac{(\phi-\mu_\phi)^2\sigma_q^2 + (\sigma_\omega-\mu_{\sigma_\omega})^2\sigma_\phi^2 - 2\rho\sigma_\phi\sigma_q(\phi-\mu_\phi)(\sigma_\omega-\mu_{\sigma_\omega})}{2(1-\rho^2)\sigma_\phi^2\sigma_q^2}\right\}$$
$$\cdot \frac{\sqrt{1-\phi^2}}{\sigma_\omega^T} \exp\left\{-\frac{(1-\phi^2)(x_1-\mu)^2 + \sum_{i=2}^n[(x_i-\mu)-\phi(x_{i-1}-\mu)]^2}{2\sigma_\omega^2}\right\}. \quad (4.10)$$

(iii) With probability α set $\boldsymbol{\xi}^{[r+1]} = \boldsymbol{\xi}$, otherwise set $\boldsymbol{\xi}^{[r+1]} = \boldsymbol{\xi}^{[r]}$.

Note that λ in Step (i) is a tuning parameter. In order to achieve a good mix rate, λ should be selected carefully.

For the parameter μ, we still assume μ has a non-informative prior, which is the same method as Section 3.1.3. That is, sample μ from

$$\mu \mid y_{1:T}, x_{1:T}, \phi, \sigma_\omega^2 \sim N(\mu_1, \sigma_\mu^2) \tag{4.11}$$

where

$$\mu_1 = \sigma_\mu \left\{ \frac{1-\phi^2}{\sigma_\omega^2} x_1 + \frac{1-\phi}{\sigma_\omega^2} \sum_{t=1}^{T-1} (x_{t+1} - \phi x_t) \right\}$$

and

$$\sigma_\mu^2 = \frac{\sigma_\omega^2}{(T-1)(1-\phi)^2 + (1-\phi^2)}.$$

By the two separated steps above, we can generate $\theta^{[r]} = (\phi, \sigma_\omega, \mu)^{[r]} \sim p_{\theta[r-1]}(\theta \mid x_{1:T}^{[r-1]}, y_{1:T})$, which is Step 4 in PGAS framework. Then, we can revise Algorithm 5 using our proposed method and summarize it in Algorithm 7.

Algorithm 7 Proposed Method for Stochastic Volatility Model

1: Set the initial value of $\boldsymbol{\theta}^{[0]}$ and $x_{1:T}^{[0]}$ arbitrarily.
2: **for** $r \geq 1$ **do**
3: Draw $x_{1:T}^{[r]}$ by Conditional Particle Filter with Ancestor Sampling (Algorithm 4), conditioned on $x_{1:T}^{[r-1]}$ and $\boldsymbol{\theta}^{[r-1]} = [\phi, \sigma_\omega, \mu]^{[r-1]}$.
4: With $x_{1:T}^{[r]}$, generate the pair $\boldsymbol{\xi}^{[r]} = [\phi, \sigma_\omega]^{[r]}$ by Random Walk Metropolis Hastings (Algorithm 6) with acceptance probability $\alpha = \frac{g(\boldsymbol{\xi})}{g(\boldsymbol{\xi}^{[r]})} \wedge 1$, where $g(\boldsymbol{\xi})$ is defined in (4.10).
5: With $x_{1:T}^{[r]}$ and $\boldsymbol{\xi}^{[r]}$, draw $\mu^{[r]}$ from $N(\mu_1, \sigma_\mu^2)$ in (4.11).
6: **end for**

By Algorithm 7, we can get the estimated parameters by Bayesian estimator. If we take the Mean Square Error (MSE) as risk function, the Bayesian estimator will be the mean of posterior distribution. Note that the Algorithm 7 is designed for the three-parameter SV model. We can easily accommodate the algorithm to two-parameter SV model by skipping sampling μ in Step 5.

In the Section 5.1, we will compare our proposed algorithm with original algorithm for two-parameter SV model and three-parameter SV model. Also, the proposed algorithm will be used

for dealing with S&P 500 data in Section 5.2. The simulation results and application on S&P 500 data will show that our proposed method is much more efficient than the original method.

4.2 PROPOSED METHOD WITH ADAPTIVE MARKOV CHAIN MONTE CARLO

For the Random Walk Metropolis Hastings, it is crucial to choose a proper proposal distribution for the convergence of the algorithm. However, sometimes a good proposal distribution is difficult to choose because both the size and the spatial orientation of the proposal distribution should be considered. The adaptive Markov Chain Monte Carlo algorithm can be used to overcome this difficulty. In this section, we are going to introduce the Adaptive Metropolis (AM) algorithm, which is an adaptive MCMC algorithm, and combine the AM method with our proposed method in Section 4.1.

4.2.1 Adaptive Markov Chain Monte Carlo

The Adaptive Metropolis algorithm is non-Markovian but can adapt continuously to the target distribution. Both the size and the spatial orientation of the proposal distribution will be adjusted by the adaptation procedure. Also, the AM algorithm is straightforward to implement and use in practice.

In the AM algorithm, we use all of the previous states to calculate the covariance of the proposal distribution. There is no extra computational cost by the AM algorithm because one may apply a simple recursion formula for the covariances involved. The AM algorithm starts using the cumulating information from the beginning of the sampling and it ensures that the search becomes more efficient at an early stage of the sampling. Also, Haario et al. (2001) proved that adaptive MCMC algorithms do indeed have the correct ergodicity properties.

In order to improve the RWMH algorithm in our proposed method, we now focus on how to modify the RWMH algorithm to the adaptive RWMH algorithm. Actually, we just need to add an additional updating step to the RWMH algorithm after each iteration. For the RWMH algorithm in Section 4.1.2, suppose that we have sampled the states $\xi^{[1]}, \xi^{[2]}, \cdots, \xi^{[r]}$ at iteration r. Then, we need to modify the proposal distribution of candidate $\eta^{[r+1]}$ in the Step 3 of Algorithm 6. It now depends on the whole history $(\xi^{[0]}, \xi^{[1]}, \cdots, \xi^{[r]})$ and $\eta^{[r+1]}$ is drawn from $P_{\theta_r}(\xi^{[r]}, \cdot)$, where $\theta_r = (\mu_r, \Sigma_r, \lambda_r)$ and $P_\theta = P_{N(0,\lambda\Sigma)}$ is the kernel of a symmetric RWMH with a Gaussian increment distribution $N(0, \lambda\Sigma)$. Note that λ is a positive constant scaling factor. The λ_r, μ_r and

Σ_r are calculated by the recursion formulas (Andrieu and Thoms, 2008),

$$\log(\lambda_{r+1}) = \log(\lambda_r) + \gamma_{r+1}[\alpha(\xi^{[r-1]}, \eta^{[r]}) - \alpha_\star], \tag{4.12}$$

$$\mu_{r+1} = \mu_r + \gamma_{r+1}(\xi^{[r+1]} - \mu_r), \tag{4.13}$$

$$\Sigma_{r+1} = \Sigma_r + \gamma_{r+1}((\xi^{[r+1]} - \mu_r)(\xi^{[r+1]} - \mu_r)^T - \Sigma_r), \tag{4.14}$$

where γ_r is a nonincreasing sequence of positive step lengths such that $\sum_{r=1}^{\infty} \gamma_r = \infty$ and $\sum_{r=1}^{\infty} \gamma_r^{1+\delta} < \infty$ for some $\delta > 0$; $\alpha(\xi^{[r-1]}, \eta^{[r]})$ is the acceptance probability in (4.7) and $\alpha_\star$ is the expected acceptance rate for the algorithm.

Algorithm 8 Adaptive Random Walk Metropolis Hastings Algorithm

1: Set the initial state $\xi^{[0]}$ arbitrarily and initialize μ_0, Σ_0 and λ_0.
2: **for** $r \geq 1$ **do**
3: Generate ϵ_r and let $\eta^{[r]} = \xi^{[r-1]} + \epsilon^{[r]}$, where $\epsilon^{[r]} \sim$ iid $N(0, \lambda_r \Sigma_r)$.
4: Generate $U_r \sim unif(0, 1)$.
5: **if** $U_r < \alpha(\xi^{[r-1]}, \eta^{[r]}) = min\Big(\frac{\pi(\eta^{[r]})}{\pi(\xi^{[r-1]})}, 1\Big)$ **then**
6: Set $\xi^{[r]} = \eta^{[r]}$.
7: **else**
8: Set $\xi^{[r]} = \xi^{[r-1]}$.
9: **end if**
10: Update λ_r, μ_r and Σ_r by

$$\log(\lambda_{r+1}) = \log(\lambda_r) + \gamma_{r+1}[\alpha(\xi^{[r]}, \eta^{[r]}) - \alpha_\star], \tag{4.15}$$

$$\mu_{r+1} = \mu_r + \gamma_{r+1}(\xi^{[r]} - \mu_r), \tag{4.16}$$

$$\Sigma_{r+1} = \Sigma_r + \gamma_{r+1}((\xi^{[r]} - \mu_r)(\xi^{[r]} - \mu_r)^T - \Sigma_r). \tag{4.17}$$

11: **end for**

These recursion formulas are called *learning* Σ_π *on the fly*, where Σ_π is the true covariance matrix of the target distribution. We can now summarize the adaptive RWMH in Algorithm 8. In addition, some other AM algorithms, which employ different recursion formulas or different update

strategies, were introduced in Andrieu and Thoms (2008), such as the Rao-Blackwellised AM algorithm, component-wise AM with component-wise adaptive scaling, global AM with component-wise adaptive scaling and so on.

4.2.2 Proposed Method with Adaptive Random Walk Metropolis Hastings

For the proposed algorithm in Section 4.1, one important step in practice is to find a good covariance matrix Σ, which can make the algorithm more efficient. However, the "hand tuning" for "optimal" covariance matrix requires some expertise and can be very time consuming. Hence, we introduce the adaptive step to the proposed algorithm to make it easy to use.

In our proposed algorithm (Algorithm 7), recall that Step 4 is to sample (ϕ, σ_ω) from the posterior distribution $f(\phi, \sigma_\omega \mid \mu, y_{1:T}, x_{1:T})$ by the RWMH algorithm. Now we are going to add an adaptive step after sampling out (ϕ, σ_ω). With the adaptive procedure, the algorithm can learn the true covariance matrix step by step and adapt to a good acceptance rate very quickly. Then, we revise our proposed algorithm and summarize the new algorithm with adaptive MCMC in Algorithm 9. Note that the recursion formulas (4.12)-(4.14) are used as the adaptive procedures in our algorithm. In addition, some other recursion formulas, which were mentioned in Section 4.2.1, could also be used as the adaptive procedure.

Algorithm 9 Proposed Method for Stochastic Volatility Model with Adaptive MCMC

1: Set the initial value of $\boldsymbol{\theta}^{[0]}$ and $x_{1:T}^{[0]}$ arbitrarily.
2: **for** $r \geq 1$ **do**
3: Draw $x_{1:T}^{[r]}$ by Conditional Particle Filter with Ancestor Sampling (Algorithm 4), conditioned on $x_{1:T}^{[r-1]}$ and $\boldsymbol{\theta}^{[r-1]} = [\phi, \sigma_\omega, \mu]^{[r-1]}$.
4: With $x_{1:T}^{[r]}$, generate the pair $\boldsymbol{\xi}^{[r]} = [\phi, \sigma_\omega]^{[r]}$ by adaptive Random Walk Metropolis Hastings (Algorithm 8) with acceptance probability $\alpha = \frac{g(\boldsymbol{\xi})}{g(\boldsymbol{\xi}^{[r]})} \wedge 1$, where $g(\boldsymbol{\xi})$ is defined in (4.10).
5: With $x_{1:T}^{[r]}$ and $\boldsymbol{\xi}^{[r]}$, draw $\mu^{[r]}$ from $N(\mu_1, \sigma_\mu^2)$ in (4.11).
6: **end for**

In the Section 5.1, we will show the simulation results after applying the adaptive procedure for the two-parameter SV model and three-parameter SV model. Our proposed algorithm with adaptive MCMC procedure will then be applied to S&P 500 data in Section 5.2. Both simulation and case studies will confirm that our new adaptive algorithm is still efficient for the SV models.

4.3 PROPOSED METHOD FOR MULTIVARIATE STOCHASTIC VOLATILITY MODEL

In this section, we will apply our proposed method to the Multivariate Stochastic Volatility (MSV) model in Section 3.1.2. The unknown parameters in the MSV model (3.10) and (3.11) are $\{\phi, \sigma_\omega, \boldsymbol{\beta}\}$. If we assume (ϕ, σ_ω) has bivariate normal prior distribution in (4.3), we can directly apply our proposed sampling method in Section 4.1.3 to the MSV model. But we still need to derive the sampling method for β_is from their posterior distributions.

The full likelihood function of $\boldsymbol{y}_{1:T} \mid x_{1:T}, \boldsymbol{\beta}$ is

$$\begin{aligned} L(\boldsymbol{y}_{1:T} \mid x_{1:T}, \boldsymbol{\beta}) &= \prod_{t=1}^{T} \frac{1}{\sqrt{(2\pi)^p |\boldsymbol{\Sigma}|}} \exp\left\{-\frac{1}{2}\boldsymbol{y}_t^T \boldsymbol{\Sigma}^{-1} \boldsymbol{y}_t\right\} \\ &= \frac{1}{\sqrt{(2\pi)^{pT}\left[\prod_{i=1}^{p}\beta_i^2\right]^T \prod_{t=1}^{T}\exp\left(x_t\right)}} \exp\left\{-\frac{1}{2}\sum_{i=1}^{p}\sum_{t=1}^{T}\frac{y_{it}^2}{\beta_i^2 \exp(x_t)}\right\}. \end{aligned} \tag{4.18}$$

Then, we will try non-informative uniform distribution, normal distribution and Inverse Gamma distribution as the prior of $\boldsymbol{\beta}$.

Uniform Priors:

Notice that $\{\beta_i^2\}_{i=1}^p$ in (4.18) can be any value larger than 0. We can assume that the prior distributions of $\{\beta_i^2\}_{i=1}^p$ are i.i.d uniform distribution $U(0, b)$. The posterior distribution is

$$\begin{aligned} \pi(\beta_i^2 \mid \boldsymbol{y}_{1:T}, x_{1:T}) &\propto L(\boldsymbol{y}_{1:T} \mid x_{1:T}, \boldsymbol{\beta})\pi(\beta_i^2) \\ &\propto \left(\frac{1}{\beta_i^2}\right)^{\frac{T}{2}} \exp\left\{-\frac{1}{2}\sum_{t=1}^{T}\frac{y_{it}^2}{\exp(x_t)}\frac{1}{\beta_i^2}\right\} \cdot \frac{1}{b} \\ &\sim \mathcal{IG}\left(\frac{T}{2} - 1, \sum_{t=1}^{T}\frac{y_{it}^2}{2\exp(x_t)}\right). \end{aligned} \tag{4.19}$$

Using the Mean Square Error (MSE) as risk, the Bayes estimator of β_i^2 is

$$\hat{\beta}_i^2 = E(\beta_i^2 \mid \boldsymbol{y}_{1:T}, x_{1:T}) = \frac{\sum_{t=1}^{T}\frac{y_{it}^2}{\exp(x_t)}}{T-4}. \tag{4.20}$$

Normal Priors:

Notice that the scale parameter $\beta_i \in \mathbb{R}$. We assume that the diffuse (non-informative) prior distributions of $\{\beta_i\}_{i=1}^p$ are i.i.d normal distribution $N(0, b^2 = \infty)$. Then, the posterior distribution is

$$\begin{aligned}
\pi(\beta_i^2 \mid \boldsymbol{y}_{1:T}, x_{1:T}) &\propto L(\boldsymbol{y}_{1:T} \mid x_{1:T}, \boldsymbol{\beta})\pi(\beta_i) \\
&\propto \left(\frac{1}{\beta_i^2}\right)^{\frac{T}{2}} \exp\left\{-\frac{1}{2}\sum_{t=1}^{T}\frac{y_{it}^2}{\exp(x_t)}\frac{1}{\beta_i^2}\right\} \cdot \exp\left\{-\frac{\beta_i^2}{2b^2}\right\} \qquad (\text{as } b^2 \to \infty) \\
&\propto \left(\frac{1}{\beta_i^2}\right)^{\frac{T}{2}} \exp\left\{-\frac{1}{2}\sum_{t=1}^{T}\frac{y_{it}^2}{\exp(x_t)}\frac{1}{\beta_i^2}\right\} \\
&\sim \mathcal{IG}\left(\frac{T}{2} - 1,\ \sum_{t=1}^{T}\frac{y_{it}^2}{2\exp(x_t)}\right). \qquad (4.21)
\end{aligned}$$

Note that the posterior distributions above are exactly the same as the posterior distributions (4.19) of uniform priors. Thus, the Bayes estimator under MSE risk is (4.20).

Inverse Gamma Priors:

Since the likelihood function of $\boldsymbol{y}_{1:T} \mid x_{1:T}, \boldsymbol{\beta}$ in (4.18) is Normal distribution with unknown parameter $\boldsymbol{\beta}$, we can use its conjugate prior distribution. Assuming the prior distributions of$\{\beta_i^2\}_{i=1}^p$ are i.i.d Inverse-Gamma distributions $\mathcal{IG}(a, b)$ with mean $\frac{b}{a-1}$ and variance $\frac{b^2}{(a-1)^2(a-2)}$, the posterior distribution is

$$\begin{aligned}
\pi(\beta_i^2 \mid \boldsymbol{y}_{1:T}, x_{1:T}) &\propto f(\boldsymbol{y}_{1:T} \mid x_{1:T}, \boldsymbol{\beta})\xi(\beta_i^2) \\
&\propto \left(\frac{1}{\beta_i^2}\right)^{\frac{T}{2}} \exp\left\{-\frac{1}{2}\sum_{t=1}^{T}\frac{y_{it}^2}{\exp(x_t)}\frac{1}{\beta_i^2}\right\} \cdot \left(\frac{1}{\beta_i^2}\right)^{a+1} \exp\left\{-\frac{b}{\beta_i^2}\right\} \\
&\propto \left(\frac{1}{\beta_i^2}\right)^{\frac{T}{2}+a+1} \exp\left\{-\left[\frac{1}{2}\sum_{t=1}^{T}\frac{y_{it}^2}{\exp(x_t)} + b\right]\frac{1}{\beta_i^2}\right\} \\
&\sim \mathcal{IG}\left(\frac{T}{2} + a,\ \sum_{t=1}^{T}\frac{y_{it}^2}{2\exp(x_t)} + b\right). \qquad (4.22)
\end{aligned}$$

Thus, under MSE risk, the Bayes estimator of β_i^2 is

$$\hat{\beta}_i^2 = E(\beta_i^2 \mid \boldsymbol{y}_{1:T}, x_{1:T}) = \frac{\sum_{t=1}^{T}\frac{y_{it}^2}{\exp(x_t)} + 2b}{T + 2a - 2}. \qquad (4.23)$$

After we derive the posterior distributions of β for three different priors above, our algorithm for MSV model can be summarized in Algorithm 10. Similar with our proposed algorithm for univariate SV models, this algorithm can be accommodated to two-parameter MSV model by skipping sampling μ in Step 5.

In Section 5.3, a case study will be shown to illuminate our proposed algorithm for MSV model. We will apply our proposed method to multiple bank stocks data. From the case study, it is easy to find that our proposed method works well for the MSV model.

Algorithm 10 Proposed Method for Multivariate Stochastic Volatility Model

1: Set the initial value of $\boldsymbol{\theta}^{[0]}$ and $x_{1:T}^{[0]}$ arbitrarily.
2: **for** $r \geq 1$ **do**
3: Draw $x_{1:T}^{[r]}$ by Conditional Particle Filter with Ancestor Sampling (Algorithm 4), conditioned on $x_{1:T}^{[r-1]}$ and $\boldsymbol{\theta}^{[r-1]} = [\phi, \sigma_\omega, \mu]^{[r-1]}$.
4: With $x_{1:T}^{[r]}$, generate the pair $[\phi, \sigma_\omega]^{[r]}$ by Random Walk Metropolis Hastings with acceptance probability α in (4.10).
5: With $x_{1:T}^{[r]}$ and $[\phi, \sigma_\omega]^{[r]}$, draw $\mu^{[r]}$ from $N(\mu_1, \sigma_\mu^2)$ in (4.11).
6: With $x_{1:T}^{[r]}$, draw $\beta_i^{2[r]}$ from posterior distribution in (4.19), (4.21) or (4.22) .
7: **end for**

5.0 SIMULATION STUDIES AND APPLICATIONS

In this chapter, we first perform simulation studies for proposed methods in Section 5.1 and apply our proposed methods to S&P 500 data in Section 5.2. In Section 5.3, the proposed method will be applied to multiple stocks data.

In order to make the simulation studies and applications for univariate SV model easy to understood, denote the approaches to solve the univariate SV model as Method A, Method B and Method C.

Method A: PGAS with individual parameter sampling and employing priors in Section 3.1.3. That is, we use the Algorithm 5 with posteriors in (3.16), (3.21) and (3.24).

Method B: PGAS with joint parameter sampling using non-adaptive MCMC (Algorithm 7).

Method C: PGAS with joint parameter sampling using adaptive MCMC (Algorithm 9).

Note that Method A is the standard (existing) method, which samples the parameters individually by drawing from the univariate posteriors $p(\phi \mid \sigma_\omega, x_{1:T}, y_{1:T})$ and $p(\sigma_\omega \mid \phi, x_{1:T}, y_{1:T})$. Method B and Method C are our proposed methods, which sample the parameters jointly from the posteriors $p(\phi, \sigma_\omega \mid x_{1:T}, y_{1:T})$.

This chapter is organized as follows. In Section 5.1.1, we first perform simulation studies for the two-parameter SV model and three-parameter SV model using Method B. For the two-parameter SV model, we also make a comparison between Method A and Method B to illustrate the advantage of our proposed methods. The Section 5.1.2 shows our simulation results for Method C. Then, in Section 5.2.1, we apply Method B to S&P 500 data and make comparison with Method A. Section 5.2.2 gives a case study for S&P 500 data using Method C. In the end, we use our proposed method to solve the MSV model for several stocks data in Section 5.3.

5.1 SIMULATION STUDIES FOR PROPOSED METHODS

5.1.1 Proposed Method with non-Adaptive MCMC (Method B) for Simulated Data

In this section, we apply our proposed method (Method B) in Section 4.1 to the two-parameter SV model in (3.26)-(3.27) and the three-parameter SV model in (3.7)-(3.8) through simulation studies. For the two-parameter SV model, we also compare the inefficiencies of Method A with the inefficiencies of Method B.

Two-parameter SV model:

For the two-parameter SV model, we set the true values of (ϕ, σ_ω^2) as $(0.85, 0.2)$. To avoid start-up problems, we generate $T = 2000$ time points series by the model (3.26)-(3.27) and discard the first 1000 time points. The generated simulation series is shown at the bottom of Figure 5.2.

Then, we apply Method B to the simulated data. The prior distribution for (ϕ, σ_ω) is

$$\begin{pmatrix} \phi \\ \sigma_\omega \end{pmatrix} \sim N\left(\begin{bmatrix} \mu_\phi \\ \mu_{\sigma_\omega} \end{bmatrix}, \begin{bmatrix} \sigma_\phi^2 & \rho\sigma_\phi\sigma_q \\ \rho\sigma_\phi\sigma_q & \sigma_q^2 \end{bmatrix} \right), \tag{5.1}$$

where the hyperparameters $(\mu_\phi, \mu_{\sigma_\omega}) = (0.9, 0.27)$ and $(\sigma_\phi, \sigma_q) = (0.05, 0.1)$. In order to represent the negative correlation between ϕ and σ_ω, we set the correlation coefficient ρ as -0.25 in our prior distribution.

The number of particles is $N = 20$. Choosing the initial value $(\phi^{[0]}, \sigma_\omega^{2\,[0]}) = (0.95, 0.15)$, we run the MCMC for 33,000 iterations and the first 3,000 samples are discarded as burn-in. In order to achieve a great acceptance rate in RWMH Step 4 in our proposed algorithm, we choose $\lambda = 1.7 \times 10^{-4}$ and $\mathbf{\Sigma}_\epsilon = \begin{bmatrix} 100 & -25 \\ -25 & 100 \end{bmatrix}$ for the Step (i) in RWMH. The acceptance rate is about 25.27% in this simulation study.

In order to examine the efficiency for our proposed method, we plot the autocorrelations (ACFs) in Figure 5.1. We could see that the autocorrelations plots for both parameters, ϕ and σ_ω, decrease extremely fast, which means the joint parameter sampling approach (Method B) figures out the slow convergence problem in the existing approach (Method A). In Figure 5.2, the

estimated mean of hidden states have relative small credible intervals, and the hidden states x_t increase when the simulated series y_t suffers a large volatility. The contour plot in Figure 5.3 presents a negative correlation between ϕ and σ_ω^2, which is similar to the contour plot in Figure 3.3.

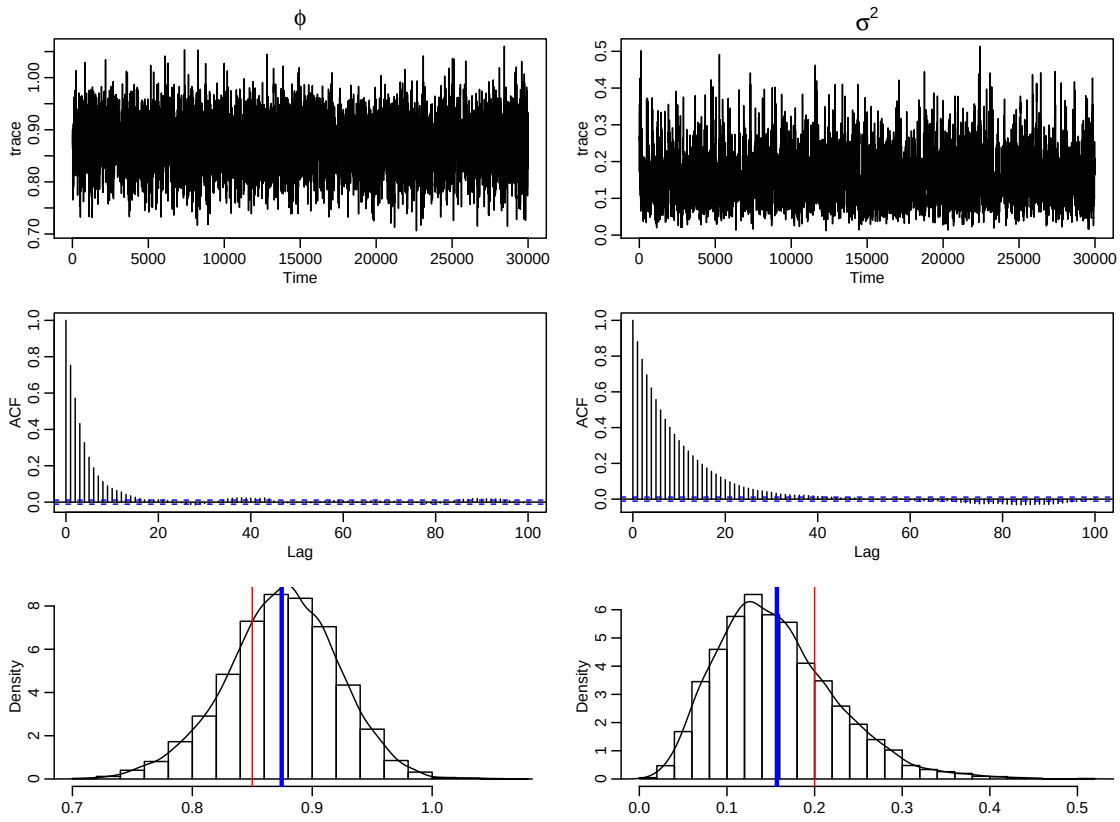

Figure 5.1: (Method B; Simulated data; Two-parameter SV model) *Results for simulated data using the two-parameter SV model. TOP: Trace plots of sampled parameters. MIDDLE: the sample ACFs of the traces. BOTTOM: Histograms of sampled parameters. The blue line and red line show the sample mean and true value of parameter, respectively. In total 33,000 iterations were drawn and discarded the first 3,000 as burn-in. The particles number is* $N = 20$.

In addition, we apply the proposed method using different number of particles for the simulated data. The number of MCMC iterations is still 33,000 and we discard the first 3,000 samples as burn-in. Employing the same bivariate normal prior distribution as the last simulation study for Method B, the inefficiencies for existing individual sampling method (Method A) and our proposed joint sampling method (Method B) are summarized in Table 5.1. A useful interpretation of

the inefficiency is that $n \times$ IF samples from the Markov chain are needed to achieve the same precision as using n i.i.d. samples from the posterior distribution. From the table, we can conclude that our new joint sampling method reduces the inefficiencies significantly because Method B gives a lower average inefficiencies than Method A. It is clear that Method B has better mixing properties than Method A. We can see that Method B gives the lowest average ineffciencies at $N = 500$ while Method A gives it at $N = 200$, and Method B is more statble than Method A for different particle numbers. Note that the inefficiencies in the table didn't take the computational cost of the algorithm into account. Since both methods are insensitive to the number of particles N, which means they are robust to small number of particles ($N = 5 - 20$), using less particles will save more time on running MCMC. So, we can get a great result with a low computational cost, which is an advantage of the PGAS algorithm.

Table 5.1: *Inefficiencies for two-parameter SV model for simulated data. The three columns to the left are inefficiencies for individual sampling (Method A). The rightmost three columns are inefficiencies for our proposed joint sampling method (Method B). The third column and the last column are the average inefficiencies for the two parameters for Method A and Method B, respectively.*

	Method A (Individual Sampling)			Method B (Joint Sampling)		
	ϕ	σ_ω^2	Average	ϕ	σ_ω^2	Average
N=10	33.484	61.244	47.364	7.526	17.159	12.342
N=20	31.986	56.802	44.394	7.514	17.820	12.667
N=50	45.742	77.387	61.564	7.463	17.180	12.322
N=100	32.154	57.931	45.043	7.615	17.510	12.562
N=200	30.963	52.564	**41.763**	7.012	17.008	12.010
N=500	40.881	114.791	77.836	7.363	15.298	**11.330**

In Table 5.2 and Table 5.3, we give the summary statistics for the parameters ϕ and σ_ω^2 of simulation studies, respectively, which tried different number of particles. The mean and standard deviation of each simulation studies are given at the first two columns in the tables. The naive SE is the standard error of the mean (adjusting for sample size). The rightmost two columns give the 95% credible intervals (C.I.). We can easily find that all simulation studies give similar 95%

credible intervals. The narrowest 95% Credible Intervals for ϕ and σ_ω^2 are given by $N = 200$ and $N = 10$, respectively. For different number of particles, the graphs of simulation results of Method B are provided in Appendix A.1.

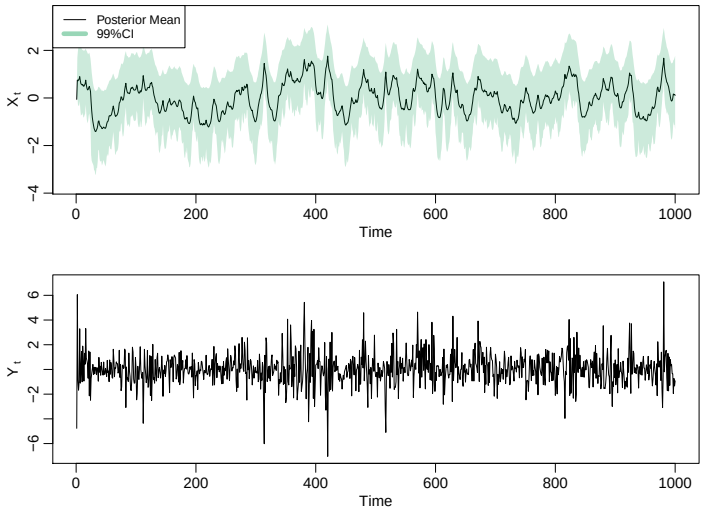

Figure 5.2: (Method B; Simulated data; Two-parameter SV model) *TOP: State process estimated posterior mean plot with 99% credible intervals. BOTTOM: The simulated data plot.*

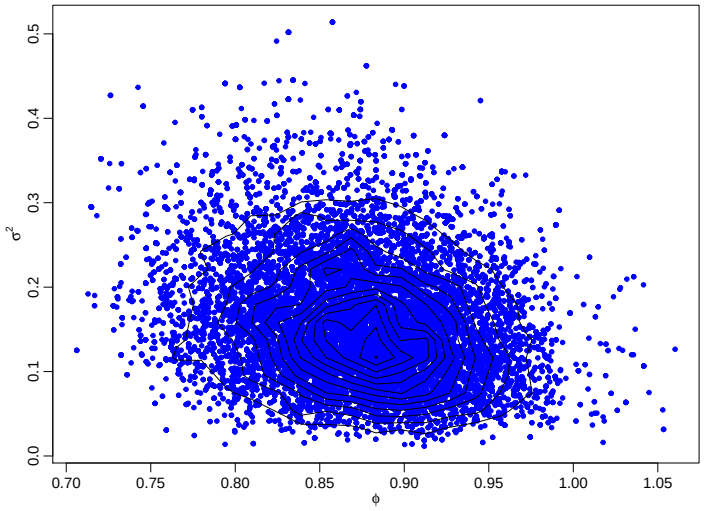

Figure 5.3: (Method B; Simulated data; Two-parameter SV model) *The contour plot of sampled parameters σ_ω^2 v.s. ϕ.*

Table 5.2: (Method B; Simulated data; Two-parameter SV model) *Summary statistics of sampled parameter ϕ.*

	Mean	SD	Naive SE	2.50%	97.50%
N=10	0.876	0.04593	0.00027	0.782	0.961
N=20	0.875	0.04620	0.00027	0.778	0.960
N=50	0.871	0.04585	0.00026	0.778	0.958
N=100	0.872	0.04659	0.00027	0.777	0.961
N=200	0.871	**0.04560**	**0.00026**	**0.777**	**0.957**
N=500	0.871	0.04602	0.00027	0.782	0.960

Table 5.3: (Method B; Simulated data; Two-parameter SV model) *Summary statistics of sampled parameter σ_ω^2.*

	Mean	SD	Naive SE	2.50%	97.50%
N=10	0.148	**0.06330**	**0.00037**	**0.047**	**0.293**
N=20	0.157	0.06794	0.00039	0.051	0.312
N=50	0.163	0.06861	0.00040	0.058	0.322
N=100	0.172	0.07068	0.00041	0.060	0.330
N=200	0.177	0.07029	0.00041	0.068	0.343
N=500	0.183	0.07054	0.00041	0.067	0.342

Three-parameter SV model:

Next, we run simulation for the three-parameter SV model in (3.7)-(3.8). Setting the true values of $(\phi, \sigma_\omega^2, \mu)$ as $(0.85, 0.2, 0.1)$, we generate $T = 1000$ time points series. The generated simulation series is shown at the bottom of Figure 5.5.

Then, we apply Method B to the simulated data with the bivariate normal prior distribution in Equation (4.3). The hyperparameters were set as $(\mu_\phi, \mu_{\sigma_\omega}) = (0.875, 0.45)$ and $(\sigma_\phi, \sigma_q) = (0.075, 0.1)$. We set the correlation coefficient $\rho = -0.25$ in bivariate normal prior distribution.

In this simulation, the number of particles is $N = 20$. Setting the initial value $(\phi^{[0]}, \sigma_\omega^{2\,[0]}, \mu^{[0]}) = (0.95, 0.25, 0.15)$, we run the MCMC sampler for 33,000 iterations and the first 3,000 samples are discarded as burn-in. Note that we use the same λ and $\mathbf{\Sigma}_\epsilon$ as we used before in the simulation for the two-parameter SV model. The acceptance rate is about 37.8%.

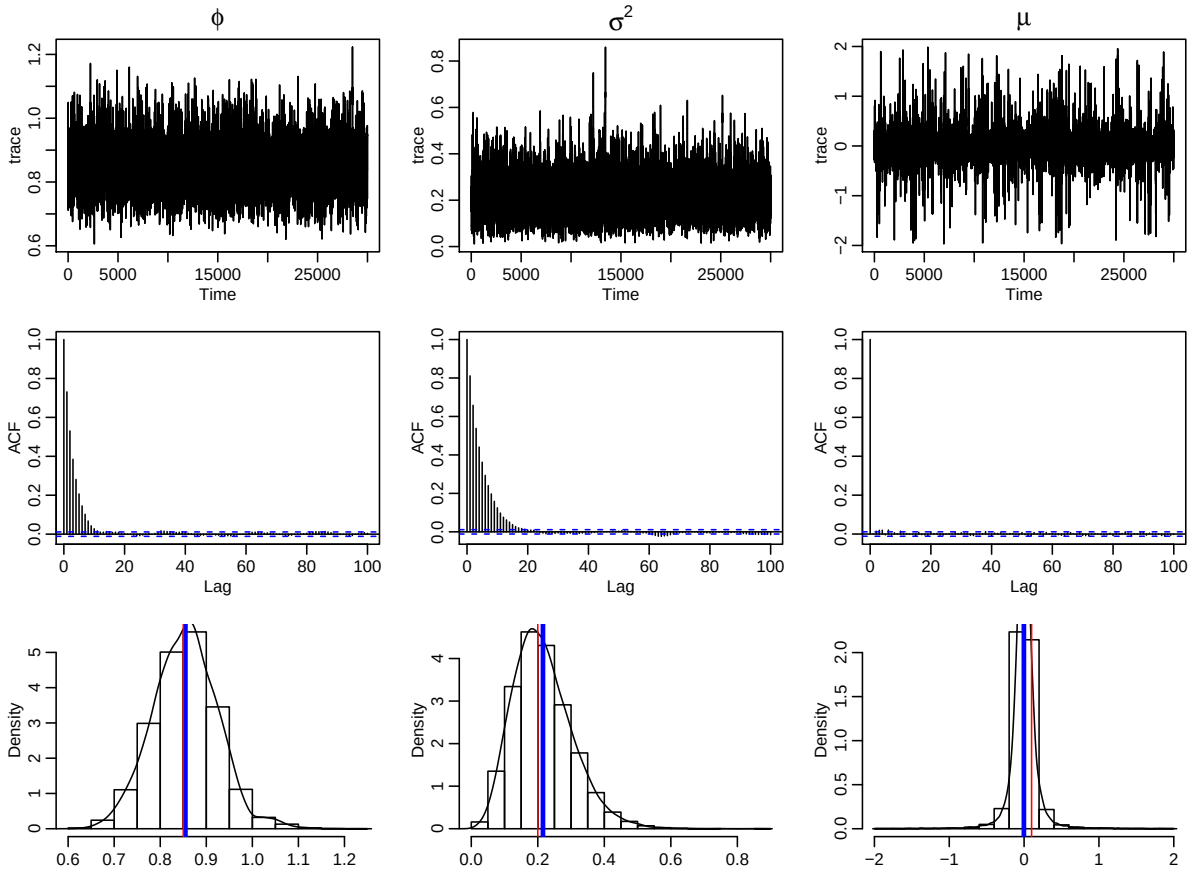

Figure 5.4: (Method B; Simulated data; Three-parameter SV model) *Results for simulated data using the three-parameter SV model. TOP: Trace plots of sampled parameters. MIDDLE: the sample ACFs of the traces. BOTTOM: Histograms of sampled parameters. The blue line and red line show the sample mean and true value of parameter, respectively. In total 33,000 iterations were drawn and discarded the first 3,000 as burn-in. The particles number is* $N = 20$.

From Figure 5.4, we can get results similar to those of the two-parameter SV model. The autocorrelations of all three parameters fall sharply. The slow convergence problem has been solved and it shows Method B also works well on the three-parameter SV model. The Figure 5.5 and Fig-

ure 5.6 give the hidden states estimated mean and contour plot for ϕ and σ_ω^2, which demonstrates a negative correlation between ϕ and σ_ω^2.

Table 5.4: (Method B) *Inefficiencies for three-parameter SV model for simulated data. The three columns to the left are inefficiencies for ϕ, σ_ω^2 and μ, respectively. The last column is the average inefficiencies for the three parameters.*

	ϕ	σ_ω^2	μ	Average
N=10	6.661	9.783	0.982	5.809
N=20	6.231	9.513	1.187	**5.643**
N=50	6.812	12.526	1.035	6.791
N=100	6.829	10.886	1.156	6.290
N=200	6.434	10.065	0.973	5.824
N=500	7.222	9.988	1.096	6.102

Table 5.5: (Method B; Simulated data; Three-parameter SV model) *Summary statistics of sampled paremater ϕ.*

	Mean	SD	Naive SE	2.50%	97.50%
N=10	0.855	**0.07071**	**0.00041**	**0.717**	**0.987**
N=20	0.855	0.07080	0.00041	0.716	0.995
N=50	0.855	0.07329	0.00042	0.712	1.010
N=100	0.855	0.07173	0.00041	0.717	1.005
N=200	0.854	0.07107	0.00041	0.715	0.998
N=500	0.853	0.07178	0.00041	0.716	0.989

Similar to the simulation studies for the two-parameter SV model, we also apply the proposed method (Method B) using different numbers of particles. We summarize the inefficiencies in Table 5.4 and we still observe very low inefficiencies for all parameters. This implies Method B is still very efficient for the three-parameter SV model. Comparing the inefficiencies in Table 5.4 and Table 5.1, we get similar inefficiencies for ϕ. But for σ_ω^2, the three-parameter SV model gives

better results. Note that the average inefficiencies for the three parameters achive the lowest value when $N = 20$.

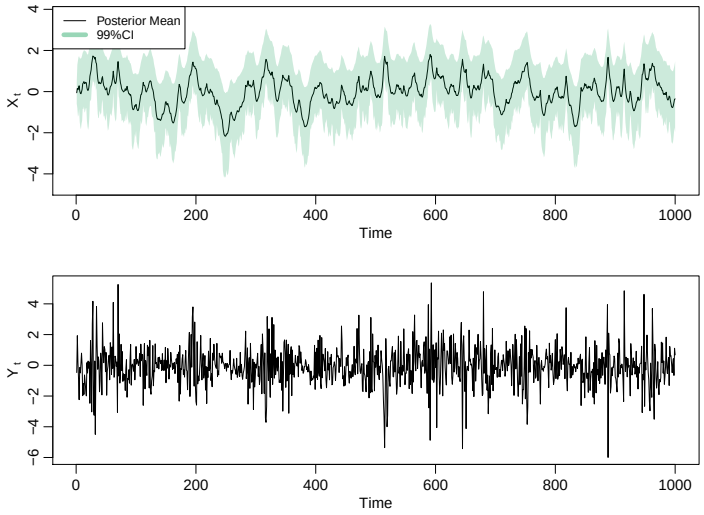

Figure 5.5: (Method B; Simulated data; Three-parameter SV model) *TOP: State process estimated posterior mean plot with 99% credible intervals. BOTTOM: The simulated data plot.*

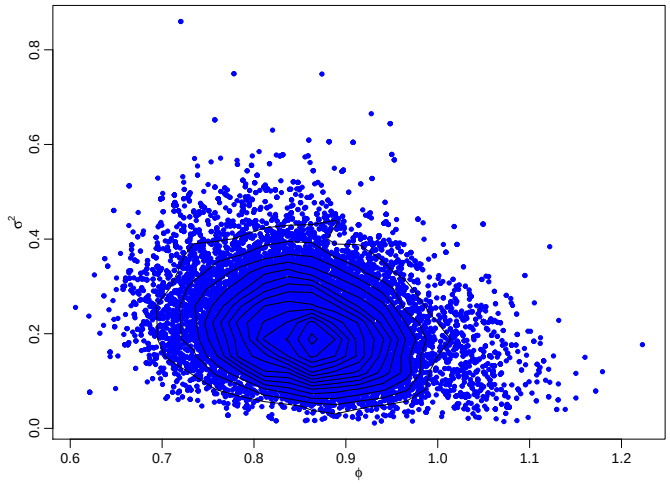

Figure 5.6: (Method B; Simulated data; Three-parameter SV model) *The contour plot of sampled parameters σ_ω^2 v.s. ϕ.*

Similar to the simulation studies for the two-parameter SV model, six different number of

particles are used for the three-parameter model. We give the summary statistics for the sampled parameters ϕ, σ_ω^2 and μ of simulation studies, respectively, in Table 5.5, Table 5.6 and Table 5.7. All simulation studies give similar sample means, standard deviations and 95% credible intervals. The narrowest 95% CI for ϕ, σ_ω^2 and μ are given by $N = 10$, $N = 20$ and $N = 10$, respectively. For different number of particles, the graphs of simulation results of our proposed method (Method B) are provided in Appendix A.2.

Table 5.6: (Method B; Simulated data; Three-parameter SV model) *Summary statistics of sampled parameter* σ_ω^2.

	Mean	SD	Naive SE	2.50%	97.50%
N=10	0.218	0.09182	0.00053	0.072	0.427
N=20	0.216	**0.08885**	**0.00051**	**0.072**	**0.416**
N=50	0.221	0.09339	0.00054	0.071	0.430
N=100	0.218	0.09179	0.00053	0.071	0.427
N=200	0.220	0.09140	0.00053	0.072	0.426
N=500	0.222	0.09169	0.00053	0.072	0.424

Table 5.7: (Method B; Simulated data; Three-parameter SV model) *Summary statistics of sampled parameter* μ.

	Mean	SD	Naive SE	2.50%	97.50%
N=10	-0.004	**0.18209**	**0.00105**	**-0.337**	**0.317**
N=20	-0.003	0.18757	0.00108	-0.331	0.323
N=50	-0.002	0.18917	0.00109	-0.336	0.325
N=100	-0.004	0.18386	0.00106	-0.323	0.303
N=200	-0.003	0.18542	0.00107	-0.323	0.320
N=500	-0.005	0.18559	0.00107	-0.340	0.311

5.1.2 Proposed Method with Adaptive MCMC (Method C) for Simulated Data

In this section, we apply our proposed method with the adaptive MCMC in Section 4.2 (Method C) to the three-parameter SV model through simulation studies.

For the three-parameter SV model, we set the true values of $(\phi, \sigma_\omega^2, \mu)$ as $(0.85, 0.2, 0.1)$ and generate $T = 1000$ time points series by the model (3.26)-(3.27) as our simulated data. The generated simulation series is shown at the bottom of Figure 5.7.

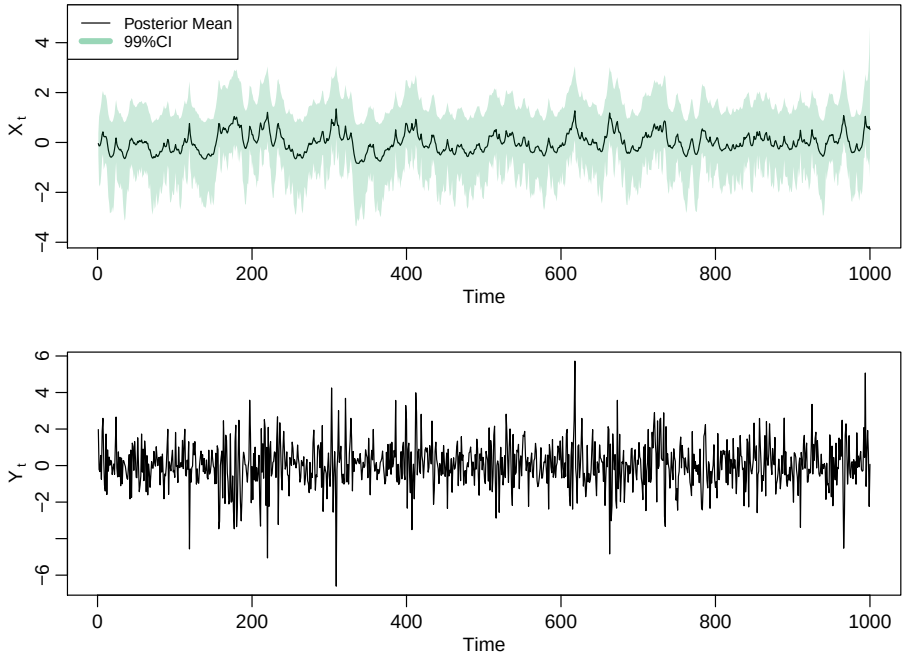

Figure 5.7: (Method C; Simulated data; Three-parameter SV model) *TOP: State process estimated posterior mean plot with 99% credible intervals. BOTTOM: The simulated data plot.*

Then, we apply Method C to the simulated data. We still set the correlation coefficient $\rho = -0.25$ in our bivariate normal prior distribution. The other hyperparameters are set as $(\mu_\phi, \mu_{\sigma_\omega}) = (0.9, 0.25)$ and $(\sigma_\phi, \sigma_q) = (0.25, 0.25)$. The number of particles is $N = 20$. To the adaptive random walk Metropolis Hastings, we choose initial value of $\mathbf{\Sigma}_0 = 0.026 \times \begin{bmatrix} 1 & -0.25 \\ -0.25 & 1 \end{bmatrix}$ and λ_0 is 2.8. The γ_r in recursion formulas (4.12) - (4.14) is set as $\gamma_r = \frac{500}{r}$ and $\alpha_\star = 0.20$.

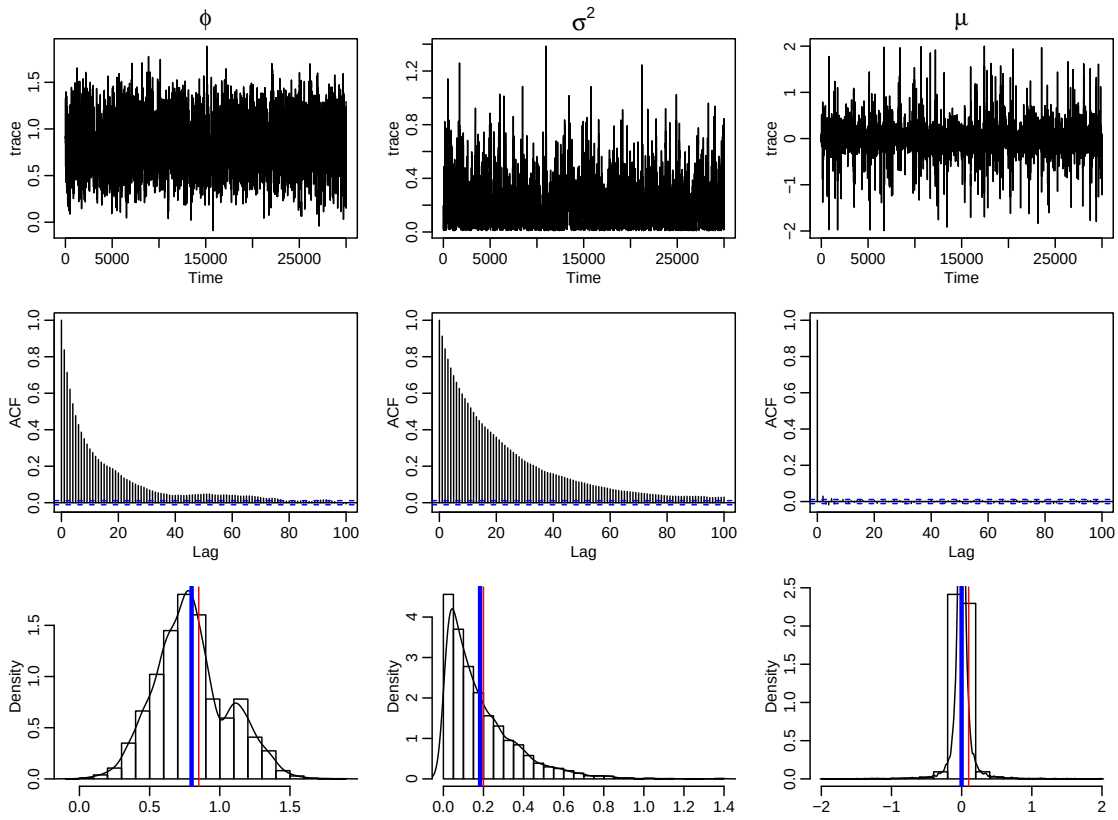

Figure 5.8: (Method C; Simulated data; Three-parameter SV model) *Results for simulated data using the three-parameter SV model. TOP: Trace plots of sampled parameters. MIDDLE: the sample ACFs of the traces. BOTTOM: Histograms of sampled parameters. The blue line and red line show the sample mean and true value of parameter, respectively. In total 33,000 iterations were drawn and discarded the first 3,000 as burn-in. The particles number is* $N = 20$.

Choosing the initial value $(\phi^{[0]}, {\sigma_\omega^2}^{[0]}, \mu^{[0]}) = (0.95, 0.25, 0.15)$, we run the MCMC for 33,000 iterations and the first 3,000 samples are discarded as burn-in. The acceptance rate of adaptive RWMH is 20.81%, which is very close to the target acceptance probability $\alpha_\star$. The autocorrelations were plotted in Figure 5.8 to evaluate the efficiency for Method C. The ACFs for all parameters, ϕ, σ_ω and μ, plummet down to the bottom. The estimated inefficiencies of ϕ, σ_ω and μ are 21.475, 42.265 and 1.030, respectively. The average ineffiencies for the three parameters is 21.590. Althrough Method C gives larger inefficiences comparing to the inefficiences of Method B in Table 5.4, the inefficiencies of ϕ and σ_ω are still smaller than those of Method A in Table

5.1. The adaptive method (Method C) can still solve the slow convergence problem although it might not be as good as the non-adaptive method (Method B). But employing Method C will save much time on finding a "good" proposal distribution in Method B. In Figure 5.7, the estimated hidden states have relatively small credible intervals. The contour plot in Figure 5.9 also displays a negative correlation between ϕ and σ_ω^2.

In Table 5.8, we give the sample mean, 95% credible interval and some other summary statistics for the sampled parameters ϕ, σ_ω^2 and μ of simulation studies, respectively.

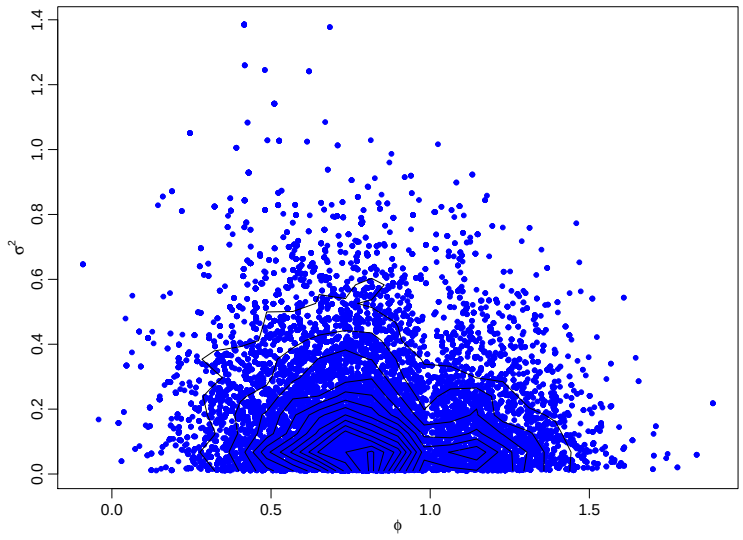

Figure 5.9: (Method C; Simulated data; Three-parameter SV model) *The contour plot of sampled parameters σ_ω^2 v.s. ϕ.*

Table 5.8: (Method C; Simulated data; Three-parameter SV model) *Parameter estimation and summary statistics for sampled parameters. The average inefficiency for the three sampled parameters is 21.590.*

	Mean	SD	Naive SE	2.50%	97.50%	Inefficiency
ϕ	0.799	0.26126	0.00151	0.335	1.347	21.475
σ_ω^2	0.183	0.17148	0.00099	0.013	0.634	42.265
μ	-0.001	0.14594	0.00084	-0.221	0.221	1.030

5.2 APPLICATION ON S&P 500 DATA

5.2.1 Proposed Method with non-Adaptive MCMC (Method B) for S&P 500 Data

In this section, we apply Method B to the S&P 500 data in Section 3.2. Similar with the simulation studies in Section 5.1, we run the MCMC for the two-parameter SV model first and the three-parameter SV model later and evaluate the performance of our proposed method.

Two-parameter SV model:

For the two-parameter SV model, we employ bivariate normal distribution as prior distribution for (ϕ, σ_ω) and setting the hyperparameters as $(\mu_\phi, \mu_{\sigma_\omega}) = (0.9, 0.5)$ and $(\sigma_\phi, \sigma_q) = (0.075, 0.1)$. To represent the negative correlation between ϕ and σ_ω, we set the hyperparameter ρ as -0.25. Similar to the setting in the simulation studies, we set the number of particles as $N = 20$ and the initial value as $(\phi^{[0]}, \sigma_\omega^{2\,[0]}) = (0.95, 0.2)$. After discarded the first 3,000 samples as burn-in, we kept 30,000 samples by Method B. Choosing $\lambda = 1.7 \times 10^{-4}$ and $\mathbf{\Sigma}_\epsilon = \begin{bmatrix} 100 & -25 \\ -25 & 100 \end{bmatrix}$, the acceptance rate is about 39.77%.

Table 5.9: *Inefficiencies for two-parameter SV model for S&P 500 data. The three columns to the left are inefficiencies for individual sampling (Method A). The rightmost three columns are inefficiencies for our proposed joint sampling method (Method B). The third column and the last column are the average inefficiencies for the two parameters for Method A and Method B, respectively.*

	Method A (Individual Sampling)			Method B (Joint Sampling)		
	ϕ	σ_ω^2	Average	ϕ	σ_ω^2	Average
N=10	27.848	100.028	63.938	7.656	10.048	8.852
N=20	32.918	131.958	82.438	6.857	9.367	**8.112**
N=50	26.595	106.743	66.669	7.123	10.174	8.648
N=100	29.606	117.346	73.476	7.943	10.542	9.242
N=200	18.982	101.597	60.289	7.148	9.818	8.483
N=500	22.622	84.119	**53.371**	7.369	9.624	8.496

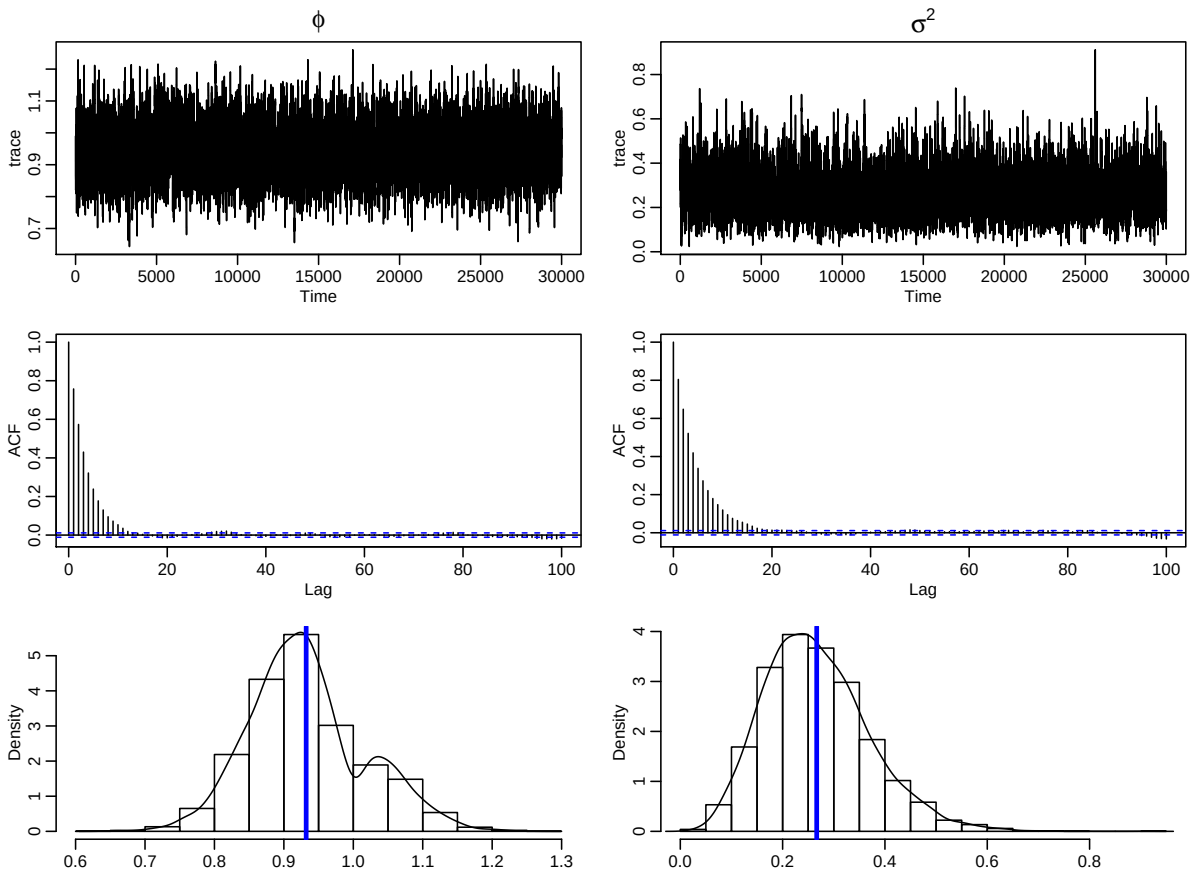

Figure 5.10: (Method B; S&P 500 data; Two-parameter SV model) *Results for S&P 500 data using the two-parameter SV model. TOP: Trace plots of sampled parameters. MIDDLE: the sample ACFs of the traces. BOTTOM: Histograms of sampled parameters. The blue lines show the sample mean of parameters. In total 33,000 iterations were drawn and discarded the first 3,000 as burn-in. The particles number is* $N = 20$.

The MCMC results are given in Figure 5.10, 5.11 and 5.12. Comparing to the Figure 3.2 for Method A, the autocorrelations for both parameters ϕ and σ_ω^2 in Figure 5.10 drop to the bottom before lag 20. It indicates that our proposed joint sampling procedure (Method B) is super efficient to deal with the two-parameter SV mdoel in practice. In Figure 5.11, the trend of posterior mean of hidden states x_t exhibits volatility trend very well. During the financial crisis period in 2008, the hidden states x_t raise significantly because of the large volatility. Note that the estimated credible intervals for the hidden states are all small. The contour plot reflects a negative correlation between

ϕ and σ_ω^2, which is similar to the contour plot in Figure 3.3.

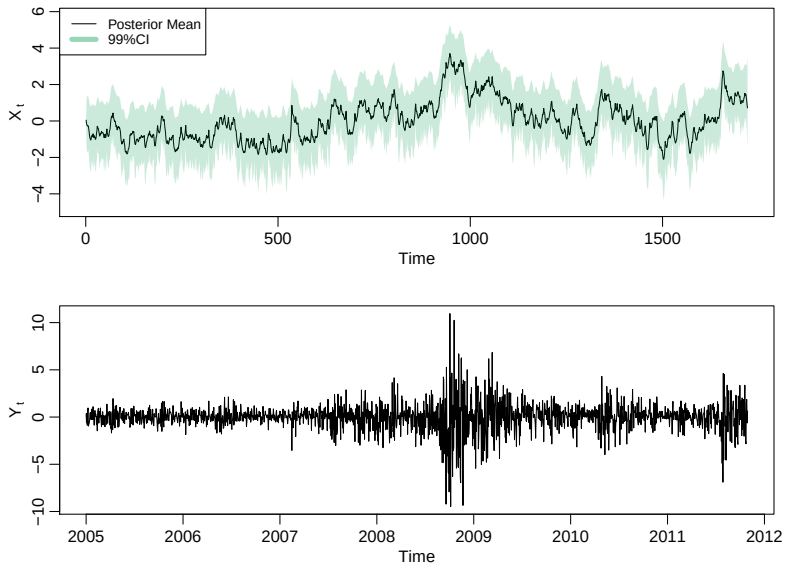

Figure 5.11: (Method B; S&P 500 data; Two-parameter SV model) *TOP: State process estimated posterior mean plot with 99% credible intervals. BOTTOM: The S&P 500 data plot.*

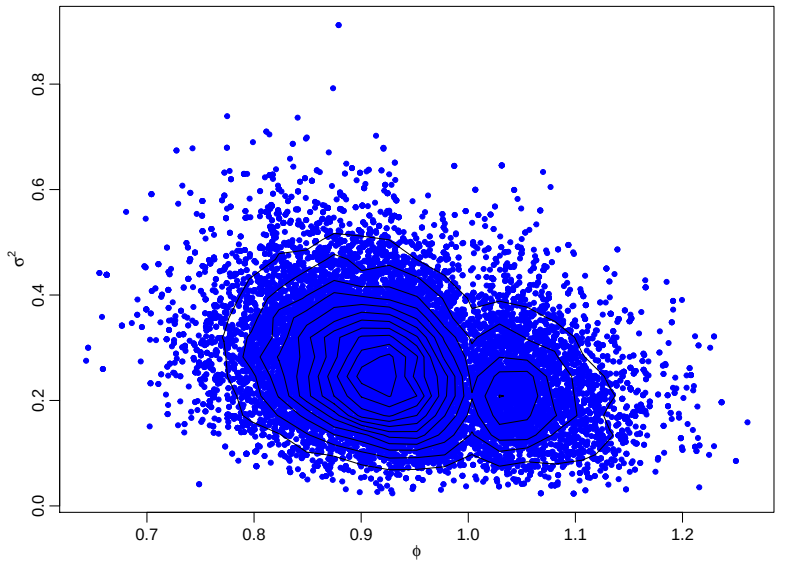

Figure 5.12: (Method B; S&P 500 data; Two-parameter SV model) *The contour plot of sampled parameters σ_ω^2 v.s. ϕ.*

Table 5.10: (Method B; S&P 500 data; Two-parameter SV model) *Summary statistics of sampled parameter ϕ.*

	Mean	SD	Naive SE	2.50%	97.50%
N=10	0.931	0.08469	0.00049	0.783	1.111
N=20	0.932	0.08379	0.00048	0.785	1.111
N=50	0.933	0.08475	0.00049	0.784	1.114
N=100	0.929	0.08394	0.00048	0.783	1.109
N=200	0.931	0.08339	0.00048	0.784	1.109
N=500	0.929	**0.08233**	**0.00048**	**0.783**	**1.106**

Table 5.11: (Method B; S&P 500 data; Two-parameter SV model) *Summary statistics of sampled parameter σ_ω^2.*

	Mean	SD	Naive SE	2.50%	97.50%
N=10	0.271	0.10500	0.00061	0.099	0.512
N=20	0.267	0.10180	0.00059	0.097	0.492
N=50	0.266	0.10368	0.00060	0.095	0.502
N=100	0.266	**0.10082**	**0.00058**	**0.099**	**0.496**
N=200	0.266	0.10264	0.00059	0.095	0.493
N=500	0.265	0.10163	0.00059	0.093	0.492

Furthermore, to different number of particles ($N = 10, 20, 50, 100, 200$ and 500), we compare our joint parameter sampling method (Method B) with the original individual parameter sampling method (Method A) by inefficiencies and summarize the results in Table 5.9. From the table, we can find that Method B is much more competent than Method A because the average inefficiencies of Method B is much lower than the average inefficiencies of Method A. Also, the lowest average inefficiencies of Method B occurs at $N = 20$ while it occurs at $N = 500$ for Method A. The summary statistics for the parameters ϕ and σ_ω^2 are given in Table 5.10 and Table 5.11, respectively. Using different number of particles gives very analogous sample means, standard deviations and 95% credible intervals. Note that a larger particle number does not lead to a narrower credible

interval. The narrowest 95% credible intervals for ϕ and σ_ω^2 are given by $N = 500$ and $N = 100$, respectively. Thus, it is not necessary to use a very large particle number when we employ the Method B. This will curtail the computational cost for running the MCMC procedure. For different number of particles, the detailed graphs of application studies of our Method B are provided in Appendix B.1.

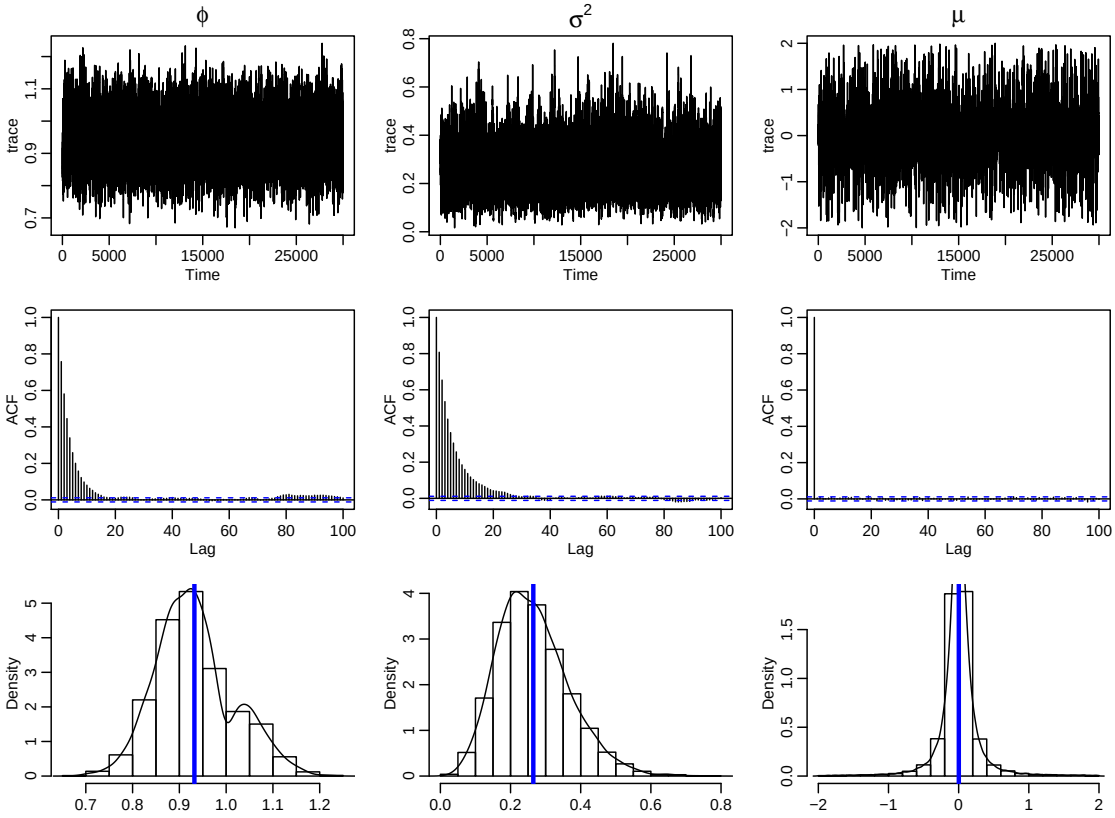

Figure 5.13: (Method B; S&P 500 data; Three-parameter SV model) *Results for S&P 500 data using the three-parameter SV model. TOP: Trace plots of sampled parameters. MIDDLE: the sample ACFs of the traces. BOTTOM: Histograms of sampled parameters. The blue lines show the sample mean of parameters. In total 33,000 iterations were drawn and discarded the first 3,000 as burn-in. The particles number is* $N = 20$.

Three-parameter SV model:

Next, we are going to use the three-parameter SV model to fit S&P 500 data by employing Method B. Employ the same bivariate normal prior distribution in the two-parameter SV model

case for (ϕ, σ_ω); that is, set $(\mu_\phi, \mu_{\sigma_\omega}) = (0.9, 0.5)$ and $(\sigma_\phi, \sigma_q) = (0.075, 0.1)$. Then we employ the non-informative Normal prior to μ. Give the number of particles as $N = 20$ and the initial value $(\phi^{[0]}, \sigma_\omega^{2\,[0]}, \mu^{[0]}) = (0.95, 0.2, 0.15)$. In addition, we use $\lambda = 1.7 \times 10^{-4}$ and $\boldsymbol{\Sigma}_\epsilon = \begin{bmatrix} 100 & -25 \\ -25 & 100 \end{bmatrix}$. Then, in total 33,000 iterations are drawn by MCMC sampler and we discard the first 3,000 samples as burn-in. The acceptance rate is about 40.18%.

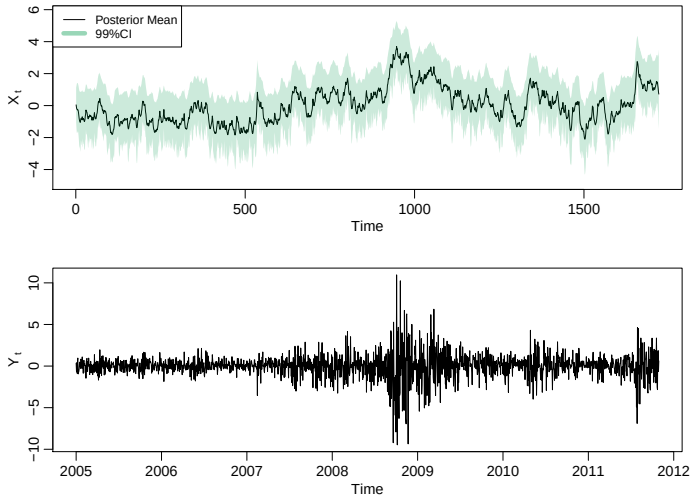

Figure 5.14: (Method B; S&P 500 data; Three-parameter SV model) *TOP: State process estimated posterior mean plot with 99% credible intervals. BOTTOM: The S&P 500 data plot.*

The detailed results of our application study are shown in Figure 5.13, 5.14 and 5.15. The autocorrelations for all three parameters ϕ, σ_ω^2 and μ decline extremely fast, which suggests the MCMC converges very quickly. The average inefficiencies for the three parameters is 6.759 and the inefficiencies of ϕ, σ_ω^2 and μ are 8.102, 11.058 and 1.117, respectively. The low inefficiencies implies that the joint parameter sampling method is remarkably efficient for the three-parameter SV model.

In addition, we also apply the our proposed method (Method B) using different numbers of particles ($N = 10, 20, 50, 100, 200$ and 500) for the S&P 500 data. Table 5.12 gives inefficiencies when we use different particles number. We can still observe very low inefficiencies for all parameters, which attests that Method B is efficient for the three-parameter SV model for the real world data. When the number of particles $N = 100$, Method B works more efficient than other cases.

Moreover, the inefficiencies are very robust to small number of particles N and we may use fewer particles ($N = 5 - 20$) to significantly save the computing time in practice.

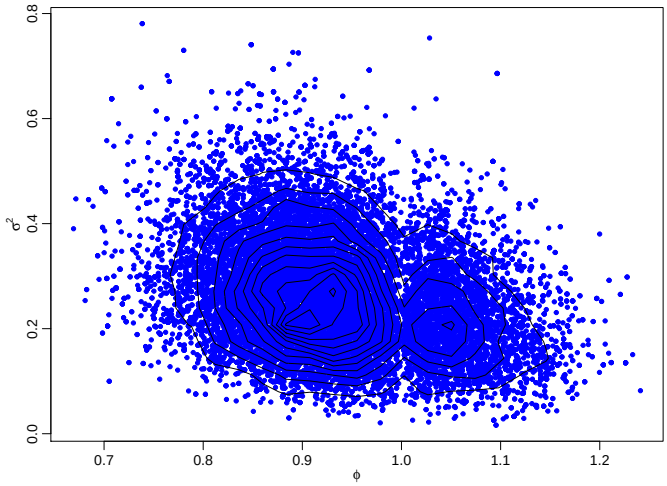

Figure 5.15: (Method B; S&P 500 data; Three-parameter SV model) *The contour plot of sampled parameters σ_ω^2 v.s. ϕ.*

Table 5.12: (Method B) *Inefficiencies for the three-parameter SV model for S&P 500 data. The three columns to the left are inefficiencies for ϕ, σ_ω^2 and μ, respectively. The last column is the average inefficiencies for the three parameters.*

	ϕ	σ_ω^2	μ	Average
N=10	7.704	9.688	1.021	6.138
N=20	8.102	11.058	1.117	6.759
N=50	7.624	10.897	1.026	6.516
N=100	7.302	8.918	1.002	**5.741**
N=200	8.470	10.687	1.017	6.724
N=500	8.248	9.293	0.977	6.173

We also show the summary statistics for the parameters ϕ, σ_ω^2 and μ, in Table 5.13, Table 5.14 and Table 5.15, respectively. Using different particles numbers, the application studies give comparable sample means, standard deviations and 95% credible intervals. The narrowest 95%

credible intervals for ϕ, σ_ω^2 and μ are given by $N = 100$, $N = 20$ and $N = 100$, respectively. For different number of particles, the graphs of simulation results of Method B are provided in Appendix A.2.

Table 5.13: (Method B; S&P 500 data; Three-parameter SV model) *Summary statistics of sampled parameter ϕ.*

	Mean	SD	Naive SE	2.50%	97.50%
N=10	0.933	0.08563	0.00049	0.781	1.116
N=20	0.932	0.08368	0.00048	0.785	1.113
N=50	0.931	0.08447	0.00049	0.780	1.110
N=100	0.932	**0.08297**	**0.00048**	**0.786**	**1.109**
N=200	0.931	0.08328	0.00048	0.785	1.110
N=500	0.931	0.08411	0.00049	0.783	1.109

Table 5.14: (Method B; S&P 500 data; Three-parameter SV model) *Summary statistics of sampled parameter σ_ω^2.*

	Mean	SD	Naive SE	2.50%	97.50%
N=10	0.268	0.10295	0.00059	0.097	0.501
N=20	0.265	**0.10135**	**0.00059**	**0.098**	**0.492**
N=50	0.268	0.10405	0.00060	0.097	0.506
N=100	0.265	0.10385	0.00060	0.094	0.496
N=200	0.267	0.10234	0.00059	0.098	0.497
N=500	0.266	0.10247	0.00059	0.095	0.495

Table 5.15: (Method B; S&P 500 data; Three-parameter SV model) *Summary statistics of sampled parameter μ.*

	Mean	SD	Naive SE	2.50%	97.50%
N=10	0.003	0.29867	0.00172	-0.590	0.614
N=20	0.002	0.29566	0.00171	-0.594	0.623
N=50	0.003	0.29262	0.00169	-0.586	0.614
N=100	0.002	**0.29225**	**0.00169**	**-0.593**	**0.605**
N=200	0.004	0.29861	0.00172	-0.593	0.613
N=500	0.000	0.30066	0.00174	-0.645	0.636

5.2.2 Proposed Method with Adaptive MCMC (Method C) for S&P 500 Data

In this section, we apply our proposed adaptive MCMC method (Method C) to modeling the S&P 500 data. The two-parameter SV model in (3.26)-(3.27) is used in this application study.

To apply Method C to the SV model, we set the hyperparameters of bivariate normal prior distribution as $(\mu_\phi, \mu_{\sigma_\omega}) = (0.9, 0.25)$ and $(\sigma_\phi, \sigma_q) = (0.25, 0.25)$. The correlation coefficient was set as $\rho = -0.25$ to represent the negative correlation between ϕ and σ_ω.

Table 5.16: (Method C; S&P 500 data; Two-parameter SV model) *Parameter estimation and summary statistics for sampled parameters. The average inefficiency for the two sampled parameters is 19.192.*

	Mean	SD	Naive SE	2.50%	97.50%	Inefficiency
ϕ	0.802	0.27214	0.00157	0.319	1.365	17.037
σ_ω^2	0.174	0.16924	0.00098	0.012	0.630	21.347

Then, we use the particles number $N = 20$ to do the MCMC sampling. For the adaptive algorithm part, we choose initial value of $\mathbf{\Sigma}_0 = 0.1 \times \begin{bmatrix} 1 & -0.25 \\ -0.25 & 1 \end{bmatrix}$ and $\lambda_0 = 2$. The γ_r in recursion formulas (4.12) - (4.14) is set as $\gamma_r = \frac{500}{r}$ and $\alpha_\star = 0.2$. Initializing the sampler

by setting $(\phi^{[0]}, \sigma_\omega^{2\,[0]}) = (0.95, 0.25)$, we run the MCMC for 33,000 iterations and the first 3,000 samples are discarded as burn-in.

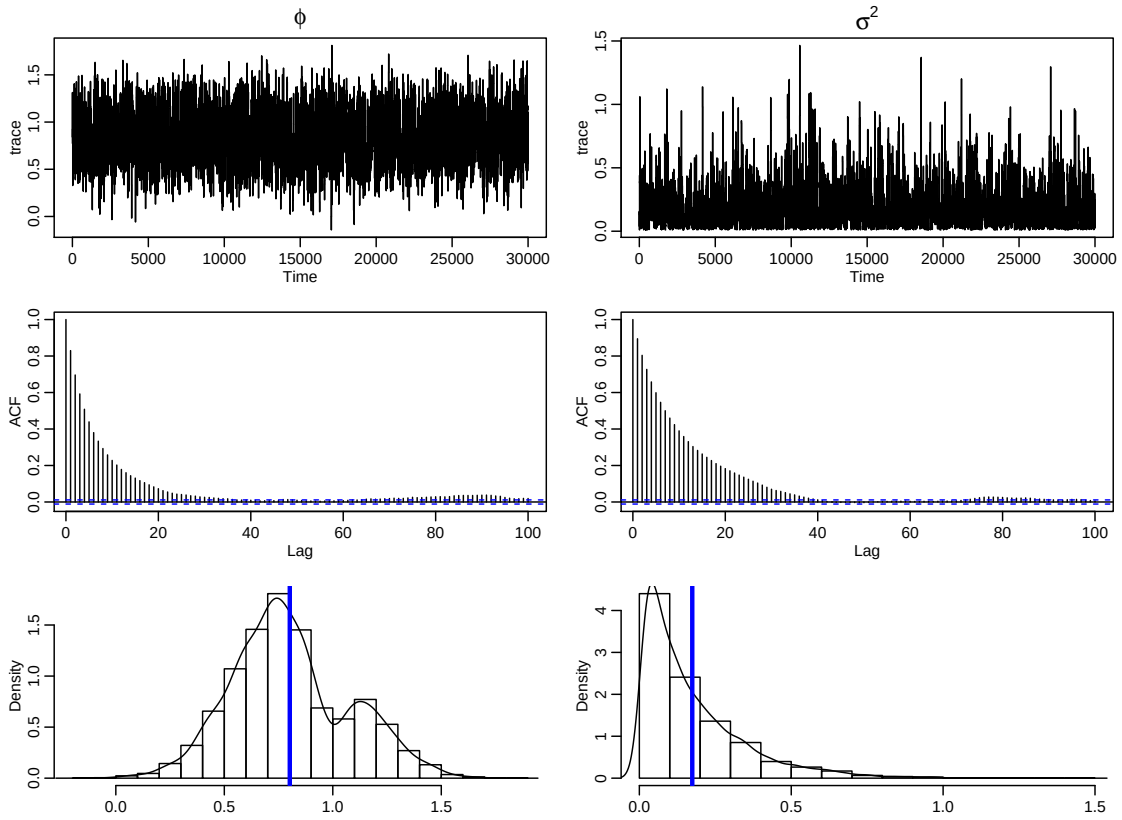

Figure 5.16: (Method C; S&P 500 data; Two-parameter SV model) *Results for S&P 500 data using the two-parameter SV model. TOP: Trace plots of sampled parameters. MIDDLE: the sample ACFs of the traces. BOTTOM: Histograms of sampled parameters. The blue lines show the sample mean of parameters. In total 33,000 samples were drawn and discarded the first 3,000 as burn-in. The particles number is* $N = 20$.

The autocorrelations for the sampled parameters were plotted in Figure 5.16. It's obvious that the ACFs decrease very fast for parameters ϕ and σ_ω^2. From the Table 5.16, the estimated inefficiencies of ϕ and σ_ω are 17.037 and 21.347, respectively. The average inefficiencies of the two sampled parameters is 19.192. From Table 5.9 and 5.16, we notice that our proposed adaptive algorithm (Method C) is still much more efficient than the existing method (Method A), even if the inefficiencies of Method C are larger than those in the non-adaptive method (Method B).

Although Method C may be less efficient than Method B, we may save more time on selecting a good proposal distribution.

In Figure 5.17, the estimated hidden states have relative small credible intervals. The contour plot in Figure 5.18 shows a negative correlation between ϕ and σ_ω^2. In Table 5.16, we give the sample means, 95% credible intervals and some other summary statistics for the parameters ϕ and σ_ω^2, respectively.

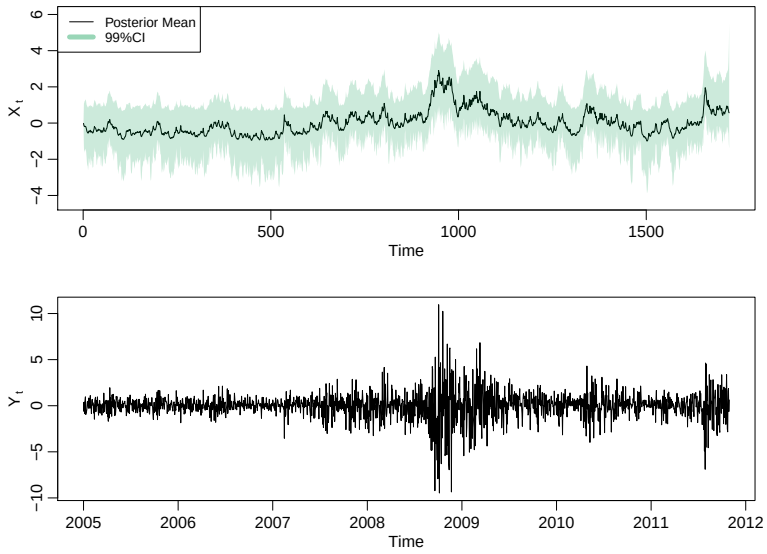

Figure 5.17: (Method C; S&P 500 data; Two-parameter SV model) *TOP: State process estimated posterior mean plot with 99% credible intervals. BOTTOM: The S&P 500 data plot.*

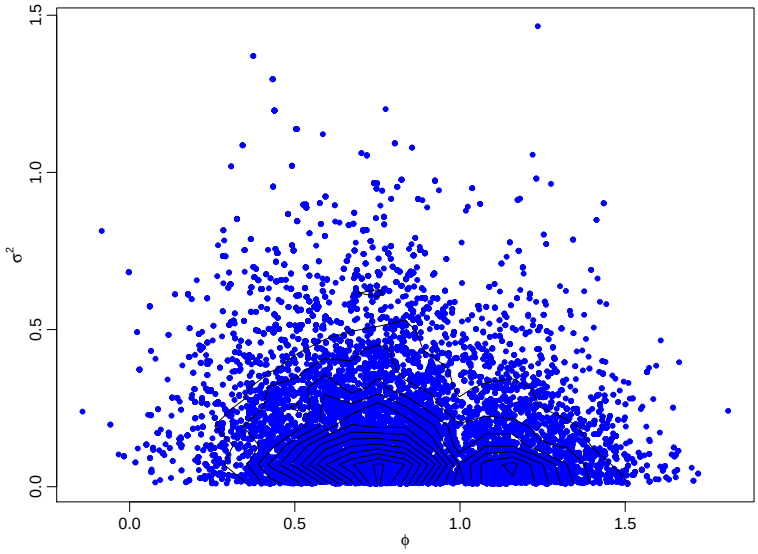

Figure 5.18: (Method C; S&P 500 data; Two-parameter SV model) *The contour plot of sampled parameters σ_ω^2 v.s. ϕ.*

5.3 APPLICATION ON MULTIPLE STOCKS DATA

In this section, we apply our proposed method in Section 4.3 to model some NYSE stocks data. Three famous banks' stocks, which are J.P. Morgan Chase (JPM), Bank of America (BOA) and Citigroup (Citi), will be used in this case study. We obtain their daily adjusted close stocks price S_t from Yahoo Finance. The stocks' price data are from 2007 to the end of 2013. The return of stock y_t is calculated by Equation (3.9). The returns of the three banks are shown in Figure 5.19. It's obvious that three stocks have a similar shape and we can assume that they may receive the same signal. Thus, the MSV model can be applied to model the volatility of three banks' stocks.

Note that the JPM series is the most stable one overall in Figure 5.19. The BOA and Citi series have higher volatility almost all the time, especially from 2008 to 2010. Thus, we may expect that the sample mean $\hat{\beta}_{JPM}$ is smaller than the others. $\hat{\beta}_{Citi}$ and $\hat{\beta}_{BOA}$ may have a analogous value.

In order to avoid overparameterizing the MSV model, we set μ as 0 in the univariate SV model and use the MSV in (3.10) and (3.11). The prior distributions of β_{BOA}, β_{Citi} and β_{JPM} are non-informative normal priors $N(0,\ \infty)$. For the bivariate Normal prior distribution of (ϕ, σ_ω), we set the hyperparameters $(\mu_\phi, \mu_{\sigma_\omega}) = (0.95, 0.5)$ and $(\sigma_\phi, \sigma_q) = (0.075, 0.1)$. Also, we set the correlation coefficient ρ as -0.25 to represent the negative correlation between ϕ and σ_ω^2. In order to achieve a great acceptance rate in RWMH Step 4 in the proposed algorithm, we choose $\lambda = 1.7 \times 10^{-4}$ and $\mathbf{\Sigma}_\epsilon = \begin{bmatrix} 100 & -25 \\ -25 & 100 \end{bmatrix}$.

Setting the number of particles as $N = 20$ and the initial value $(\phi^{[0]}, \sigma_\omega^{2\,[0]}, \beta_{BOA}^{[0]}, \beta_{Citi}^{[0]}, \beta_{JPM}^{[0]})$ as $(0.95, 0.15, 1.5, 1.5, 1.5)$, in total 33,000 samples are drawn through MCMC and we discarded the first 3,000 iterations as burn-in. The acceptance rate for RWMH is about 36.02%.

The results are shown in Figure 5.20. Compared to previous results, autocorrelations still have excellent performances even in multivariate case, especially for the parameters ϕ and σ_ω^2. The estimated inefficiencies of parameters are listed in Table 5.17. Conspicuously, our proposed method works efficiently for the MSV model. In addition, the posterior mean of hidden state x_t is shown at the top of Figure 5.21. The 99% credible interval for posterior mean is very narrow and the trend of the posterior mean describes the stocks' volatility very well.

Table 5.17: *Parameter estimation and summary statistics for the MSV model for the three banks data. The average inefficiency for the five parameters is 9.871.*

	ϕ	σ_ω^2	β_{BOA}	β_{Citi}	β_{JPM}
Mean	0.914	0.357	2.357	2.449	1.992
SD	0.08180	0.11542	0.05905	0.06340	0.04865
Naive SE	0.00047	0.00067	0.00034	0.00037	0.00028
95% C.I.	(0.769, 1.093)	(0.157, 0.610)	(2.246, 2.478)	(2.331, 2.579)	(1.900, 2.091)
Inefficiency	8.342	17.229	8.010	8.783	6.988

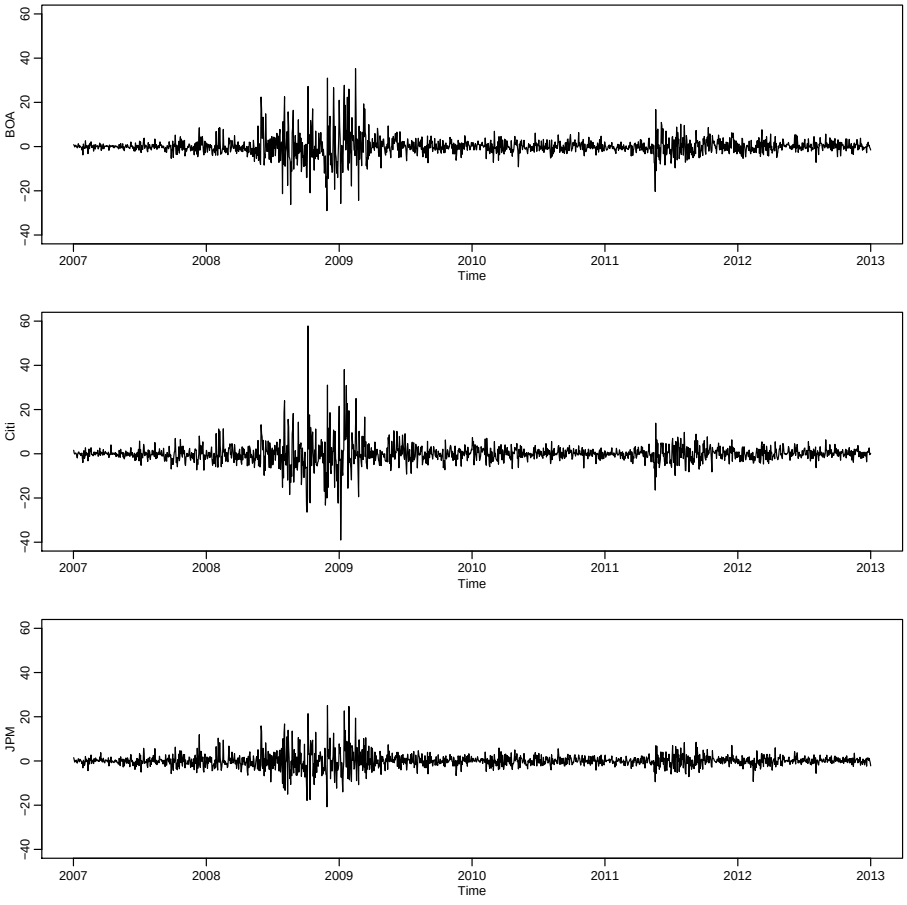

Figure 5.19: *Stock returns plot of Bank of America, Citi bank and J.P. Morgan bank from 2007 to 2013.*

The sample mean of parameters and some statistics are also shown in Table 5.17. The sample mean of β_{JPM} (1.992) is the smallest and it is significantly smaller than the sample mean of β_{BOA} (2.357) and β_{Citi} (2.449). It means that the return of J.P. Morgan's stock has the least volatility among the returns of three banks, which proves our previous finding from Figure 5.19. We may use the sample mean of β_i to rank the volatility of stocks. Also, the 95% credible intervals for all parameters are provided in the table.

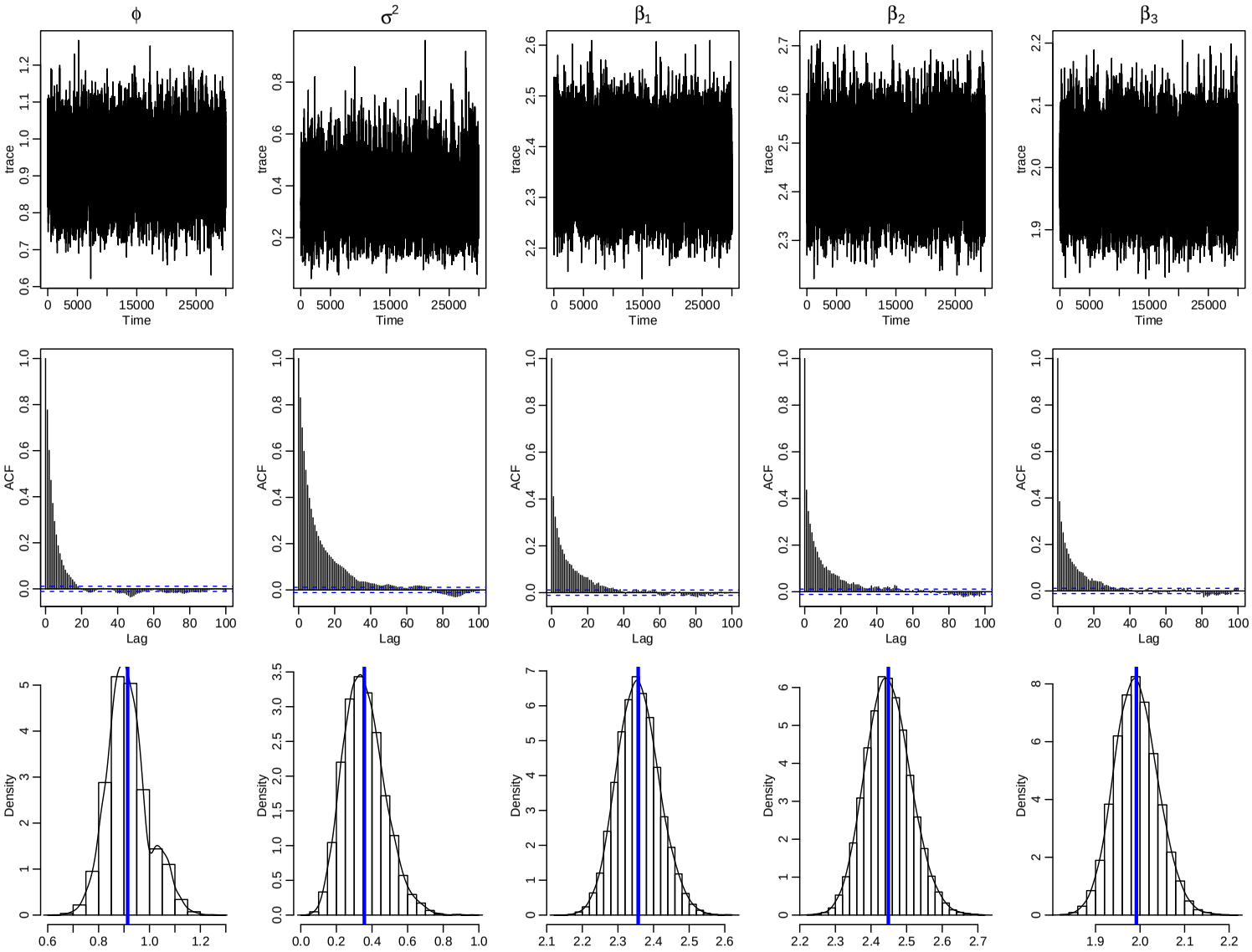

Figure 5.20: *Results for the three banks data using the MSV model. TOP: Trace plots of sampled parameters. MIDDLE: the sample ACFs of the traces. BOTTOM: Histograms of sampled parameters. The blue lines show the sample mean of parameters. In total 33,000 samples were drawn and discarded the first 3,000 as burn-in. The particles number is* $N = 20$.

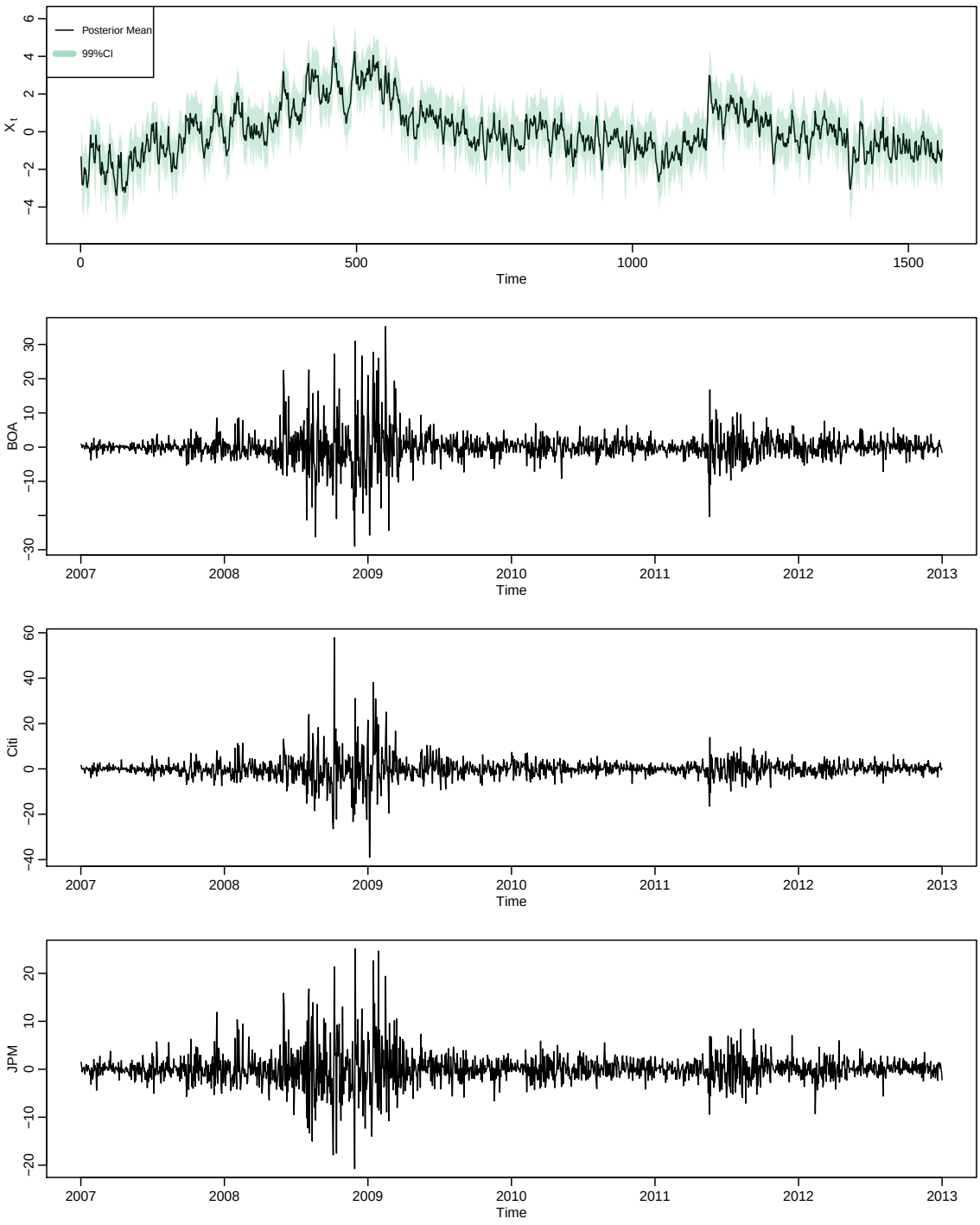

Figure 5.21: *Top: State process estimated posterior mean plot with 99% credible intervals. The other plots: The stocks volatility plots for Bank of America, Citi bank and J.P. Morgan bank, respectively.*

6.0 CONCLUDING REMARKS

The conditional particle filter with ancestral sampling (Algorithm 4) was a breakthrough for analyzing nonlinear state space models. The algorithm can efficiently draw samples from the posterior of the hidden state trajectories. However, when we applied the algorithm to stochastic volaitility model, the low efficiency problem can still be observed in Section 3.2. The reason is that the autoregressive parameter ϕ and the noise variance σ_ω^2 in the state equation have a strong negative correlation and they tend to work in opposite directions. The existing methods are trying to sample ϕ and σ_ω^2 individually from the posterior distribution and this will result in the inefficiency of algorithm.

In this thesis, we have proposed a method to overcome this problem by sampling ϕ and σ_ω^2 jointly from the posterior distribution. We employed a bivariate normal distribution for ϕ and σ_ω^2 and the negative correlation can be explained by a negative correlation coefficient, which is predetermined. From the simulation studies and real world data applications in Chapter 5, we showed that the new method (Algorithm 7) works very efficiently on stochastic volaitility model. The new method uses Random Walk Metropolis Hastings when it samples the ϕ and σ_ω^2 from the posterier distribution. To overcome the difficulty in finding a great proposal distribution for Random Walk Metropolis Hastings, we also presented an adaptive MCMC method (Algorithm 9). At last, we extended our new method to multivariate stochastic volaitility model in Section 4.3 (Algorithm 10).

APPENDIX A

SIMULATION STUDIES RESULTS FOR DIFFERENT PARTICLE NUMBERS

A.1 RESULTS FOR THE TWO-PARAMETER SV MODEL

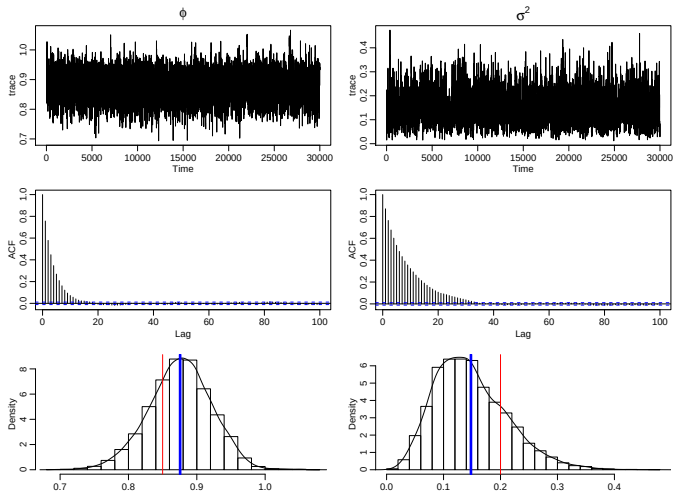

Figure A.1: (Method B; Simulated data; Two-parameter SV model) *Results for simulated data using the two-parameter SV model. TOP: Trace plots of sampled parameters. MIDDLE: the sample ACFs of the traces. BOTTOM: Histograms of sampled parameters. The blue line and red line show the sample mean and true value of parameter, respectively. In total 33,000 iterations were drawn and discarded the first 3,000 as burn-in. The particles number is* $N = 10$.

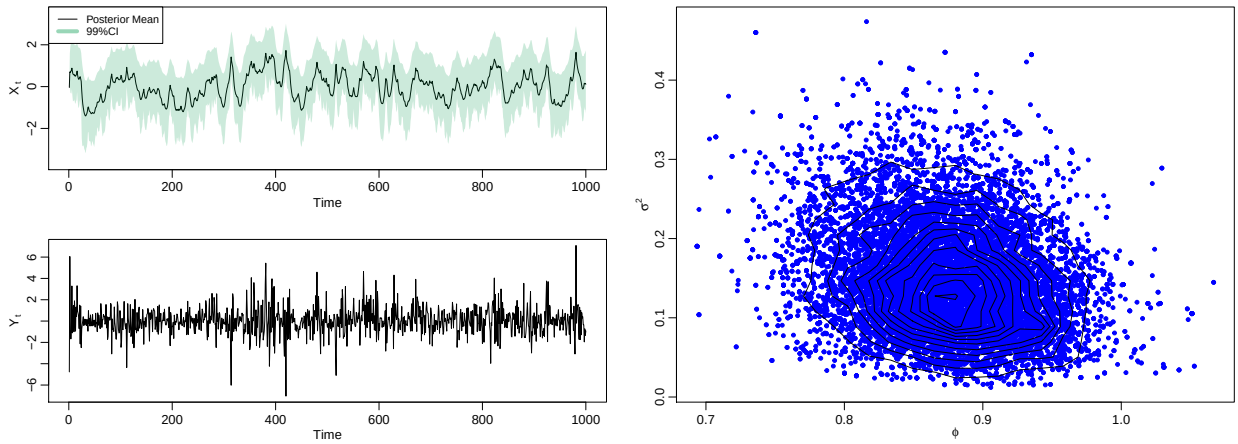

Figure A.2: (Method B; Simulated data; Two-parameter SV model; N=10) *LEFT TOP: State process estimated posterior mean plot with 99% credible intervals. LEFT BOTTOM: The simulated data. RIGHT: The contour plot of sampled parameters* σ_ω^2 *v.s.* ϕ*.*

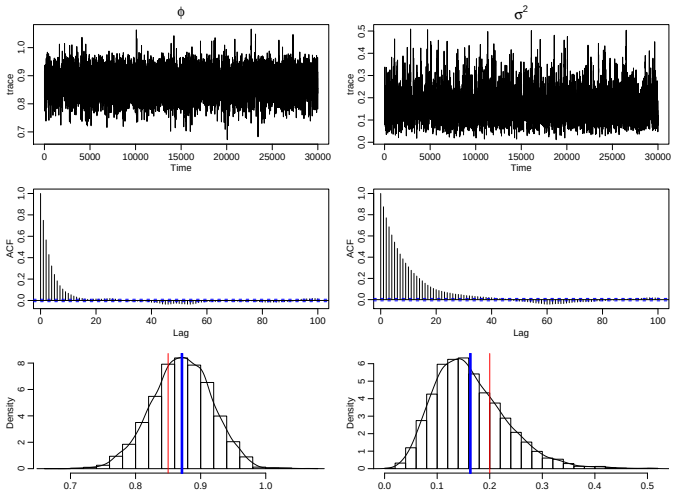

Figure A.3: (Method B; Simulated data; Two-parameter SV model) *Results for simulated data using the two-parameter SV model. TOP: Trace plots of sampled parameters. MIDDLE: the sample ACFs of the traces. BOTTOM: Histograms of sampled parameters. The blue line and red line show the sample mean and true value of parameter, respectively. In total 33,000 iterations were drawn and discarded the first 3,000 as burn-in. The particles number is* $N = 50$*.*

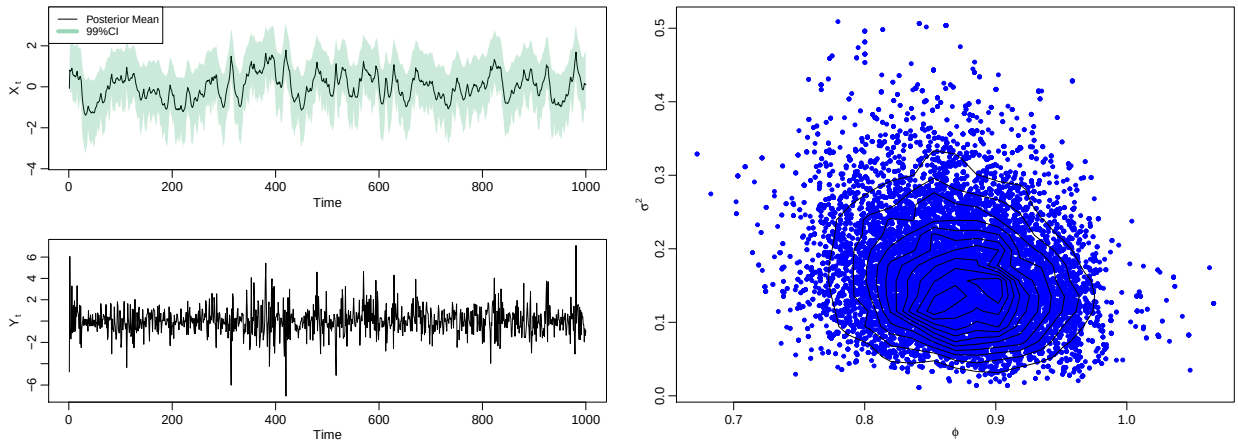

Figure A.4: (Method B; Simulated data; Two-parameter SV model; N=50) *LEFT TOP: State process estimated posterior mean plot with 99% credible intervals. LEFT BOTTOM: The simulated data. RIGHT: The contour plot of sampled parameters σ_ω^2 v.s. ϕ.*

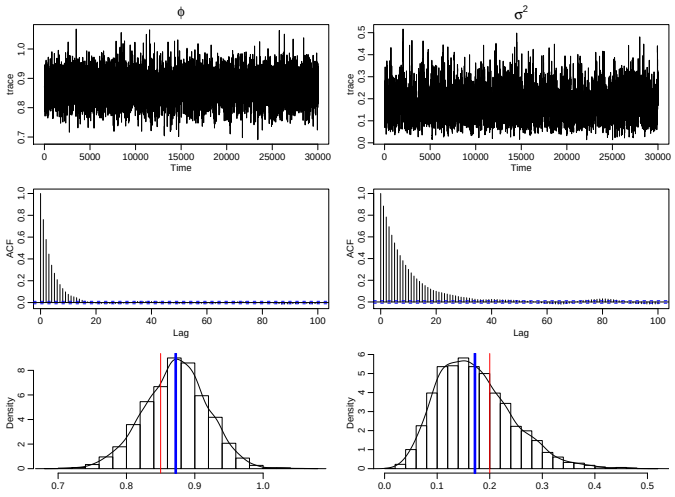

Figure A.5: (Method B; Simulated data; Two-parameter SV model) *Results for simulated data using the two-parameter SV model. TOP: Trace plots of sampled parameters. MIDDLE: the sample ACFs of the traces. BOTTOM: Histograms of sampled parameters. The blue line and red line show the sample mean and true value of parameter, respectively. In total 33,000 iterations were drawn and discarded the first 3,000 as burn-in. The particles number is $N = 100$.*

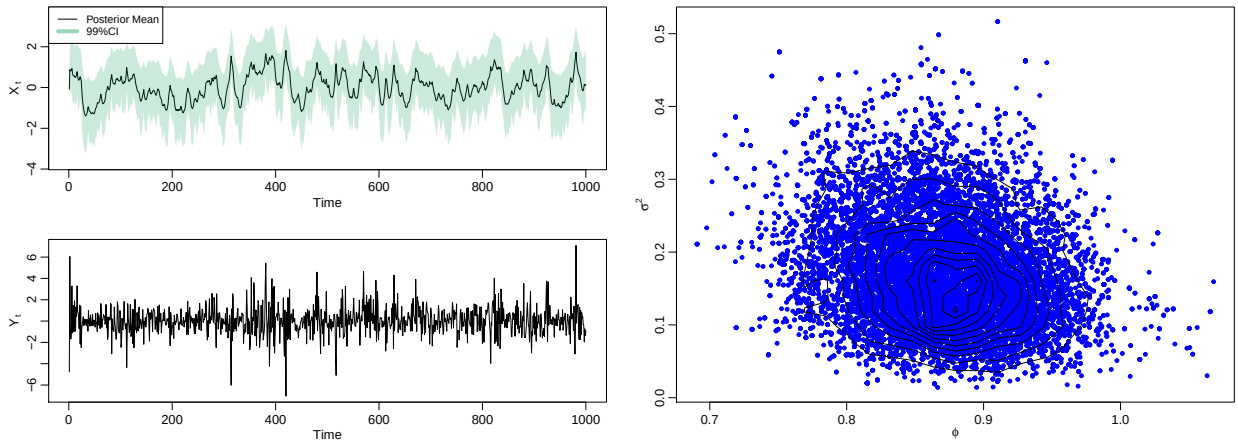

Figure A.6: (Method B; Simulated data; Two-parameter SV model; N=100) *LEFT TOP: State process estimated posterior mean plot with 99% credible intervals. LEFT BOTTOM: The simulated data. RIGHT: The contour plot of sampled parameters σ_ω^2 v.s. ϕ.*

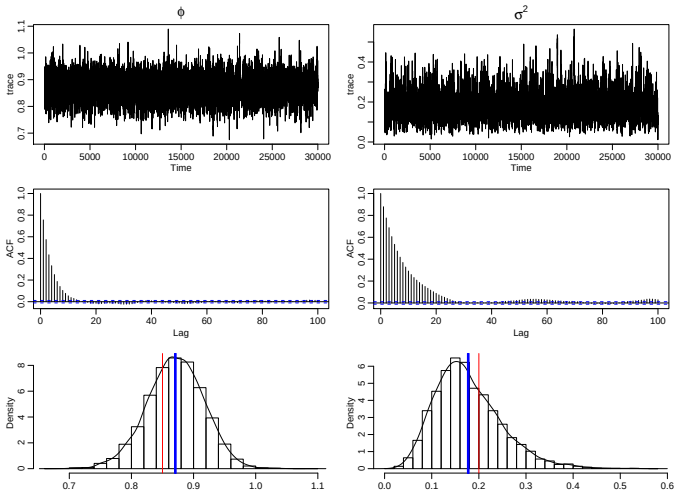

Figure A.7: (Method B; Simulated data; Two-parameter SV model) *Results for simulated data using the two-parameter SV model. TOP: Trace plots of sampled parameters. MIDDLE: the sample ACFs of the traces. BOTTOM: Histograms of sampled parameters. The blue line and red line show the sample mean and true value of parameter, respectively. In total 33,000 iterations were drawn and discarded the first 3,000 as burn-in. The particles number is $N = 200$.*

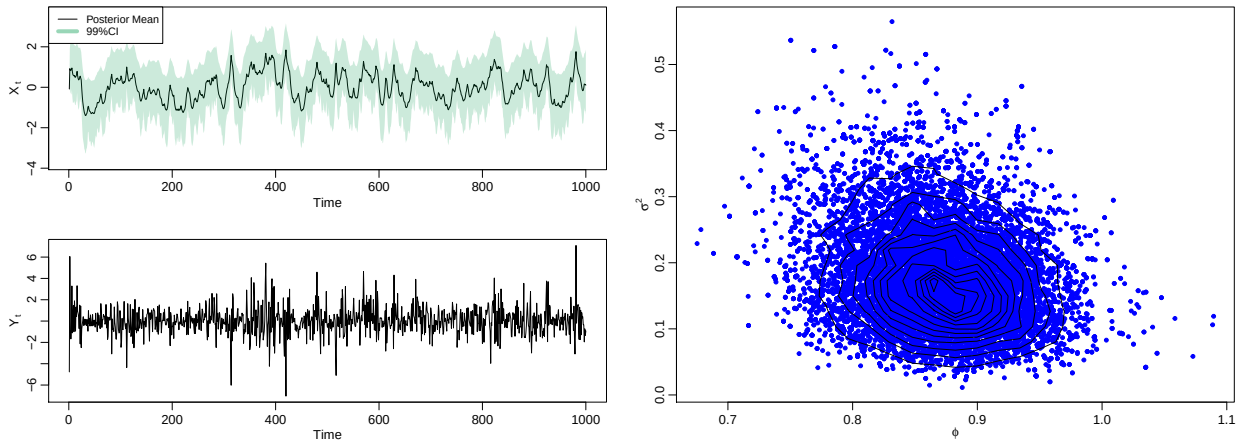

Figure A.8: (Method B; Simulated data; Two-parameter SV model; N=200) *LEFT TOP: State process estimated posterior mean plot with 99% credible intervals. LEFT BOTTOM: The simulated data. RIGHT: The contour plot of sampled parameters σ_ω^2 v.s. ϕ.*

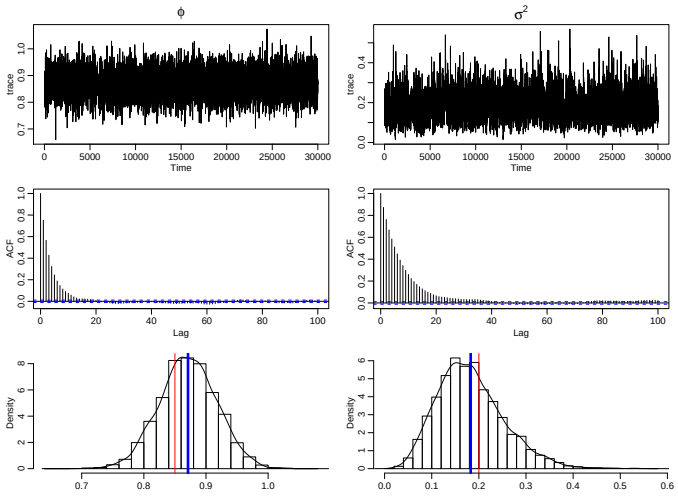

Figure A.9: (Method B; Simulated data; Two-parameter SV model) *Results for simulated data using the two-parameter SV model. TOP: Trace plots of sampled parameters. MIDDLE: the sample ACFs of the traces. BOTTOM: Histograms of sampled parameters. The blue line and red line show the sample mean and true value of parameter, respectively. In total 33,000 iterations were drawn and discarded the first 3,000 as burn-in. The particles number is $N = 500$.*

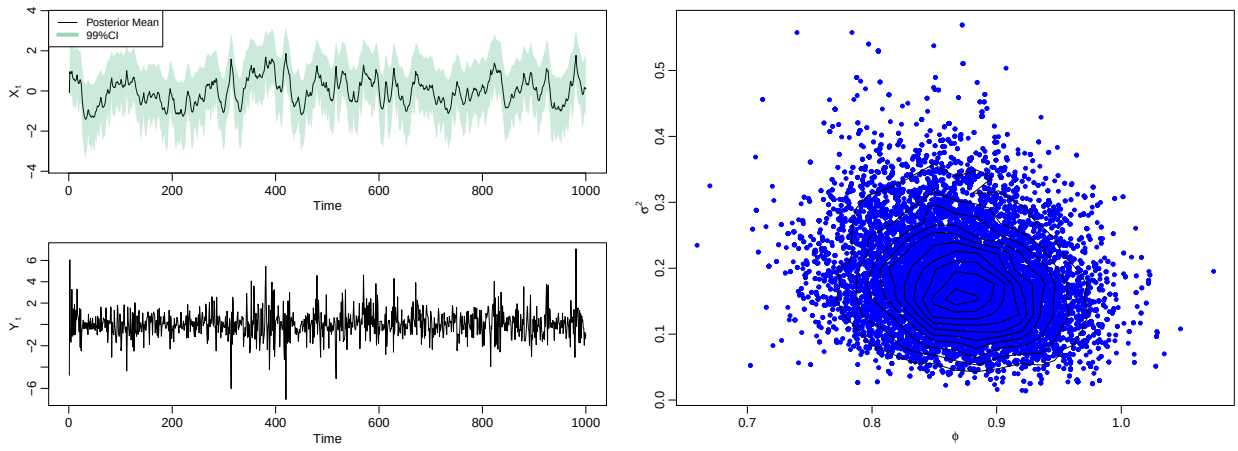

Figure A.10: (Method B; Simulated data; Two-parameter SV model; N=500) *LEFT TOP: State process estimated posterior mean plot with 99% credible intervals. LEFT BOTTOM: The simulated data. RIGHT: The contour plot of sampled parameters* σ_ω^2 *v.s.* ϕ.

A.2 RESULTS FOR THE THREE-PARAMETER SV MODEL

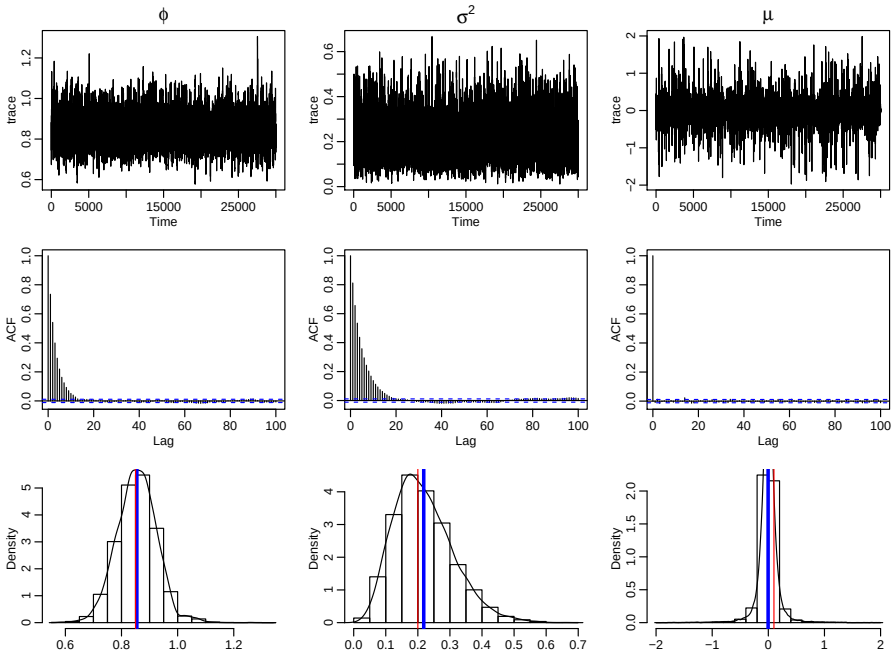

Figure A.11: (Method B; Simulated data; Three-parameter SV model) *Results for simulated data using the two-parameter SV model. TOP: Trace plots of sampled parameters. MIDDLE: the sample ACFs of the traces. BOTTOM: Histograms of sampled parameters. The blue line and red line show the sample mean and true value of parameter, respectively. In total 33,000 iterations were drawn and discarded the first 3,000 as burn-in. The particles number is* $N = 10$.

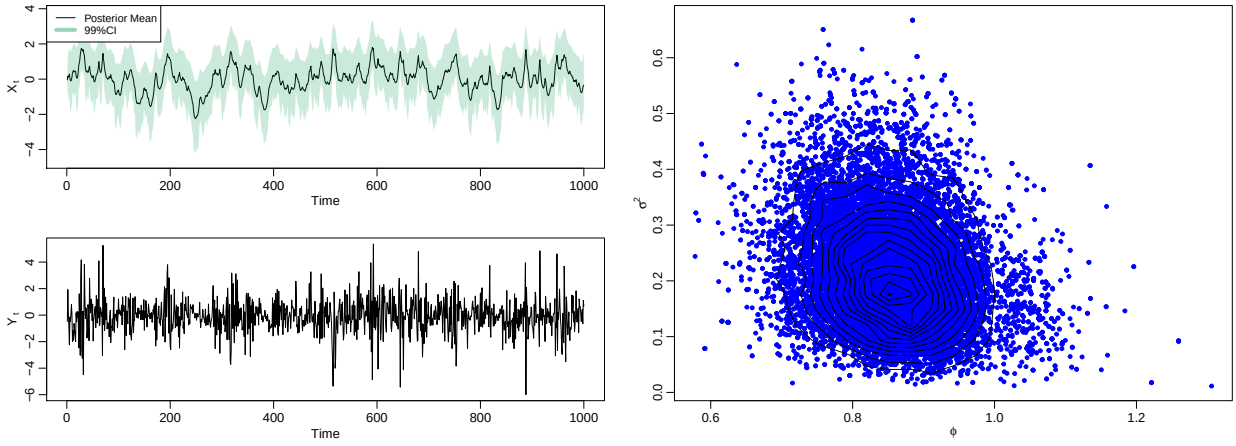

Figure A.12: (Method B; Simulated data; Three-parameter SV model; N=10) *LEFT TOP: State process estimated posterior mean plot with 99% credible intervals. LEFT BOTTOM: The simulated data. RIGHT: The contour plot of sampled parameters σ_ω^2 v.s. ϕ.*

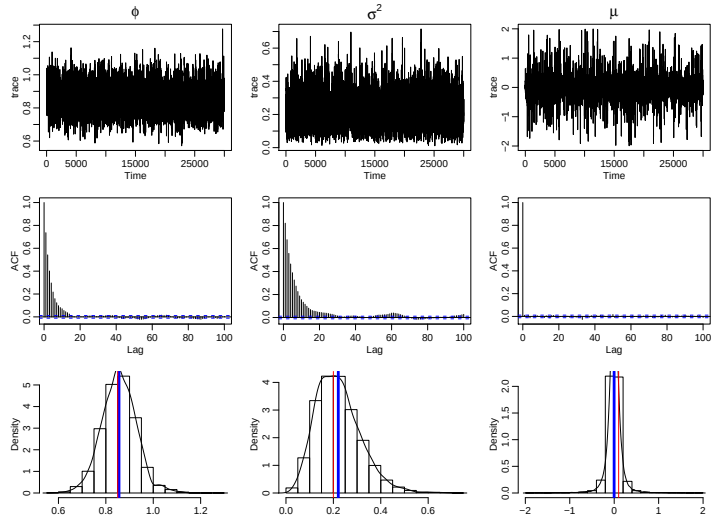

Figure A.13: (Method B; Simulated data; Three-parameter SV model) *Results for simulated data using the two-parameter SV model. TOP: Trace plots of sampled parameters. MIDDLE: the sample ACFs of the traces. BOTTOM: Histograms of sampled parameters. The blue line and red line show the sample mean and true value of parameter, respectively. In total 33,000 iterations were drawn and discarded the first 3,000 as burn-in. The particles number is $N = 50$.*

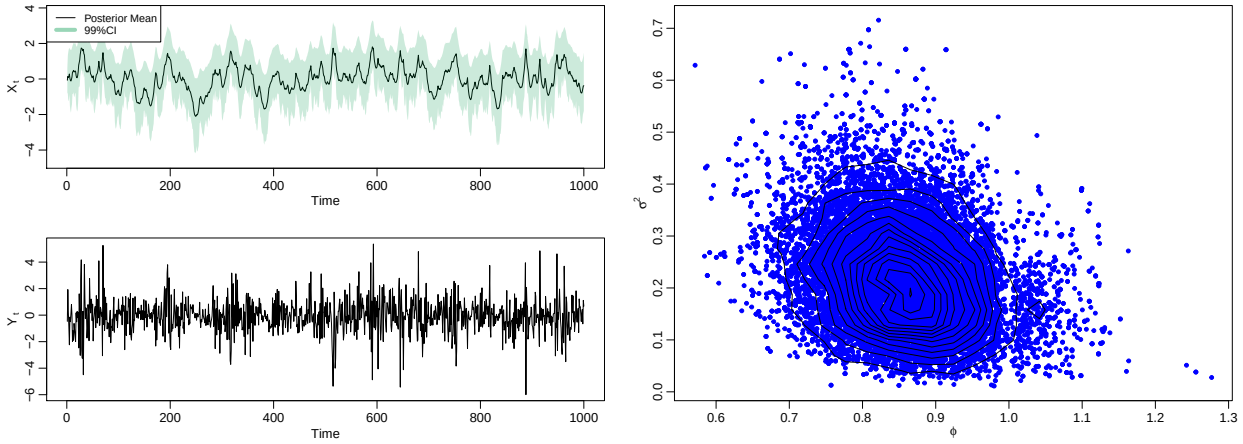

Figure A.14: (Method B; Simulated data; Three-parameter SV model; N=50) *LEFT TOP: State process estimated posterior mean plot with 99% credible intervals. LEFT BOTTOM: The simulated data. RIGHT: The contour plot of sampled parameters σ_ω^2 v.s. ϕ.*

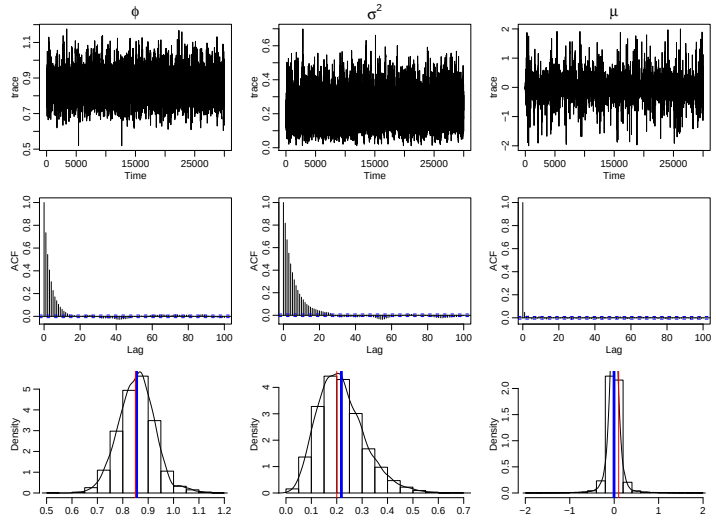

Figure A.15: (Method B; Simulated data; Three-parameter SV model) *Results for simulated data using the two-parameter SV model. TOP: Trace plots of sampled parameters. MIDDLE: the sample ACFs of the traces. BOTTOM: Histograms of sampled parameters. The blue line and red line show the sample mean and true value of parameter, respectively. In total 33,000 iterations were drawn and discarded the first 3,000 as burn-in. The particles number is $N = 100$.*

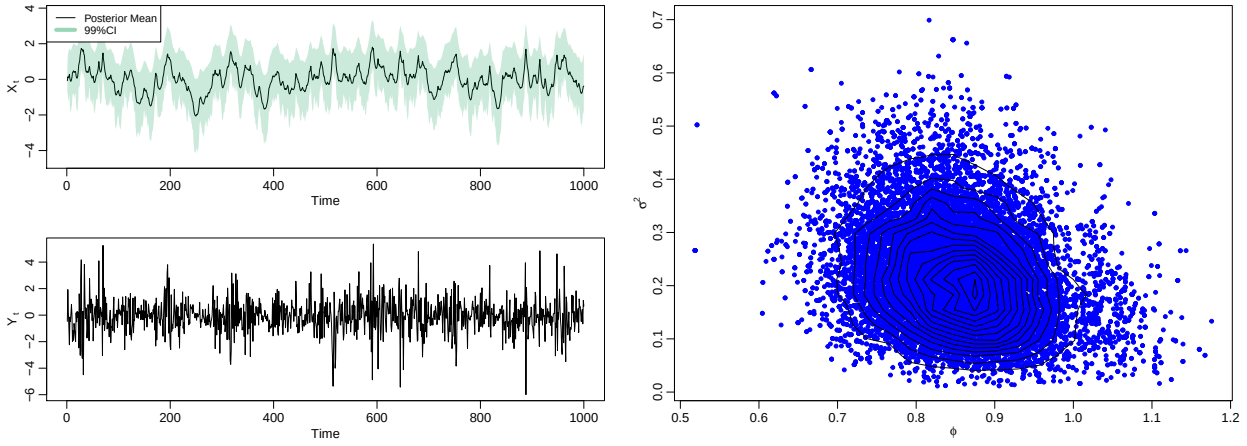

Figure A.16: (Method B; Simulated data; Three-parameter SV model; N=100) *LEFT TOP: State process estimated posterior mean plot with 99% credible intervals. LEFT BOTTOM: The simulated data. RIGHT: The contour plot of sampled parameters σ_ω^2 v.s. ϕ.*

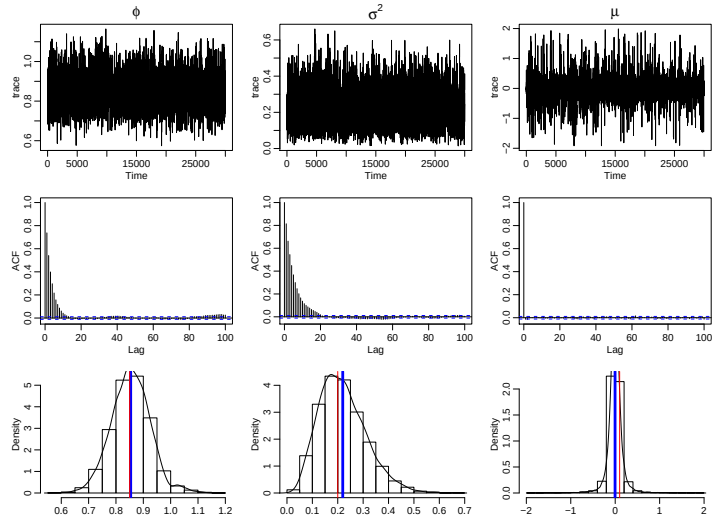

Figure A.17: (Method B; Simulated data; Three-parameter SV model) *Results for simulated data using the two-parameter SV model. TOP: Trace plots of sampled parameters. MIDDLE: the sample ACFs of the traces. BOTTOM: Histograms of sampled parameters. The blue line and red line show the sample mean and true value of parameter, respectively. In total 33,000 iterations were drawn and discarded the first 3,000 as burn-in. The particles number is $N = 200$.*

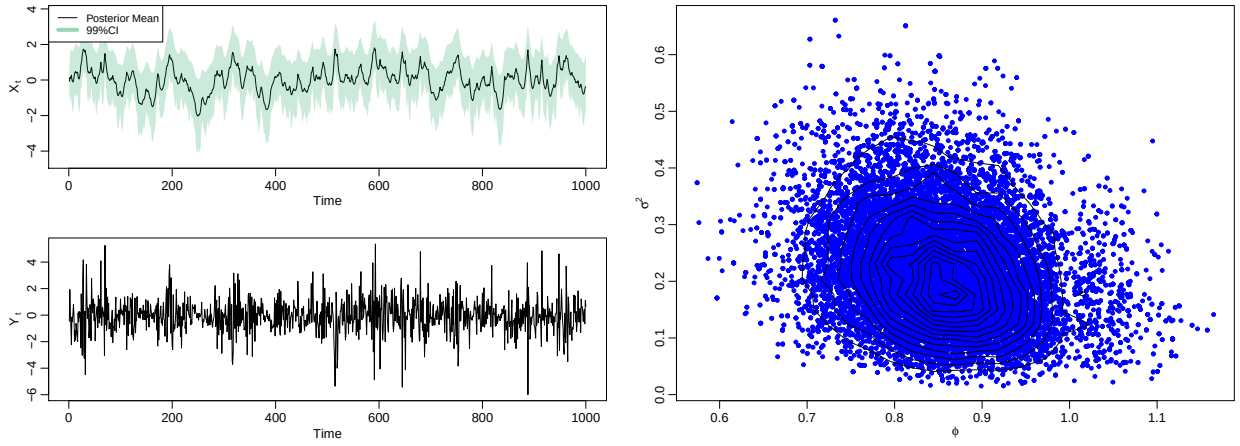

Figure A.18: (Method B; Simulated data; Three-parameter SV model; N=200) *LEFT TOP: State process estimated posterior mean plot with 99% credible intervals. LEFT BOTTOM: The simulated data. RIGHT: The contour plot of sampled parameters σ_ω^2 v.s. ϕ.*

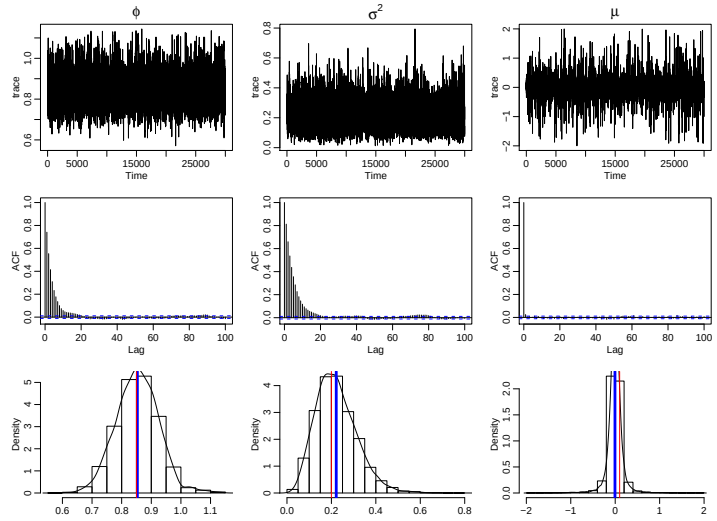

Figure A.19: (Method B; Simulated data; Three-parameter SV model) *Results for simulated data using the two-parameter SV model. TOP: Trace plots of sampled parameters. MIDDLE: the sample ACFs of the traces. BOTTOM: Histograms of sampled parameters. The blue line and red line show the sample mean and true value of parameter, respectively. In total 33,000 iterations were drawn and discarded the first 3,000 as burn-in. The particles number is $N = 500$.*

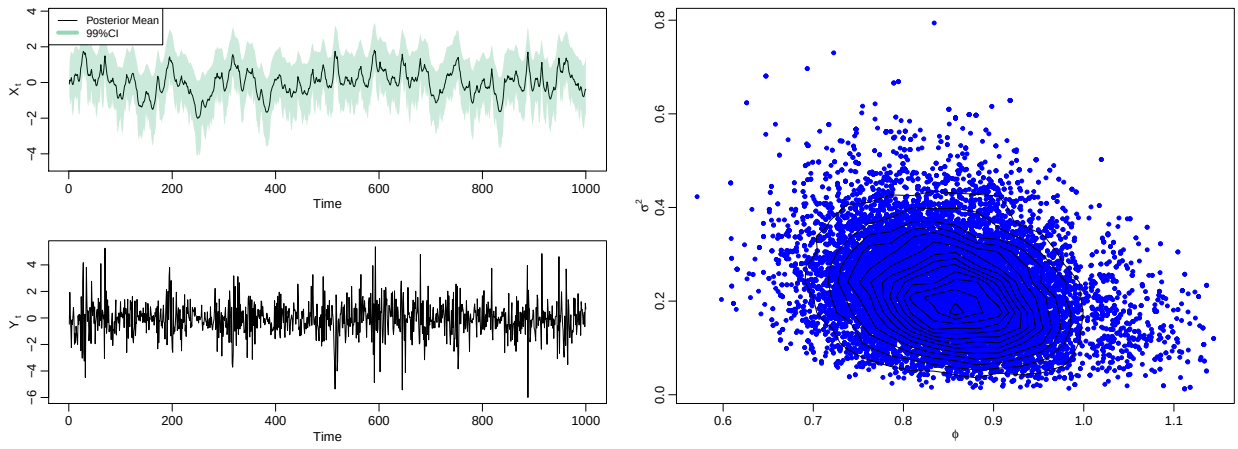

Figure A.20: (Method B; Simulated data; Three-parameter SV model; N=500) *LEFT TOP: State process estimated posterior mean plot with 99% credible intervals. LEFT BOTTOM: The simulated data. RIGHT: The contour plot of sampled parameters* σ_ω^2 *v.s.* ϕ.

APPENDIX B

S&P 500 DATA MODELING RESULTS FOR DIFFERENT PARTICLE NUMBERS

B.1 RESULTS FOR THE TWO-PARAMETER SV MODEL

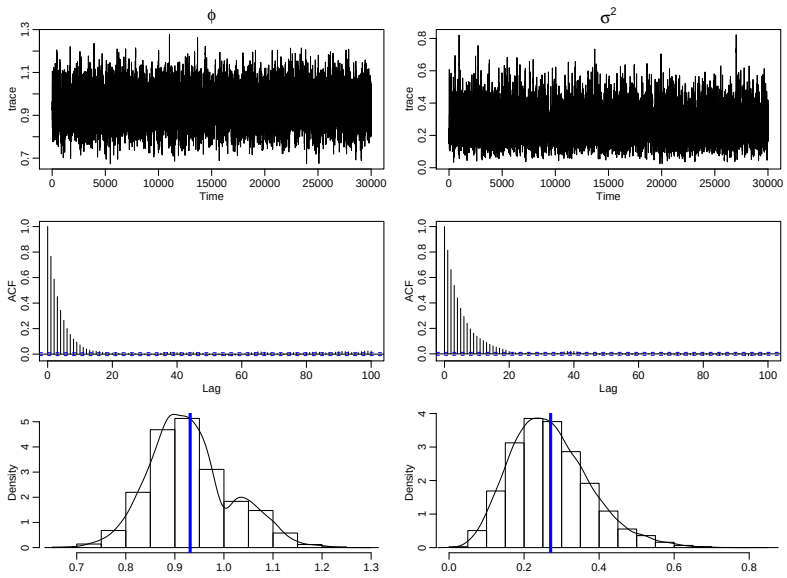

Figure B.1: (Method B; S&P 500 data; Two-parameter SV model) *Results for S&P 500 data using the two-parameter SV model. TOP: Trace plots of sampled parameters. MIDDLE: the sample ACFs of the traces. BOTTOM: Histograms of sampled parameters. The blue lines show the sample mean of parameters. In total 33,000 iterations were drawn and discarded the first 3,000 as burn-in. The particles number is* $N = 10$.

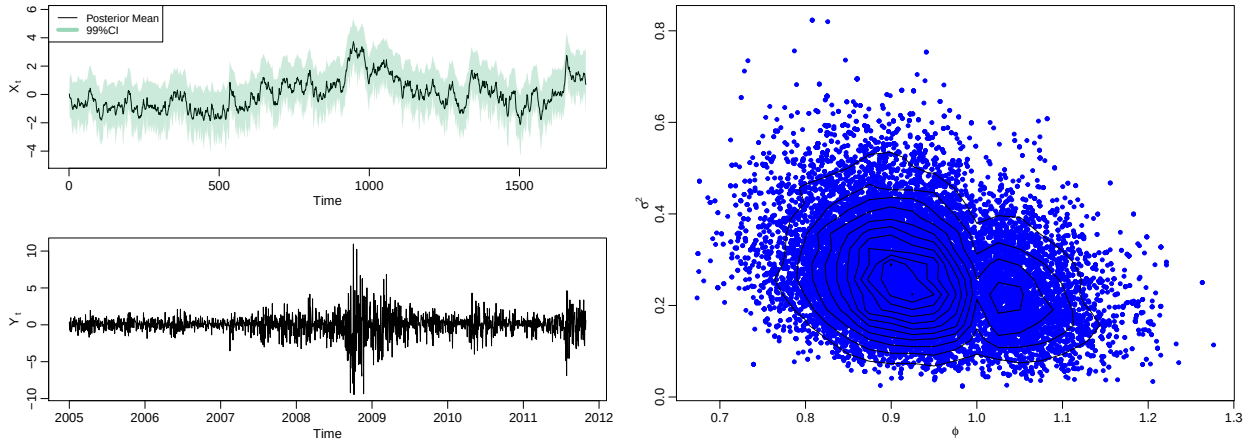

Figure B.2: (Method B; S&P 500 data; Two-parameter SV model; N=10) *LEFT TOP: State process estimated posterior mean plot with 99% credible intervals. LEFT BOTTOM: The S&P 500 data. RIGHT: The contour plot of sampled parameters σ_ω^2 v.s. ϕ.*

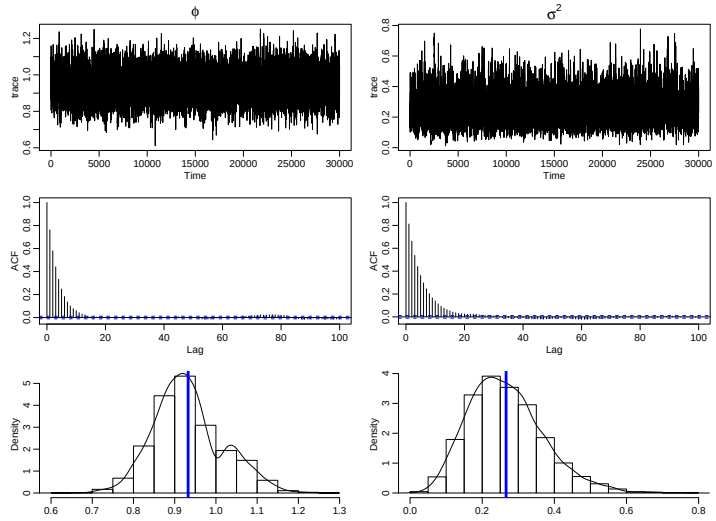

Figure B.3: (Method B; S&P 500 data; Two-parameter SV model) *Results for S&P 500 data using the two-parameter SV model. TOP: Trace plots of sampled parameters. MIDDLE: the sample ACFs of the traces. BOTTOM: Histograms of sampled parameters. The blue lines show the sample mean of parameters. In total 33,000 iterations were drawn and discarded the first 3,000 as burn-in. The particles number is $N = 50$.*

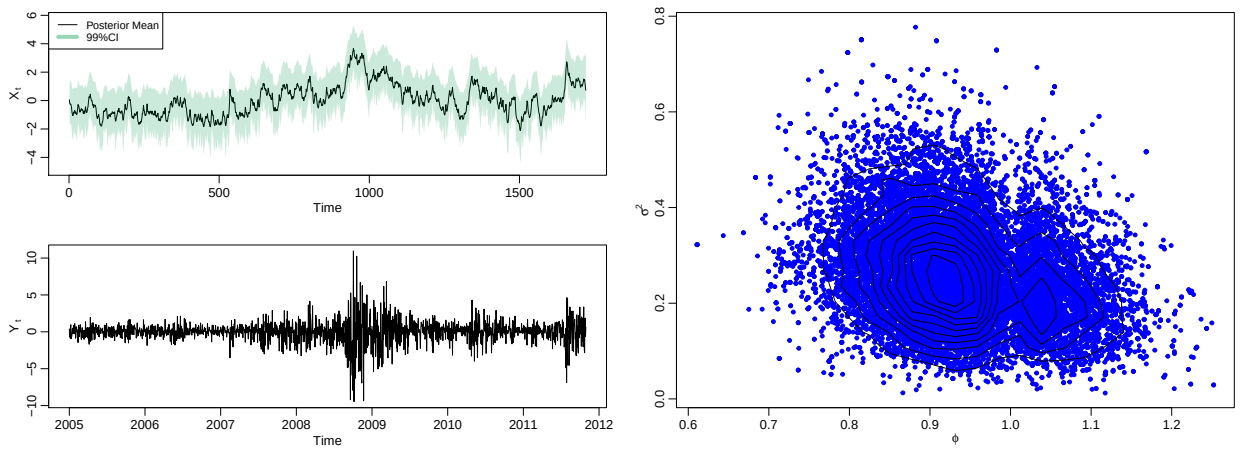

Figure B.4: (Method B; S&P 500 data; Two-parameter SV model; N=50) *LEFT TOP: State process estimated posterior mean plot with 99% credible intervals. LEFT BOTTOM: The S&P 500 data. RIGHT: The contour plot of sampled parameters σ_ω^2 v.s. ϕ.*

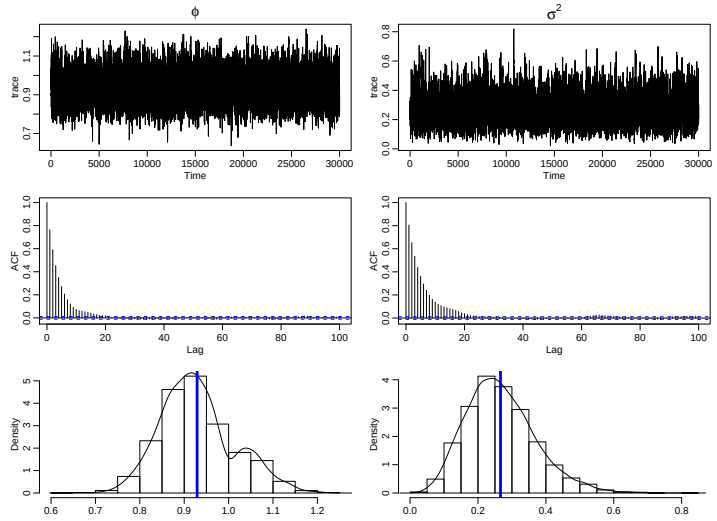

Figure B.5: (Method B; S&P 500 data; Two-parameter SV model) *Results for S&P 500 data using the two-parameter SV model. TOP: Trace plots of sampled parameters. MIDDLE: the sample ACFs of the traces. BOTTOM: Histograms of sampled parameters. The blue lines show the sample mean of parameters. In total 33,000 iterations were drawn and discarded the first 3,000 as burn-in. The particles number is $N = 100$.*

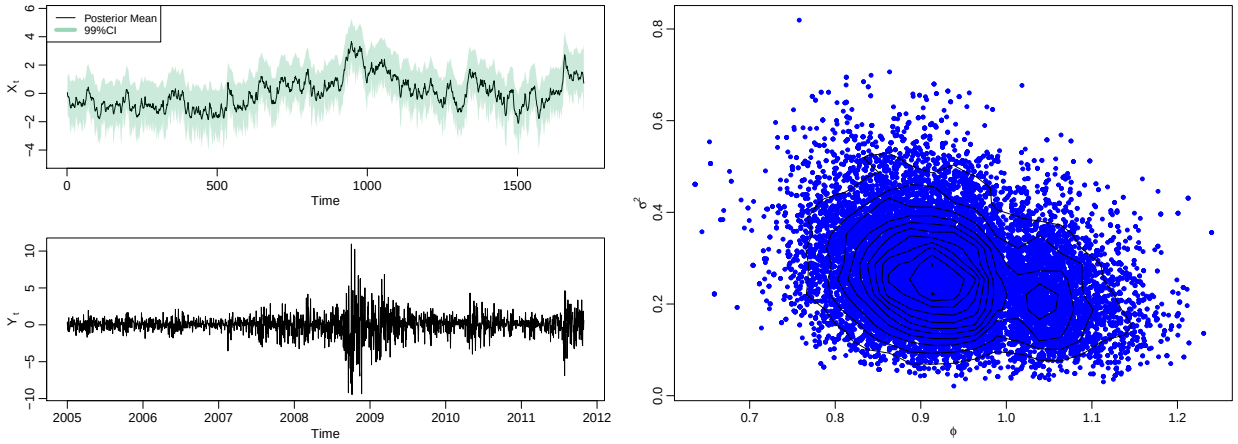

Figure B.6: (Method B; S&P 500 data; Two-parameter SV model; N=100) *LEFT TOP: State process estimated posterior mean plot with 99% credible intervals. LEFT BOTTOM: The S&P 500 data. RIGHT: The contour plot of sampled parameters σ_ω^2 v.s. ϕ.*

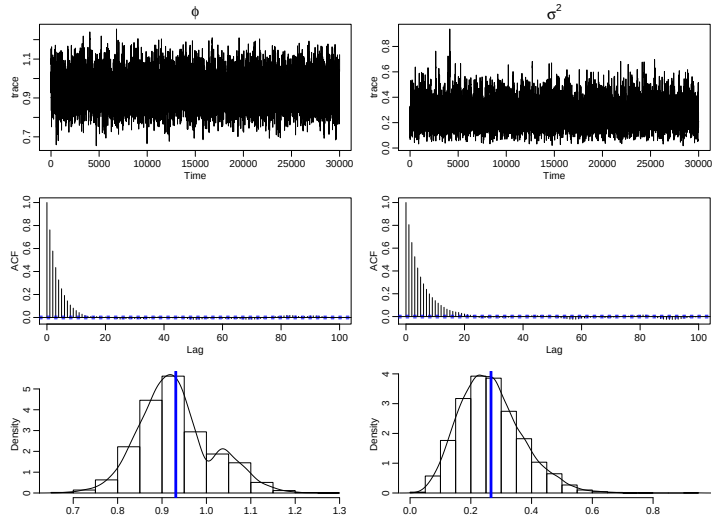

Figure B.7: (Method B; S&P 500 data; Two-parameter SV model) *Results for S&P 500 data using the two-parameter SV model. TOP: Trace plots of sampled parameters. MIDDLE: the sample ACFs of the traces. BOTTOM: Histograms of sampled parameters. The blue lines show the sample mean of parameters. In total 33,000 iterations were drawn and discarded the first 3,000 as burn-in. The particles number is $N = 200$.*

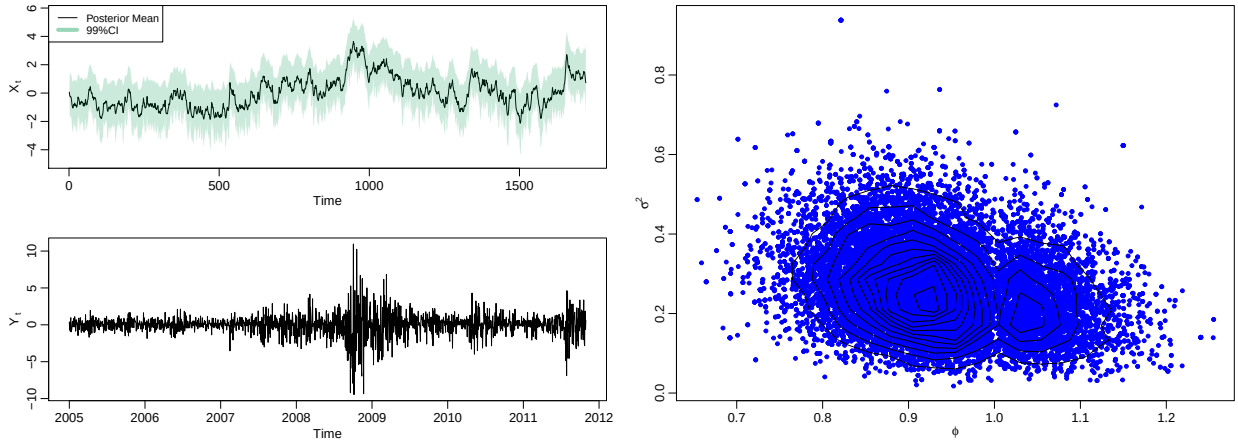

Figure B.8: (Method B; S&P 500 data; Two-parameter SV model; N=200) *LEFT TOP: State process estimated posterior mean plot with 99% credible intervals. LEFT BOTTOM: The S&P 500 data. RIGHT: The contour plot of sampled parameters σ_ω^2 v.s. ϕ.*

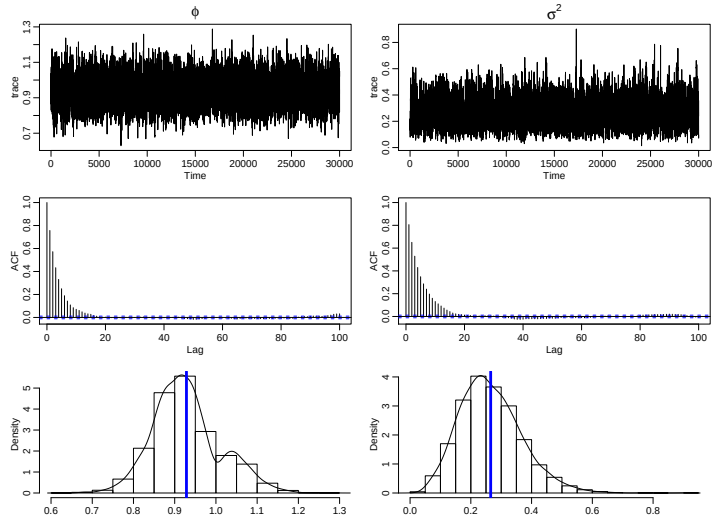

Figure B.9: (Method B; S&P 500 data; Two-parameter SV model) *Results for S&P 500 data using the two-parameter SV model. TOP: Trace plots of sampled parameters. MIDDLE: the sample ACFs of the traces. BOTTOM: Histograms of sampled parameters. The blue lines show the sample mean of parameters. In total 33,000 iterations were drawn and discarded the first 3,000 as burn-in. The particles number is $N = 500$.*

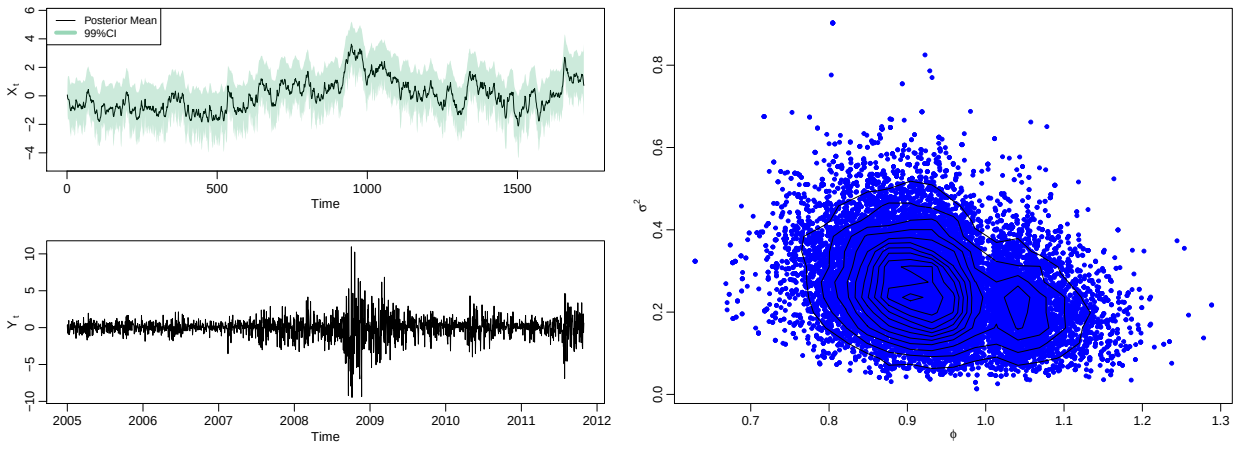

Figure B.10: (Method B; S&P 500 data; Two-parameter SV model; N=500) *LEFT TOP: State process estimated posterior mean plot with 99% credible intervals. LEFT BOTTOM: The S&P 500 data. RIGHT: The contour plot of sampled parameters* σ_ω^2 *v.s.* ϕ.

B.2 RESULTS FOR THE THREE-PARAMETER SV MODEL

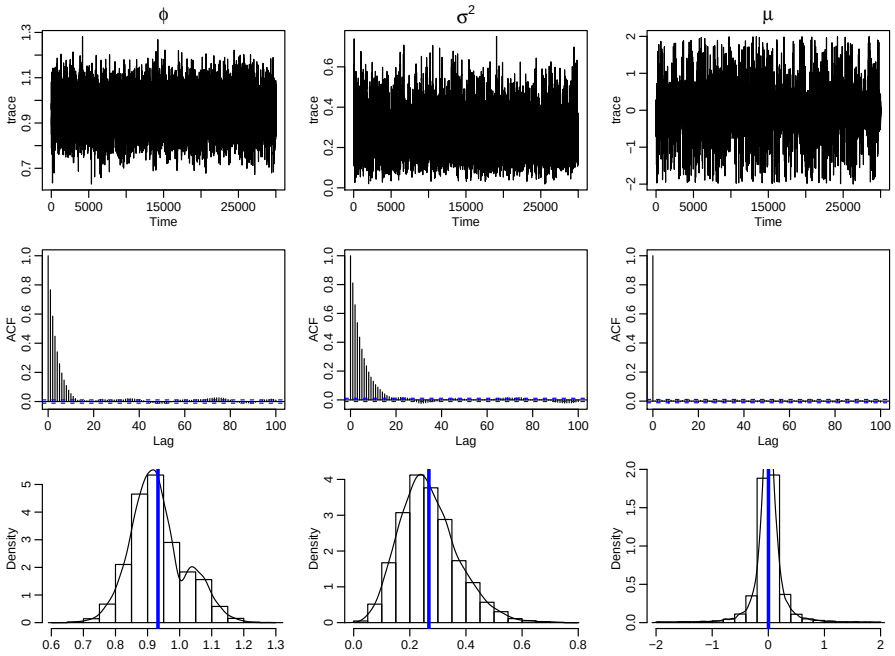

Figure B.11: (Method B; S&P 500 data; Three-parameter SV model) *Results for S&P 500 data using the three-parameter SV model. TOP: Trace plots of sampled parameters. MIDDLE: the sample ACFs of the traces. BOTTOM: Histograms of sampled parameters. The blue lines show the sample mean of parameters. In total 33,000 iterations were drawn and discarded the first 3,000 as burn-in. The particles number is* $N = 10$.

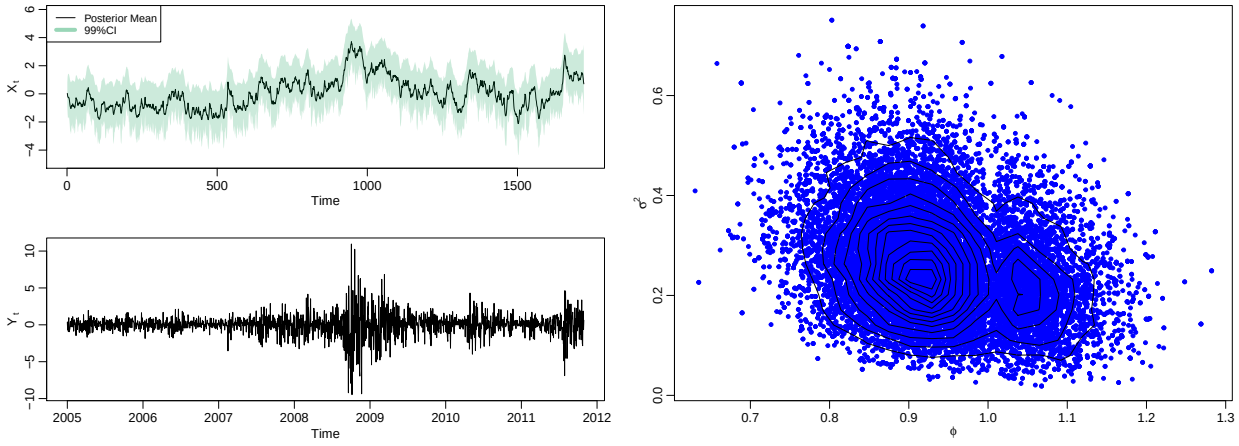

Figure B.12: (Method B; S&P 500 data; Three-parameter SV model; N=10) *LEFT TOP: State process estimated posterior mean plot with 99% credible intervals. LEFT BOTTOM: The S&P 500 data. RIGHT: The contour plot of sampled parameters σ_ω^2 v.s. ϕ.*

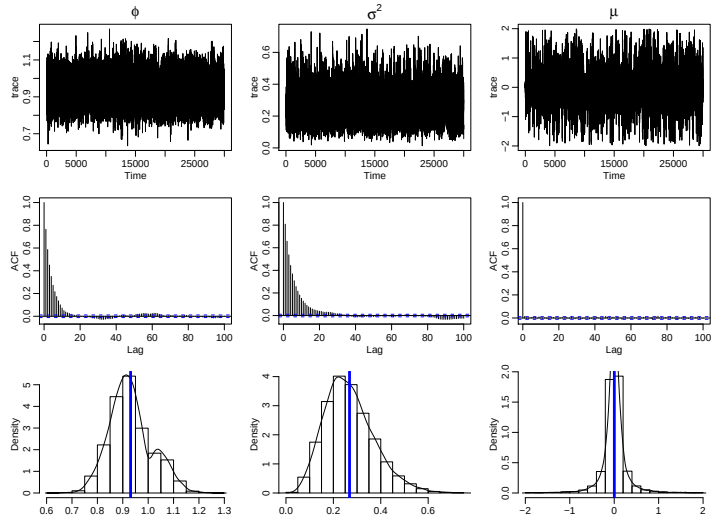

Figure B.13: (Method B; S&P 500 data; Three-parameter SV model) *Results for S&P 500 data using the three-parameter SV model. TOP: Trace plots of sampled parameters. MIDDLE: the sample ACFs of the traces. BOTTOM: Histograms of sampled parameters. The blue lines show the sample mean of parameters. In total 33,000 iterations were drawn and discarded the first 3,000 as burn-in. The particles number is $N = 50$.*

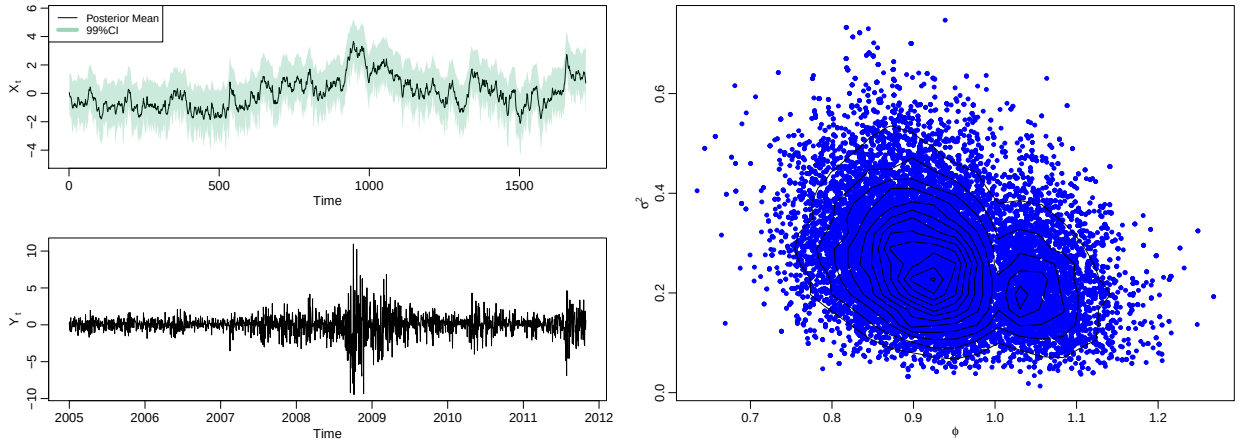

Figure B.14: (Method B; S&P 500 data; Three-parameter SV model; N=50) *LEFT TOP: State process estimated posterior mean plot with 99% credible intervals. LEFT BOTTOM: The S&P 500 data. RIGHT: The contour plot of sampled parameters σ_ω^2 v.s. ϕ.*

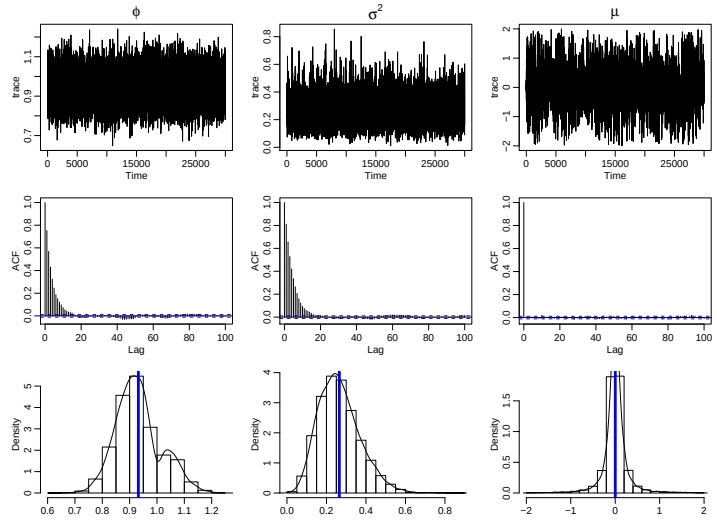

Figure B.15: (Method B; S&P 500 data; Three-parameter SV model) *Results for S&P 500 data using the three-parameter SV model. TOP: Trace plots of sampled parameters. MIDDLE: the sample ACFs of the traces. BOTTOM: Histograms of sampled parameters. The blue lines show the sample mean of parameters. In total 33,000 iterations were drawn and discarded the first 3,000 as burn-in. The particles number is $N = 100$.*

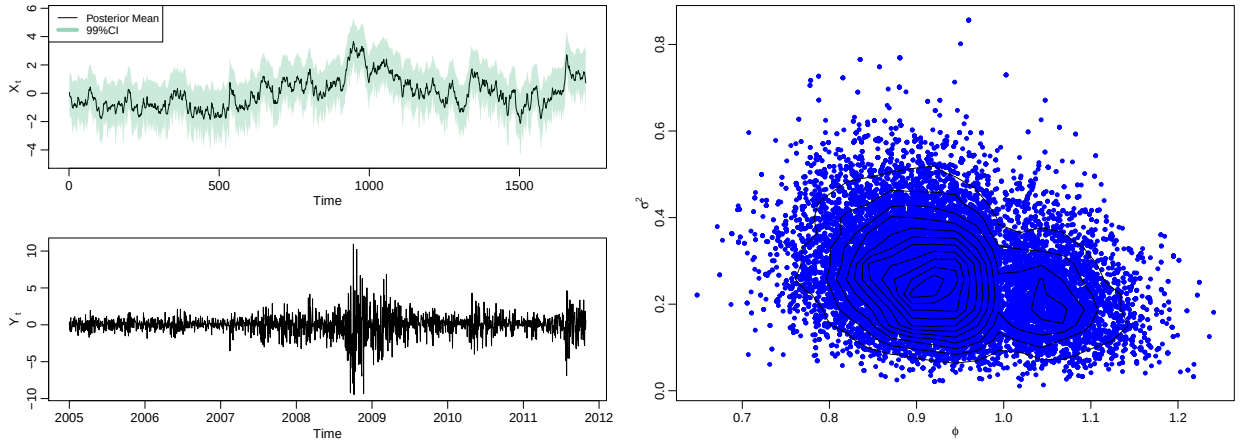

Figure B.16: (Method B; S&P 500 data; Three-parameter SV model; N=100) *LEFT TOP: State process estimated posterior mean plot with 99% credible intervals. LEFT BOTTOM: The S&P 500 data. RIGHT: The contour plot of sampled parameters σ_ω^2 v.s. ϕ.*

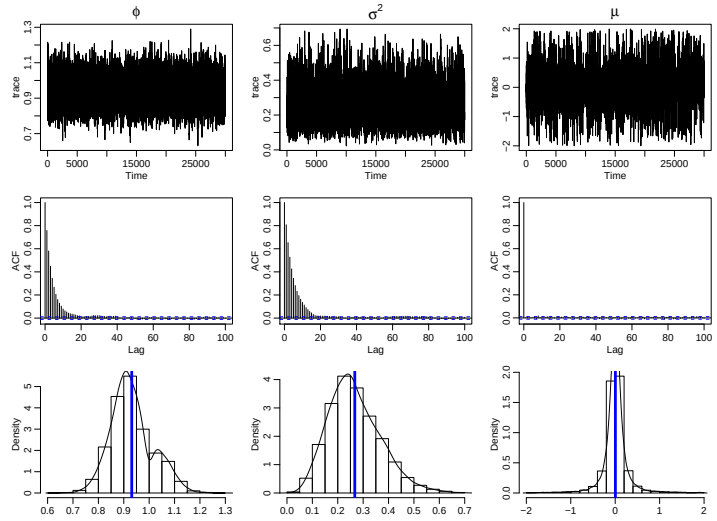

Figure B.17: (Method B; S&P 500 data; Three-parameter SV model) *Results for S&P 500 data using the three-parameter SV model. TOP: Trace plots of sampled parameters. MIDDLE: the sample ACFs of the traces. BOTTOM: Histograms of sampled parameters. The blue lines show the sample mean of parameters. In total 33,000 iterations were drawn and discarded the first 3,000 as burn-in. The particles number is $N = 200$.*

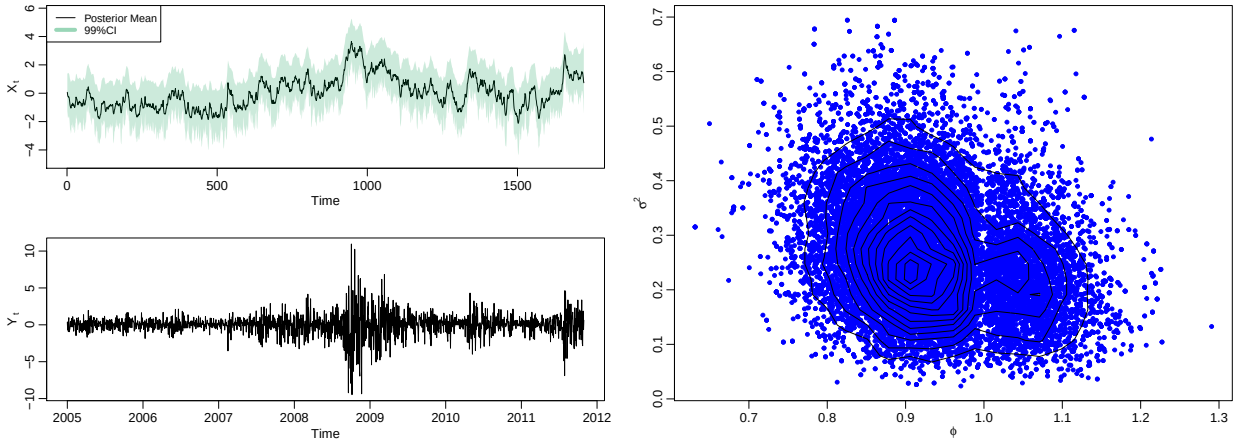

Figure B.18: (Method B; S&P 500 data; Three-parameter SV model; N=200) *LEFT TOP: State process estimated posterior mean plot with 99% credible intervals. LEFT BOTTOM: The S&P 500 data. RIGHT: The contour plot of sampled parameters σ_ω^2 v.s. ϕ.*

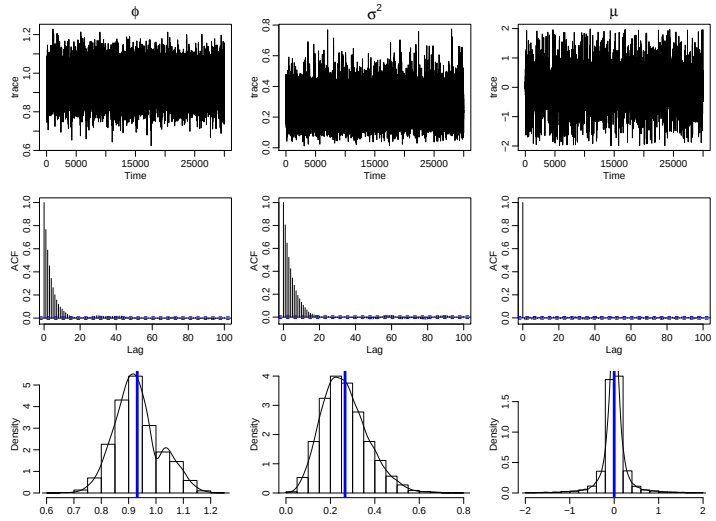

Figure B.19: (Method B; S&P 500 data; Three-parameter SV model) *Results for S&P 500 data using the three-parameter SV model. TOP: Trace plots of sampled parameters. MIDDLE: the sample ACFs of the traces. BOTTOM: Histograms of sampled parameters. The blue lines show the sample mean of parameters. In total 33,000 iterations were drawn and discarded the first 3,000 as burn-in. The particles number is $N = 500$.*

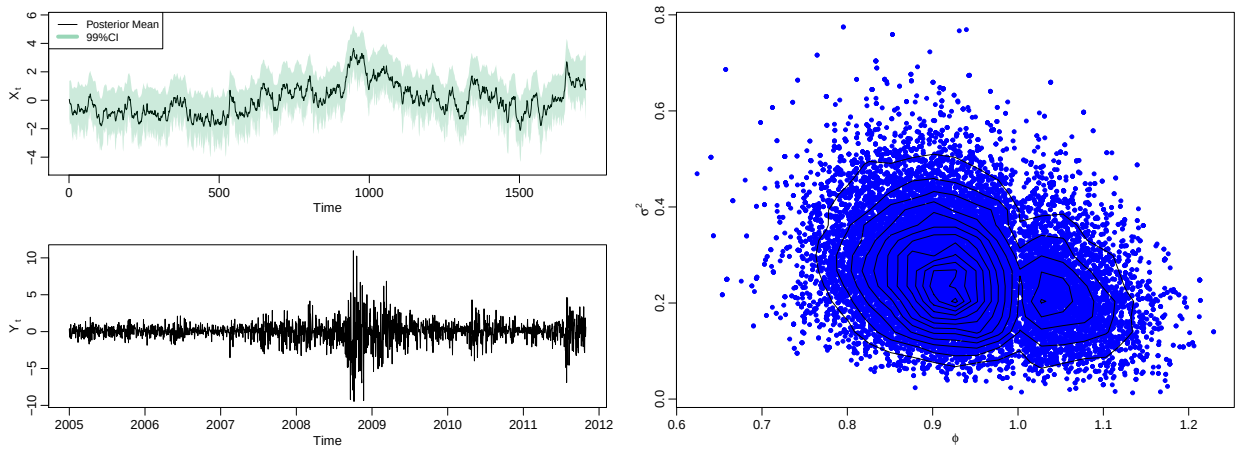

Figure B.20: (Method B; S&P 500 data; Three-parameter SV model; N=500) *LEFT TOP: State process estimated posterior mean plot with 99% credible intervals. LEFT BOTTOM: The S&P 500 data. RIGHT: The contour plot of sampled parameters σ_ω^2 v.s. ϕ.*

APPENDIX C

R CODE FOR SIMULATION STUDIES AND APPLICAITONS

C.1 CODE FOR FIGURES IN SECTION 4.1

```
library(astsa)
library(ggplot2)
library(cowplot)

# Series A and Series B
set.seed(1989)
num  = 1000; lev  =  0
beta = .1; Vsd  =  1
sig1 =  0.15; sig2 =  0.38
x1   =  arima.sim(list(order=c(1,0,0), ar=.99), sd=sig1, n=num) + lev
y1   =  beta*exp(x1/2)*rnorm(num,0,Vsd)
x2   =  arima.sim(list(order=c(1,0,0), ar=.95), sd=sig2, n=num) + lev
y2   =  beta*exp(x2/2)*rnorm(num,0,Vsd)
y1   =  as.vector(y1)
y2   =  as.vector(y2)
y = data.frame(Time=seq(1:1000), Series_A = y1, Series_B = y2)

## Figure 4.1
ggplot(data=y) +
geom_line(aes(y=Series_B, x=Time, color="Series B")) +
geom_line(aes(y=Series_A, x=Time, color="Series A")) +
labs(color="") +
theme_bw() +
ylab('%') +
theme(legend.position = c(0.1, 0.2))

numer = function(sig,phi,h) { exp(sig*phi^h)-1 }
denom = function(sig) { 3*exp(sig)-1 }

# ACFs
phi1 =.99
sigx1 = sig1^2/(1-phi1^2)
u1  = numer(sigx1,phi1,1:30)/denom(sigx1)
phi2 =.95
sigx2 = sig2^2/(1-phi2^2)
u2  = numer(sigx2,phi2,1:30)/denom(sigx2)

acf.1 = data.frame(cbind(ACF=acf(y1^2, 30, plot = FALSE)$acf[2:31]
                  , Lag=seq(1:30), acf.1=u1, acf.2=u2))
acf.2 = data.frame(cbind(ACF=acf(y2^2, 30, plot = FALSE)$acf[2:31]
```

```
                    , Lag=seq(1:30), acf.1=u1, acf.2=u2))

## Figure 4.2
a = ggplot(data = acf.1[1:30,], aes(x=Lag, y=ACF)) +
geom_bar(stat='identity',width=.1, color = "orangered3") +
ylim(-0.05,0.5) +
geom_line(aes(y=acf.1, x=Lag, color="Model I"))+
geom_line(aes(y=acf.2, x=Lag, color="Model II"))+
theme_bw()+
labs(color="") +
theme(legend.position = c(0.8, 0.85),
plot.title = element_text(colour = "orangered3", face="bold"))+
ggtitle("Squared Series A")

b = ggplot(data = acf.2[1:30,], aes(x=Lag, y=ACF)) +
geom_bar(stat='identity',width=.1, color = "forestgreen") +
ylim(-0.05,0.5) +
geom_line(aes(y=acf.1, x=Lag, color="Model I"))+
geom_line(aes(y=acf.2, x=Lag, color="Model II"))+
theme_bw()+
labs(color="") +
theme(legend.position = c(0.8, 0.85),
plot.title = element_text(colour = "forestgreen", face="bold"))+
ggtitle("Squared Series B")

plot_grid(a, b, labels = "")
```

C.2 CODE FOR METHOD A IN SECTION 5.1.1 AND SECTION 5.2.1

```
#########################################################################
#   PGAS for two-parms SV model.
# Input (nmcmc, burnin, y, prior, phi_init, q_init, npart, mcmseed)
#        source this file then use "Run PGAS for SV model" code
#    2 parms: X(t)=phi X(t-1) + W(t) ~ iid N(0,q) ;
#   Y(t)=exp{X(t)/2)}V(t) ~ iid N(0,1)
#########################################################################

pgasSV = function(nmcmc, burnin, y, prior, phi_init, q_init, npart, mcmseed,
   prior_alpha_sig, prior_beta_sig, prior_alpha_phi, prior_beta_phi){
numMCMC = nmcmc+burnin
N = npart
set.seed(mcmseed)

pr <- progress_text()         # displays progress (from plyr)
pr$init(numMCMC)

T = length(y)
q = rep(0,numMCMC)             # q = state variance
X = matrix(0,numMCMC,T)
phi = rep(0,numMCMC)           #  phi-values

# Initialize the parameters
q[1] = q_init
phi[1] = phi_init

# Initialize the state by running a PF
u = cpf_as_sv(y, phi[1], q[1], N,  X[1,])   #  changed from X to X[1,]
particles = u$x           # returned particles
w = u$w                   # returned weights
# Draw J
J = which( (runif(1)-cumsum(w[,T])) < 0 )[1]
X[1,] = scale(particles[J,], center=TRUE, scale=FALSE)

# Run MCMC loop
for(k in 2:numMCMC){
# Sample the parameters (inverse gamma posteriors)
Z = cbind(X[k-1,2:T], X[k-1,1:(T-1)])
V = prior*diag(1,2) +   t(Z)%*%Z
m = V[2,1]/V[2,2]
L = V[2,2]
S = V[1,1] - m*V[1,2]
# # Sample from NIG posterior
# q[k] = 1/rgamma(1, (T-1)/2, S/2)
# phi[k] = m + sqrt(q[k]/L)*rnorm(1)

#sample q
part1 = sum(X[k-1,1:T]^2)
part2 = 2*phi[k-1]*V[2,1]
part3 = (phi[k-1]^2)*(sum(X[k-1,2:T-1]^2))
beta_sig = prior_beta_sig + .5*(part1-part2+part3)
alpha_sig = T/2 + prior_alpha_sig
q[k] = 1/rgamma(1,alpha_sig, beta_sig)

#sample phi
part5 = sum(X[k-1,2:T-1]^2)/q[k] + 1/prior_beta_phi^2
part6 = V[2,1]/q[k] + prior_alpha_phi/prior_beta_phi^2
beta_phi = 1/part5
alpha_phi = beta_phi * part6

phi[k] = alpha_phi + sqrt(beta_phi)*rnorm(1)
# while(phi[k] >=1 | phi[k] <=-1){
#   phi[k] = alpha_phi + sqrt(beta_phi)*rnorm(1)
```

```
# }

# Run CPF-AS
u = cpf_as_sv(y, phi[k], q[k], N, X[k-1,])
particles = u$x   # returned particles
w = u$w                   # returned weight
# Draw J (extract a particle trajectory)
J = which( (runif(1)-cumsum(w[,T])) < 0 )[1]
X[k,] = scale(particles[J,], center=TRUE, scale=FALSE)      # center X
pr$step()
}#end
bi = 1:burnin
list(phi=phi[-bi], q=q[-bi],  X=X[-bi,])
}#end

#-------------------------------------------------------------------------
cpf_as_sv = function(y, phi, q, N, X){
# Conditional particle filter with ancestor sampling
# Input:
#   y - measurements
#   phi - transition parameter
#   q - state noise variance
#   N - number of particles
#   X - conditioned particles
T = length(y)
x = matrix(0, N, T); # Particles
a = matrix(0, N, T); # Ancestor indices
w = matrix(0, N, T); # Weights
x[,1] = 0; # Deterministic initial condition
x[N,1] = X[1]

for (t in 1:T){
if(t != 1){
ind = resamplew(w[,t-1]);
ind = ind[sample.int(N)];
t1 = t-1
xpred = phi*x[, t1]
x[,t] = xpred[ind] + sqrt(q)*rnorm(N);
x[N,t] = X[t];
# Ancestor sampling
m = exp(-1/(2*q)*(X[t]-xpred)^2);
w_as = w[,t-1]*m
w_as = w_as/sum(w_as);
ind[N] =  which( (runif(1)-cumsum(w_as)) < 0 )[1]
# Store the ancestor indices
a[,t] = ind;
}#end
# Compute importance weights
var_now = exp(x[,t])
logweights = -y[t]^2/(2*var_now) - 1/2*log(var_now)  # (up to an additive constant)
const = max(logweights); # Subtract the maximum value for numerical stability
weights = exp(logweights-const)
w[,t] = weights/sum(weights) # Save the normalized weights
}#end

# Generate the trajectories from ancestor indices
ind = a[,T];
for(t in (T-1):1){
x[,t] = x[ind,t];
ind = a[ind,t];
}#end
list(x=x, w=w)
}#end
#-------------------------------------------------------------------
resamplew = function(w){  # multinomial resampling
N = length(w); u = runif(N)
cw = cumsum(w); cw = cw/cw[N]
ucw = c(u,cw)
```

```
ind1 = sort(ucw, index.return=TRUE)$ix
ind2 = which(ind1<=N)
i = ind2-(0:(N-1))
return(i)
}
```

```
#########################################################
## Simulation for SV Model using PGAS                  ##
##    2 parms SV model                                 ##
##    Run pgasSV(), cpf_as_sv(), resamplew() first     ##
#########################################################
library(nltsa)
require(plyr)
# Generate data
set.seed(1989)
phi_sim = 0.9; q_sim = 0.2; mu_sim = 0
n = 2000
h_sim = rep(0, n); y_sim = rep(0, n)
h_sim[1] = mu_sim + sqrt(q_sim/(1-phi_sim^2))*rnorm(1)

for(i in 1:2000){
  y_sim[i] = exp(mu_sim/2)*exp(h_sim[i]/2)*rnorm(1)
  h_sim[i+1] = mu_sim + phi_sim*(h_sim[i]-mu_sim) + sqrt(q_sim)*rnorm(1)
}
y_sim = y_sim[1001:2000]
plot.ts(y_sim)

npart   = 20        # Number of particles used in pgas
nmcmc   = 30000     # Number of iterations in the mcmc samplers after burnin
burnin  = 3000      # Number of iterations to burn
mcmseed = 90210

#  Prior and initial values of phi and q
prior = .1
phi_init=.95
q_init= .25

prior_alpha_sig = 5
prior_beta_sig = 0.02
prior_alpha_phi = 0
prior_beta_phi = 1

# Run it
ptm <- proc.time()
u = pgasSV(nmcmc, burnin,  y_sim, prior, phi_init, q_init, npart, mcmseed,
prior_alpha_sig, prior_beta_sig, prior_alpha_phi, prior_beta_phi)
(time2run = proc.time() - ptm)

#################### parallel ######################
require(parallel)
require(doParallel)
require(foreach)
c1 = makeCluster(3)
registerDoParallel(c1)
out.all = foreach(npart=c(10,20,50,500,200,100)
    ,.packages=c("plyr","MASS"))%dopar%{
u_10d1 = pgasSV(nmcmc, burnin,  y_sim, prior, phi_init, q_init
    , npart, mcmseed, prior_alpha_sig, prior_beta_sig
    , prior_alpha_phi, prior_beta_phi)
return(u_10d1)
}
stopCluster(c1)

```

```
#######################################
# Pretty pictures
parms = cbind(u$phi, u$q)
names = c(expression(phi), expression(sigma^2))
true_parms = c(phi_sim, q_sim)
nobs=length(y_sim)

#dev.new(height=5.5, width=9)
par(mfcol=c(3,2), mar=c(3,3,1,1), mgp=c(1.6,.6,0), oma=c(0,0,1.5,0))
# Parameters ...
for (i in 1:2){
  plot.ts(parms[,i], ylab='trace')
  mtext(names[i], 3, line=.25)
  acf(parms[,i], 200, xlim=c(1,200))
  hist(parms[,i], breaks=20,  prob=TRUE, main="", xlab="")
  lines(density(parms[,i], adjust=1.5))
  lp = quantile(parms[,i], .005)
  up = quantile(parms[,i], .995)
  abline(v=mean(parms[,i]), col=4, lwd=3, lty=1)
  abline(v=true_parms[i], col=2, lwd=1, lty=1)
  print(mean(parms[,i]))
}

# and States ...
#dev.new(height=6, width=9)
par(mfrow=c(2,1), mar=c(3,3,1.5,1), mgp=c(1.6,.6,0), oma=c(0,0,0,0))
mX = ts(apply(u$X, 2, mean), start=start(y_sim), freq=frequency(y_sim))
lX = ts(apply(u$X, 2, quantile, 0.005), start=start(y_sim), freq=frequency(y_sim))
uX = ts(apply(u$X, 2, quantile, 0.995), start=start(y_sim), freq=frequency(y_sim))
plot.ts(mX, ylab=expression(X[~t]), ylim=c(min(lX,mX)-.5,max(uX,mX)+.5))
xx=c(time(lX), rev(time(uX)))
yy=c(lX, rev(uX))
polygon(xx, yy, border=NA, col=rgb(0,.6,.3, alpha = .2))
legend("topleft",legend=c("Posterior Mean","99%CI")
      ,col=c(1,rgb(0,.6,.3, alpha = .4)), lty=c(1,1), lwd=c(1,5), cex=.85)
plot.ts(y_sim/100, ylab=expression(Y[~t]))

#dev.new()
library(MASS)
par(mfrow=c(1,1), mar=c(3,3,1.5,1), mgp=c(1.6,.6,0), oma=c(0,0,0,0))
z=kde2d(parms[,1],parms[,2], h=c(.02,.05))
plot(parms[,1],parms[,2], pch=20, col=4
     , xlab=expression(phi), ylab=expression(sigma^2))
contour(z, drawlabels=FALSE, nlevels=15,  add=TRUE)

# Inefficiency
ineff = function(parm){
  parmacf = acf(parm, nmcmc, plot = F)$acf
  inef = 0
  for(i in 1:(nmcmc/2)){
    temp1 = parmacf[2*i-1]+parmacf[2*i]
    if(temp1 > 0){inef = inef + temp1}else{break}
  }
  inef = 2*inef - parmacf[1]
  return(inef)
}
ineff(parms[,1])
ineff(parms[,2])
```

```
#########################################################
## SP500 for SV Model using PGAS                       ##
##    2 parms SV model (individual sampling)           ##
##    Run pgasSV(), cpf_as_sv(), resamplew() first     ##
```

```
#######################################################
#install.packages("devtools")
#devtools::install_github("nickpoison/nltsa")
library(nltsa)
data(sp500.gr)
require(plyr)

npart   = 20        # Number of particles used in pgas
nmcmc   = 30000     # Number of iterations in the mcmc samplers after burnin
burnin  = 3000      # Number of iterations to burn
mcmseed = 90210

y = 100*window(sp500.gr, start=2005)  # data in %
plot.ts(y)

#  Prior and initial values of phi and q
prior = .1
phi_init=.95
q_init= .25
prior_alpha_sig = 5
prior_beta_sig = 0.02
prior_alpha_phi = 0
prior_beta_phi = 1

# Run it
ptm <- proc.time()
u = pgasSV(nmcmc, burnin,  y, prior, phi_init, q_init, npart, mcmseed,
prior_alpha_sig, prior_beta_sig, prior_alpha_phi, prior_beta_phi)
(time2run = proc.time() - ptm)

##################### parallel ########################
require(parallel)
require(doParallel)
require(foreach)
c1 = makeCluster(3)
registerDoParallel(c1)
out.all = foreach(npart=c(10,20,50,500,200,100)
  , .packages=c("plyr","MASS"))%dopar%{
u_d = pgasSV(nmcmc, burnin,  y, prior, phi_init, q_init, npart, mcmseed,
  prior_alpha_sig, prior_beta_sig, prior_alpha_phi, prior_beta_phi)
return(u_d)
}
stopCluster(c1)

#######################################
# Pretty pictures
parms = cbind(u$phi, u$q)
names = c(expression(phi), expression(sigma^2))
nobs=length(y)

#dev.new(height=5.5, width=9)
par(mfcol=c(3,2), mar=c(3,3,1,1), mgp=c(1.6,.6,0), oma=c(0,0,1.5,0))

# Parameters ...
for (i in 1:2){
  plot.ts(parms[,i], ylab='trace')
  mtext(names[i], 3, line=.25)
  acf(parms[,i], 100, xlim=c(1.5,100))
  hist(parms[,i], breaks=20,  prob=TRUE, main="", xlab="")
  lines(density(parms[,i], adjust=1.5))
  lp = quantile(parms[,i], .005)
  up = quantile(parms[,i], .995)
  abline(v=mean(parms[,i]), col=4, lwd=3, lty=1)
  print(mean(parms[,i]))
}

# and States ...
#dev.new(height=6, width=9)
```

```
par(mfrow=c(2,1), mar=c(3,3,1.5,1), mgp=c(1.6,.6,0), oma=c(0,0,0,0))
mX = ts(apply(u$X, 2, mean), start=start(y), freq=frequency(y))
lX = ts(apply(u$X, 2, quantile, 0.005), start=start(y), freq=frequency(y))
uX = ts(apply(u$X, 2, quantile, 0.995), start=start(y), freq=frequency(y))
plot.ts(mX, ylab=expression(X[~t]), ylim=c(min(lX,mX)-.5,max(uX,mX)+.5))
xx=c(time(lX), rev(time(uX)))
yy=c(lX, rev(uX))
polygon(xx, yy, border=NA, col=rgb(0,.6,.3, alpha = .2))
legend("topleft",legend=c("Posterior Mean","99%CI"),col=c(1,rgb(0,.6,.3, alpha = .4)),
lty=c(1,1), lwd=c(1,5), cex=.85)
plot(y/100, ylab=expression(Y[~t]))

#dev.new()
library(MASS)
z=kde2d(parms[,1],sqrt(parms[,2]), h=c(.02,.05))
plot(parms[,1],sqrt(parms[,2]), pch=20, col=4, xlab='phi', ylab='sigma')
contour(z, drawlabels=FALSE, nlevels=15,  add=TRUE)

# Ineff
ineff = function(parm){
  parmacf = acf(parm, nmcmc, plot = F)$acf
  inef = 0
  for(i in 1:(nmcmc/2)){
    temp1 = parmacf[2*i-1]+parmacf[2*i]
    if(temp1 > 0){inef = inef + temp1}else{break}
  }
  inef = 2*inef - parmacf[1]
  return(inef)
}
ineff(parms[,1])
ineff(parms[,2])
```

C.3 CODE FOR METHOD B IN SECTION 5.1.1 AND SECTION 5.2.1

```
##############################################################################
#   PGAS for SV model.  (Joint Parameters Sampling)
#       source this file for SV model
#   3 parameters: X(t) = mu + phi (X(t-1)-mu) + W(t)~iid N(0,q);
#     Y(t) = beta*exp{X(t)/2)}V(t)~iid N(0,1);
#     beta = exp(mu/2)
#   2 parameters: X(t) = phi*X(t-1) + W(t) ~ iid N(0,q);
#     Y(t) = beta*exp{X(t)/2)}V(t)~iid N(0,1);
#     beta = exp(mu/2)
##############################################################################

pgasSV_binorm = function(nmcmc, burnin, y, init, hyper, sigma_MH,
         npart, parms_2 = T, mcmseed){
# Input:
#   nmcmc - number of MCMC
#   burnin - number of burnin
#   y - measurements
#   init - initial value of (phi, q, mu)
#   npart - number of particles
#   hyper - hyperparameters for bivariate normal (phi, q),
#   user inputs (mu_phi, mu_q, sigma_phi, sigma_q, rho)
#   sigma_MH - covariance matrix for Random Walk Metropolis Hastings
#   parms_2 - IF (parms_2=TRUE) THEN (2 parms SV model)
#       ELSE (3 parms SV model)
#   mcmseed - seed for mcmc
# Output:
#   phi - sampled phi
#   q - sampled phi
#   mu - sampled mu
#   X - sampled hidden states
#   time - running time
#   acp - acceptence rate of Random Walk Metropolis Hastings

ptm <- proc.time()
numMCMC = nmcmc+burnin
N = npart
set.seed(mcmseed)

pr <- progress_text()         # displays progress (from plyr)
pr$init(numMCMC)

T = length(y)
X = matrix(0,numMCMC,T)
q = rep(0,numMCMC)             # q = state variance ## W(t) ~ iid N(0,q)
phi = rep(0,numMCMC)           # phi-values
mu = rep(0,numMCMC)            # leverage term

# Initialize the parameters
phi[1] = init[1]
q[1] = init[2]
mu[1] = init[3]

# hyperparameters
mu_phi = hyper[1]
mu_q = hyper[2]
sigma_phi = hyper[3]
sigma_q = hyper[4]
rho = hyper[5]  # negative correlation
mu_MH = c(0,0)  # tuning parameter
sigma_MH = sigma_MH # tuning parameter

# Initialize the state by running a PF
if (parms_2){
```

```
  u = cpf_as_sv(y, phi[1], q[1], 0, N,  X[1,])
  particles = u$x           # returned particles
  w = u$w                   # returned weights
  # Draw J
  J = which( (runif(1)-cumsum(w[,T])) < 0 )[1]
  X[1,] = scale(particles[J,], center=TRUE, scale=FALSE)
} else {
  u = cpf_as_sv(y, phi[1], q[1], mu[1], N,  X[1,])
  particles = u$x           # returned particles
  w = u$w                   # returned weights
  # Draw J
  J = which( (runif(1)-cumsum(w[,T])) < 0 )[1]
  X[1,] = scale(particles[J,], center=TRUE, scale=FALSE)
}

# Run MCMC loop
if (parms_2){

  for(k in 2:numMCMC){
  # Sample the parameters (phi, sqrt_q) ~ bivariate normal RWMH
  parms = cbind(phi[k-1], sqrt(q[k-1]))
  (parms_star = parms + mvrnorm(1, mu_MH, sigma_MH))

  while(parms_star[2]^2 < 1e-2|parms_star[1]>2){
  (parms_star = parms + mvrnorm(1, mu_MH, sigma_MH))
  }

  g_star = log_g_func(parms_star, mu_phi, sigma_phi, mu_q
        , sigma_q, rho, X[(k-1),], 0)
  g_old = log_g_func(parms, mu_phi, sigma_phi, mu_q
        , sigma_q, rho, X[(k-1),], 0)
  g_diff = g_star - g_old
  if(is.na(g_diff)){
  phi[k] = parms[1]
  q[k] = parms[2]^2
  }else {
  if(log(runif(1)) < g_diff){
  phi[k] = parms_star[1]
  q[k] = parms_star[2]^2
  }else{
  phi[k] = parms[1]
  q[k] = parms[2]^2
  }
  }

  # Run CPF-AS
  u1 = cpf_as_sv(y, phi[k], q[k], 0, N, X[k-1,])
  particles = u1$x           # returned particles
  w = u1$w                   # returned weight
  # Draw J (extract a particle trajectory)
  J = which( (runif(1)-cumsum(w[,T])) < 0 )[1]
  X[k,] = scale(particles[J,], center=TRUE, scale=FALSE) # center X

  pr$step()
  }

} else {

  for(k in 2:numMCMC){
  # Sample the parameters (phi, sqrt_q) ~ bivariate normal, RWMH
  parms = cbind(phi[k-1], sqrt(q[k-1]))
  (parms_star = parms + mvrnorm(1, mu_MH, sigma_MH))

  while(parms_star[2]^2 < 1e-2|parms_star[1]>2){
  (parms_star = parms + mvrnorm(1, mu_MH, sigma_MH))
  }

  g_star = log_g_func(parms_star, mu_phi, sigma_phi, mu_q
```

```
      , sigma_q, rho, X[(k-1),], mu[k-1])
  g_old = log_g_func(parms, mu_phi, sigma_phi, mu_q
      , sigma_q, rho, X[(k-1),], mu[k-1])
  g_diff = g_star - g_old
  if(is.na(g_diff)){
  phi[k] = parms[1]
  q[k] = parms[2]^2
  }else {
  if(log(runif(1)) < g_diff){
  phi[k] = parms_star[1]
  q[k] = parms_star[2]^2
  }else{
  phi[k] = parms[1]
  q[k] = parms[2]^2
  }
  }

  # Sample mu
  sigmu = q[k]/((T-1)*(1-phi[k])^2+(1-phi[k]^2))
  muhat = sigmu*((1-phi[k]^2)*X[k-1,1]/q[k]+(1-phi[k])*
    (sum(X[k-1,2:T])-phi[k]*sum(X[k-1,1:(T-1)]))/q[k])
  mu[k] = muhat + sqrt(abs(sigmu))*rnorm(1)
  while(abs(mu[k]) > 2){
  mu[k] = muhat + sqrt(abs(sigmu))*rnorm(1)
  }

  # Run CPF-AS
  u = cpf_as_sv(y, phi[k], q[k], mu[k], N, X[k-1,])
  particles = u$x          # returned particles
  w = u$w                  # returned weight
  # Draw J (extract a particle trajectory)
  J = which( (runif(1)-cumsum(w[,T])) < 0 )[1]
  X[k,] = scale(particles[J,], center=TRUE, scale=FALSE) # center X
  pr$step()
  }#end

}

bi = 1:burnin
time2run = proc.time() - ptm
acp = dim(table(phi[-bi]))/nmcmc*100
list(phi=phi[-bi], q=q[-bi], mu=mu[-bi], X=X[-bi,], time=time2run, acp=acp)
}#end

#-------------------------------------------------------------------------
cpf_as_sv = function(y, phi, q, mu, N, X){
# Conditional particle filter with ancestor sampling
# Input:
#   y - measurements
#   phi - transition parameter
#   q - state noise variance
#   mu - leverage term
#   N - number of particles
#   X - conditioned particles

  T = length(y)
  x = matrix(0, N, T); # Particles
  a = matrix(0, N, T); # Ancestor indices
  w = matrix(0, N, T); # Weights
  x[,1] = 0; # Deterministic initial condition
  x[N,1] = X[1]

  for (t in 1:T){
  if(t != 1){
  ind = resamplew(w[,t-1]);
  ind = ind[sample.int(N)]; # default: no replacement
  t1 = t-1
  xpred = phi*x[, t1] + (1-phi)*mu # length N Vector
```

```
  x[,t] = xpred[ind] + sqrt(q)*rnorm(N);
  x[N,t] = X[t]; #Line 6

  # Ancestor sampling

  logm = -1/(2*q)*(X[t]-xpred)^2;
  maxlogm = max(log(w[,t-1])+logm)
  w_as = exp(log(w[,t-1])+logm - maxlogm)
  w_as = w_as/sum(w_as);
  ind[N] =  which( (runif(1)-cumsum(w_as)) < 0 )[1]

  # Store the ancestor indices
  a[,t] = ind;
  }#end IF
  # Compute importance weights
  var_now = exp(x[,t]+mu)
  logweights = -y[t]^2/(2*var_now) - 1/2*log(var_now)
  const = max(logweights); #for numerical stability
  weights = exp(logweights-const)
  w[,t] = weights/sum(weights) # Save the normalized weights
  }#end FOR

  # Generate the trajectories from ancestor indices
  ind = a[,T];
  for(t in (T-1):1){
  x[,t] = x[ind,t];
  ind = a[ind,t];
  }#end
  list(x=x, w=w)
}#end
#-------------------------------------------------------------------
resamplew = function(w){
# multinomial resampling
  N = length(w)
  u = runif(N)    # uniform random number
  cw = cumsum(w)  # Cumulative Sum
  cw = cw/cw[N]
  ucw = c(u,cw)
  ind1 = sort(ucw, index.return=TRUE)$ix   # $ix will give the index
  ind2 = which(ind1<=N)
  i = ind2-(0:(N-1))
  return(i)
}

#-------------------------------------------------------------------
log_g_func = function(parms, mu_phi, sigma_phi, mu_q, sigma_q, rho, x, mu){
# Calculate acceptance probability for RWMH
  phi = parms[1]
  q = parms[2]
  Z = cbind(x[2:T]-mu, x[1:(T-1)]-mu)  # (T-1)*2 matrix
  V = t(Z)%*%Z
  term1 = -((phi-mu_phi)^2/(sigma_phi^2) + (q-mu_q)^2/(sigma_q^2) -
    2*rho*(phi-mu_phi)*(q-mu_q)/(sigma_q*sigma_phi))/(2*(1-rho^2))
  term2 = -(1-phi^2)*(x[1]-mu)^2/(2*q^2) - V[1,1]/(2*q^2) -
    V[2,2]*(phi^2)/(2*q^2) + V[1,2]*phi/(q^2)
  g = .25*(log((1-phi^2)^2)) - .5*T*log((q)^2) + term1 + term2
  return(g)
}
```

```
#################################################################
##      Simulation of  2-parm SV Model using PGAS               ##
##                  2 parameter model (set mu=0)                ##
##  Please run pgasSV_binorm(), cpf_as_sv(), resamplew()        ##
##    and log_g_func() first                                    ##
#################################################################
library(nltsa)
require(plyr)
library(MASS)

# Generate data
set.seed(1989)
phi_sim = 0.85; q_sim = 0.2; n = 2000
h_sim = rep(0, n); y_sim = rep(0, n)
h_sim[1] = sqrt(q_sim/(1-phi_sim^2))*rnorm(1)
for(i in 1:2000){
  y_sim[i] = exp(h_sim[i]/2)*rnorm(1)
  h_sim[i+1] = phi_sim*h_sim[i] + sqrt(q_sim)*rnorm(1)
}
y_sim = y_sim[-1:-1000]
plot.ts(y_sim)
#-----------------------------------------------------------------
npart   = 20       # Number of particles used in pgas
nmcmc   = 30000     # Number of iterations in the mcmc samplers after burnin
burnin  = 3000      # Number of iterations to burn
mcmseed = 90210

#  Bi-normal prior and initial values
init=c(.95, .15, 0)
hyper = c(0.9, 0.27, 0.05, 0.1, -0.25)
sigma_MH = 1.7*10^(-4) * matrix(c(100,-25,-25,100),nrow=2,ncol=2)

# N=20 Run it
ptm <- proc.time()
u1 = pgasSV_binorm(nmcmc, burnin, y_sim, init, hyper, sigma_MH,
    npart, parms_2 = T, mcmseed)
(time2run = proc.time() - ptm)

# Acceptance Rate
cat("The acceptance rate is", u1$acp, "%.")

############### parallel  ########################
require(parallel)
require(doParallel)
require(foreach)
c1 = makeCluster(3)
registerDoParallel(c1)
out.all = foreach(npart=c(10,20,50,500,200,100),
      .packages=c("plyr","MASS"))%dopar%{
  u_d = pgasSV_binorm(nmcmc, burnin, y_sim, init, hyper, sigma_MH,
      npart, parms_2 = T, mcmseed)
  u_d
}
stopCluster(c1)

############### u1  N=20  ########################
### Pretty pictures
parms = cbind(u1$phi, u1$q)
names = c(expression(phi), expression(sigma^2))
nobs=length(y_sim)
#dev.new(height=5.5, width=9)
par(mfcol=c(3,2), mar=c(3,3,1,1), mgp=c(1.6,.6,0), oma=c(0,0,1.5,0))

# Parameters ...
for (i in 1:2){
  plot.ts(parms[,i], ylab='trace')
  mtext(names[i], 3, line=.25)
```

```
  acf(parms[,i], 100, xlim=c(1.4,100))
  hist(parms[,i], breaks=20,  prob=TRUE, main="", xlab="")
  lines(density(parms[,i], adjust=1.5))
  lp = quantile(parms[,i], .005)
  up = quantile(parms[,i], .995)
  abline(v=mean(parms[,i]), col=4, lwd=3, lty=1)
}

# States ...
#dev.new(height=6, width=9)
par(mfrow=c(2,1), mar=c(3,3,1.5,1), mgp=c(1.6,.6,0), oma=c(0,0,0,0))
mX = ts(apply(u1$X, 2, mean))
lX = ts(apply(u1$X, 2, quantile, 0.005))
uX = ts(apply(u1$X, 2, quantile, 0.995))
plot.ts(mX, ylab=expression(X[~t]), ylim=c(min(lX,mX)-.5,max(uX,mX)+.5))
xx=c(time(lX), rev(time(uX)))
yy=c(lX, rev(uX))
polygon(xx, yy, border=NA, col=rgb(0,.6,.3, alpha = .2))
legend("topleft",legend=c("Posterior Mean","99%CI"),col=c(1,rgb(0,.6,.3, alpha = .4)),
lty=c(1,1), lwd=c(1,5), cex=.85)
plot.ts(y_sim, ylab=expression(Y[~t]))


#dev.new()
par(mfrow=c(1,1))
z=kde2d(parms[,1],(parms[,2]))
plot(parms[,1],(parms[,2]), pch=20, col=4
     , xlab=expression(phi), ylab=expression(sigma^2))
contour(z, drawlabels=FALSE, nlevels=15,  add=TRUE)

# Ineff
ineff = function(parm){
parmacf = acf(parm, nmcmc, plot = F)$acf
inef = 0
for(i in 1:(nmcmc/2)){
temp1 = parmacf[2*i-1]+parmacf[2*i]
if(temp1 > 0){inef = inef + temp1}else{break}
}
inef = 2*inef - parmacf[1]
return(inef)
}
ineff(parms[,1])
ineff(parms[,2])
```

```
##########################################################################
##              Simulation for 3-parms SV Model using PGAS              ##
##                  3 parameter model                                   ##
##    Please run pgasSV_binorm(), cpf_as_sv(), resamplew()            ##
##      and log_g_func() first                                 ##
##########################################################################
library(nltsa)
require(plyr)
library(MASS)

# Generate data
set.seed(90210)
phi_sim = 0.85; q_sim = 0.2; mu_sim = 0.1
n = 1000
h_sim = rep(0, n)
y_sim = rep(0, n)
h_sim[1] = mu_sim + sqrt(q_sim/(1-phi_sim^2))*rnorm(1)
for(i in 1:1000){
y_sim[i] = exp(mu_sim/2)*exp(h_sim[i]/2)*rnorm(1)
```

```
h_sim[i+1] = mu_sim + phi_sim*(h_sim[i]-mu_sim) + sqrt(q_sim)*rnorm(1)
}
plot.ts(y_sim)
#-----------------------------------------------------------------
npart   = 20        # Number of particles used in pgas
nmcmc   = 30000     # Number of iterations in the mcmc samplers after burnin
burnin  = 3000      # Number of iterations to burn
mcmseed = 90210

#  Bi-normal prior and initial values
init=c(0.95, 0.25, 0.15)
hyper = c(0.875, 0.45, 0.075, 0.1, -0.25)
sigma_MH = 1.7*10^(-4) * matrix(c(100,-25,-25,100),nrow=2,ncol=2)

# N=20 Run it
ptm <- proc.time()
u1 = pgasSV_binorm(nmcmc, burnin, y_sim, init, hyper, sigma_MH, npart
      , parms_2 = F, mcmseed)
(time2run = proc.time() - ptm)

# Acceptance Rate
cat("The acceptance rate is", u1$acp, "%.")

############### parallel  ########################
require(parallel)
require(doParallel)
require(foreach)
c1 = makeCluster(3)
registerDoParallel(c1)
out.all = foreach(npart=c(10,20,50,500,200,100),
    .packages=c("plyr","MASS"))%dopar%{
  u = pgasSV_binorm(nmcmc, burnin, y_sim, init, hyper, sigma_MH, npart
      , parms_2 = F, mcmseed)
u
}
stopCluster(c1)

################  u1 N=20 ######################
### Pretty pictures
parms = cbind(u1$phi, u1$q, u1$mu)
names = c(expression(phi), expression(sigma^2), expression(mu))
nobs=length(y_sim)

#dev.new(height=5.5, width=9)
par(mfcol=c(3,3), mar=c(3,3,1,1), mgp=c(1.6,.6,0), oma=c(0,0,1.5,0))
# Parameters ...
for (i in 1:3){
plot.ts(parms[,i], ylab='trace')
mtext(names[i], 3, line=.25)
acf(parms[,i], 100, xlim=c(1.4,100))
hist(parms[,i], breaks=20,  prob=TRUE, main="", xlab="")
lines(density(parms[,i], adjust=1.5))
lp = quantile(parms[,i], .005)
up = quantile(parms[,i], .995)
abline(v=mean(parms[,i]), col=4, lwd=3, lty=1)
}

# States ...
#dev.new(height=6, width=9)
par(mfrow=c(2,1), mar=c(3,3,1.5,1), mgp=c(1.6,.6,0), oma=c(0,0,0,0))
mX = ts(apply(u1$X, 2, mean))
lX = ts(apply(u1$X, 2, quantile, 0.005))
uX = ts(apply(u1$X, 2, quantile, 0.995))
plot.ts(mX, ylab=expression(X[~t]), ylim=c(min(lX,mX)-.5,max(uX,mX)+.5))
xx=c(time(lX), rev(time(uX)))
yy=c(lX, rev(uX))
polygon(xx, yy, border=NA, col=rgb(0,.6,.3, alpha = .2))
legend("topleft",legend=c("Posterior Mean","99%CI"),col=c(1,rgb(0,.6,.3, alpha = .4)),
```

```
lty=c(1,1), lwd=c(1,5), cex=.85)
plot.ts(y_sim, ylab=expression(Y[~t]))

#dev.new()
par(mfrow=c(1,1))
z=kde2d(parms[,1],(parms[,2]))
plot(parms[,1],(parms[,2]), pch=20, col=4
     , xlab=expression(phi), ylab=expression(sigma^2))
contour(z, drawlabels=FALSE, nlevels=15,  add=TRUE)

# Ineff
ineff = function(parm){
parmacf = acf(parm, nmcmc, plot = F)$acf
inef = 0
for(i in 1:(nmcmc/2)){
temp1 = parmacf[2*i-1]+parmacf[2*i]
if(temp1 > 0){inef = inef + temp1}else{break}
}
inef = 2*inef - parmacf[1]
return(inef)
}
ineff(parms[,1])
ineff(parms[,2])
ineff(parms[,3])
```

```
##########################################################################
##       Application on S&P500 using 2-parm SV Model                    ##
##            2 parms model (set mu=0)                                  ##
##       Please run pgasSV_binorm(), cpf_as_sv(), resamplew()           ##
##             and log_g_func() first                                   ##
##########################################################################
#install.packages("devtools")
#devtools::install_github("nickpoison/nltsa")
library(nltsa)
require(plyr)
library(MASS)
data(sp500.gr)
y = 100*window(sp500.gr, start=2005)
plot.ts(y)
#-------------------------------------------------------------------
npart   = 20        # Number of particles used in pgas
nmcmc   = 30000      # Number of iterations in the mcmc samplers after burnin
burnin  = 3000       # Number of iterations to burn
mcmseed = 90210

#  Bi-normal prior and initial values
init=c(0.95, 0.2, 0)
hyper = c(0.95, 0.5, 0.075, 0.1, -0.25)
sigma_MH = 1.7*10^(-4) * matrix(c(100,-25,-25,100),nrow=2,ncol=2)

# N=20 Run it
ptm <- proc.time()
u1 = pgasSV_binorm(nmcmc, burnin, y, init, hyper, sigma_MH, npart
                   , parms_2 = T, mcmseed)
(time2run = proc.time() - ptm)

# Acceptance Rate
cat("The acceptance rate is", u1$acp, "%.")

############### parallel  ########################
require(parallel)
require(doParallel)
require(foreach)
```

```
c1 = makeCluster(3)
registerDoParallel(c1)
out.all = foreach(npart=c(10,20,50,500,200,100), .packages=c("plyr","MASS"))%dopar%{
  u_10d1 = pgasSV_binorm(nmcmc, burnin, y, init, hyper, sigma_MH
                         , npart, parms_2 = T, mcmseed)
  u_10d1
}
stopCluster(c1)

############### Results  ########################
### Pretty pictures
parms = cbind(u1$phi, u1$q)
names = c(expression(phi), expression(sigma^2))
nobs=length(y)

#dev.new(height=5.5, width=9)
par(mfcol=c(3,2), mar=c(3,3,1,1), mgp=c(1.6,.6,0), oma=c(0,0,1.5,0))

# Parameters ...
for (i in 1:2){
  plot.ts(parms[,i], ylab='trace')
  mtext(names[i], 3, line=.25)
  acf(parms[,i], 100, xlim=c(1.4,100))
  hist(parms[,i], breaks=20,  prob=TRUE, main="", xlab="")
  lines(density(parms[,i], adjust=1.5))
  lp = quantile(parms[,i], .005)
  up = quantile(parms[,i], .995)
  abline(v=mean(parms[,i]), col=4, lwd=3, lty=1)
}

# States ...
#dev.new(height=6, width=9)
par(mfrow=c(2,1), mar=c(3,3,1.5,1), mgp=c(1.6,.6,0), oma=c(0,0,0,0))
mX = ts(apply(u1$X, 2, mean))
lX = ts(apply(u1$X, 2, quantile, 0.005))
uX = ts(apply(u1$X, 2, quantile, 0.995))
plot.ts(mX, ylab=expression(X[~t]), ylim=c(min(lX,mX)-.5,max(uX,mX)+.5))
xx=c(time(lX), rev(time(uX)))
yy=c(lX, rev(uX))
polygon(xx, yy, border=NA, col=rgb(0,.6,.3, alpha = .2))
legend("topleft",legend=c("Posterior Mean","99%CI"),col=c(1,rgb(0,.6,.3, alpha = .4)),
lty=c(1,1), lwd=c(1,5), cex=.85)
plot.ts(y, ylab=expression(Y[~t]))

#dev.new()
par(mfrow=c(1,1))
z=kde2d(parms[,1],(parms[,2]))
plot(parms[,1],(parms[,2]), pch=20, col=4
     , xlab=expression(phi), ylab=expression(sigma^2))
contour(z, drawlabels=FALSE, nlevels=15,  add=TRUE)

# Ineff
ineff = function(parm){
  parmacf = acf(parm, nmcmc, plot = F)$acf
  inef = 0
  for(i in 1:(nmcmc/2)){
    temp1 = parmacf[2*i-1]+parmacf[2*i]
    if(temp1 > 0){inef = inef + temp1}else{break}
  }
  inef = 2*inef - parmacf[1]
  return(inef)
}
ineff(parms[,1])
ineff(parms[,2])
```

```
##########################################################################
##      Application on S&P500 using 3-parms SV Model                    ##
##      Please run pgasSV_binorm(), cpf_as_sv(), resamplew()            ##
##             and log_g_func() first                                   ##
##########################################################################
#install.packages("devtools")
#devtools::install_github("nickpoison/nltsa")
library(nltsa)
require(plyr)
library(MASS)
data(sp500.gr)
y = 100*window(sp500.gr, start=2005)
plot.ts(y)
#-------------------------------------------------------------------
npart   = 20       # Number of particles used in pgas
nmcmc   = 2000     # Number of iterations in the mcmc samplers after burnin
burnin  = 200      # Number of iterations to burn
mcmseed = 90210

#  Bi-normal prior and initial values
init=c(0.95, 0.2, 0.15)
hyper = c(0.95, 0.5, 0.075, 0.1, -0.25)
sigma_MH = 1.7*10^(-4) * matrix(c(100,-25,-25,100),nrow=2,ncol=2)

# N=20 Run it
ptm <- proc.time()
u1 = pgasSV_binorm(nmcmc, burnin, y, init, hyper, sigma_MH, npart
                   , parms_2 = F, mcmseed)
(time2run = proc.time() - ptm)
# Acceptance Rate
cat("The acceptance rate is", u1$acp, "%.")

############### parallel  ########################
require(parallel)
require(doParallel)
require(foreach)
c1 = makeCluster(3)
registerDoParallel(c1)
out.all = foreach(npart=c(10,20,50,500,200,100)
                  , .packages=c("plyr","MASS"))%dopar%{
  u = pgasSV_binorm(nmcmc, burnin, y, init, hyper, sigma_MH
                    , npart, parms_2 = F, mcmseed)
  u
}
stopCluster(c1)

################  u1 N=20 ######################
### Pretty pictures
parms = cbind(u1$phi, u1$q, u1$mu)
true_parms = c(phi_sim, q_sim, mu_sim)
names = c(expression(phi), expression(sigma^2), expression(mu))

#dev.new(height=5.5, width=9)
par(mfcol=c(3,3), mar=c(3,3,1,1), mgp=c(1.6,.6,0), oma=c(0,0,1.5,0))
# Parameters ...
for (i in 1:3){
  plot.ts(parms[,i], ylab='trace')
  mtext(names[i], 3, line=.25)
  acf(parms[,i], 100, xlim=c(1.4,100))
  hist(parms[,i], breaks=20,  prob=TRUE, main="", xlab="")
  lines(density(parms[,i], adjust=1.5))
  lp = quantile(parms[,i], .005)
  up = quantile(parms[,i], .995)
  abline(v=mean(parms[,i]), col=4, lwd=3, lty=1)
  print(mean(parms[,i]))
}

```

```
# and States ...
#dev.new(height=6, width=9)
par(mfrow=c(2,1), mar=c(3,3,1.5,1), mgp=c(1.6,.6,0), oma=c(0,0,0,0))
mX = ts(apply(u1$X, 2, mean))
lX = ts(apply(u1$X, 2, quantile, 0.005))
uX = ts(apply(u1$X, 2, quantile, 0.995))
plot.ts(mX, ylab=expression(X[~t]), ylim=c(min(lX,mX)-.5,max(uX,mX)+.5))
xx=c(time(lX), rev(time(uX)))
yy=c(lX, rev(uX))
polygon(xx, yy, border=NA, col=rgb(0,.6,.3, alpha = .2))
legend("topleft",legend=c("Posterior Mean","99%CI"),col=c(1,rgb(0,.6,.3, alpha = .4)),
lty=c(1,1), lwd=c(1,5), cex=.85)
plot.ts(y, ylab=expression(Y[~t]))

#dev.new()
par(mfrow=c(1,1))
z=kde2d(parms[,1],(parms[,2]))
plot(parms[,1],(parms[,2]), pch=20, col=4
     , xlab=expression(phi), ylab=expression(sigma^2))
contour(z, drawlabels=FALSE, nlevels=15,  add=TRUE)

# Ineff
ineff = function(parm){
  parmacf = acf(parm, nmcmc, plot = F)$acf
  inef = 0
  for(i in 1:(nmcmc/2)){
    temp1 = parmacf[2*i-1]+parmacf[2*i]
    if(temp1 > 0){inef = inef + temp1}else{break}
  }
  inef = 2*inef - parmacf[1]
  return(inef)
}
ineff(parms[,1])
ineff(parms[,2])
ineff(parms[,3])
```

C.4 CODE FOR METHOD C IN SECTION 5.1.2 AND SECTION 5.2.2

```
##############################################################################
##   PGAS for SV model with Adaptive MCMC.                                  ##
##   Input (nmcmc, burnin, y, phi_init, q_init, npart, mcmseed)             ##
##   3 parameters: X(t)= mu + phi (X(t-1)-mu) + W(t) ~ iid N(0,q) ;         ##
##                 Y(t)=beta*exp{X(t)/2)}V(t) ~ iid N(0,1);                 ##
##                 beta=exp(mu/2)                                           ##
##   2 parameters: X(t)= phi*X(t-1) + W(t) ~ iid N(0,q) ;                   ##
##                 Y(t)=beta*exp{X(t)/2)}V(t) ~ iid N(0,1);                 ##
##                 beta=exp(mu/2)                                           ##
##############################################################################

pgasSV_binorm_ada = function(nmcmc, burnin, y, init, hyper, sigma_MH, npart
                             , lambda, alpha.tar, parms_2 = T, mcmseed){
# Input:
#   nmcmc - number of MCMC
#   burnin - number of burnin
#   y - measurements
#   init - initial value of (phi, q, mu)
#   npart - number of particles
#   hyper - hyperparameters for bivariate normal (phi, q),
#           user inputs (mu_phi, mu_q, sigma_phi, sigma_q, rho)
#   sigma_MH - initial value of covariance matrix for RWMH
#   lambda - initial value of lambda
#   alpha.tar - target acceptance rate
#   parms_2 - IF (parms_2=TRUE) THEN (2-parms SV model) ELSE (3-parms SV model)
#   mcmseed - seed for mcmc
# Output:
#   phi - sampled phi
#   q - sampled phi
#   mu - sampled mu
#   X - sampled hidden states
#   time - running time
#   acp - acceptence rate of Random Walk Metropolis Hastings

ptm <- proc.time()
numMCMC = nmcmc+burnin
N = npart

set.seed(mcmseed)
pr <- progress_text()         # displays progress (from plyr)
pr$init(numMCMC)

T = length(y)
X = matrix(0,numMCMC,T)
q = rep(0,numMCMC)             # q = state variance ## W(t) ~ iid N(0,q)
phi = rep(0,numMCMC)           # phi-values
mu = rep(0,numMCMC)            # leverage term
lambda.trace = rep(0,numMCMC)
gamma = rep(0.5,numMCMC)
g_diff.all = rep(0,numMCMC)
sigma.all = list(NA)
mu.all = list(NA)

# Initialize the parameters
phi[1] = init[1]
q[1] = init[2]
mu[1] = init[3]
lambda.trace[1] = lambda

# hyperparameters
mu_phi = hyper[1]
mu_q = hyper[2]
sigma_phi = hyper[3]
```

```
sigma_q = hyper[4]
rho = hyper[5]  # negative correlation
mu_MH = c(0,0)  # tuning parameter
sigma_MH = sigma_MH # tuning parameter
mu.all[[1]] = mu_MH
sigma.all[[1]] = sigma_MH

# Initialize the state by running a PF
if (parms_2){

  u = cpf_as_sv(y, phi[1], q[1], 0, N,  X[1,])   #  changed from X to X[1,]
  particles = u$x           # returned particles
  w = u$w                   # returned weights
  # Draw J
  J = which( (runif(1)-cumsum(w[,T])) < 0 )[1]
  X[1,] = scale(particles[J,], center=TRUE, scale=FALSE)

} else {

  u = cpf_as_sv(y, phi[1], q[1], mu[1], N,  X[1,])   #  changed from X to X[1,]
  particles = u$x           # returned particles
  w = u$w                   # returned weights
  # Draw J
  J = which( (runif(1)-cumsum(w[,T])) < 0 )[1]
  X[1,] = scale(particles[J,], center=TRUE, scale=FALSE)

}

# Run MCMC loop
if (parms_2){
  for(k in 2:numMCMC){
    # Sample the parameters (phi, sqrt_q) ~ bivariate normal RWMH
    parms = cbind(phi[k-1], sqrt(q[k-1]))
    (parms_star = parms + mvrnorm(1, mu_MH, lambda.trace[k-1]*sigma.all[[k-1]]))

    while(parms_star[2]^2 < 1e-2|parms_star[1]>2){
      (parms_star = parms + mvrnorm(1, mu_MH, lambda.trace[k-1]*sigma.all[[k-1]]))
    }

    g_star = log_g_func(parms_star, mu_phi, sigma_phi, mu_q, sigma_q
                        , rho, X[(k-1),], mu[k-1])
    g_old = log_g_func(parms, mu_phi, sigma_phi, mu_q, sigma_q
                        , rho, X[(k-1),], mu[k-1])
    g_diff = g_star - g_old
    if(is.na(g_diff)){
      phi[k] = parms[1]
      q[k] = parms[2]^2
    }else {
    if(log(runif(1)) < g_diff){
      phi[k] = parms_star[1]
      q[k] = parms_star[2]^2
      }else{
      phi[k] = parms[1]
      q[k] = parms[2]^2
      }
    }
    g_diff.all[k-1]=g_diff
    parm.new = cbind(phi[k],sqrt(q[k]))

    # Update Lambda
    lambda.trace[k] = exp(log(lambda.trace[k-1]) + gamma[k-1]*(exp(g_diff.all[k-1])-alpha.tar))
    sigma.all[[k]] = sigma.all[[k-1]] + gamma[k-1]*
                     (as.matrix(t(parm.new-mu.all[[k-1]]))%*%
                      (as.matrix(parm.new-mu.all[[k-1]])) - sigma.all[[k-1]])
     mu.all[[k]] = mu.all[[k-1]] + gamma[k-1]*(parm.new - mu.all[[k-1]])
     gamma[k] = 1/(500*k)
     mu[k] = 0

```

```
  # Run CPF-AS
  u = cpf_as_sv(y, phi[k], q[k], 0, N, X[k-1,])
  particles = u$x           # returned particles
  w = u$w                   # returned weight
  # Draw J (extract a particle trajectory)
  J = which( (runif(1)-cumsum(w[,T])) < 0 )[1]
  X[k,] = scale(particles[J,], center=TRUE, scale=FALSE)     # center X
  pr$step()
  }#end

} else {

  for(k in 2:numMCMC){
    # Sample the parameters (phi, sqrt_q) ~ bivariate normal RWMH
    parms = cbind(phi[k-1], sqrt(q[k-1]))
    (parms_star = parms + mvrnorm(1, mu_MH, lambda.trace[k-1]*sigma.all[[k-1]]))
    while(parms_star[2]^2 < 1e-2|parms_star[1]>2){
    (parms_star = parms + mvrnorm(1, mu_MH, lambda.trace[k-1]*sigma.all[[k-1]]))
    }

    g_star = log_g_func(parms_star, mu_phi, sigma_phi, mu_q, sigma_q
                        , rho, X[(k-1),], mu[k-1])
    g_old = log_g_func(parms, mu_phi, sigma_phi, mu_q, sigma_q
                        , rho, X[(k-1),], mu[k-1])
    g_diff = g_star - g_old
    if(is.na(g_diff)){
      phi[k] = parms[1]
      q[k] = parms[2]^2
    }else{
      if(log(runif(1)) < g_diff){
        phi[k] = parms_star[1]
        q[k] = parms_star[2]^2
      }else{
        phi[k] = parms[1]
        q[k] = parms[2]^2
      }
    }
    g_diff.all[k-1]=g_diff
    parm.new = cbind(phi[k],sqrt(q[k]))

    # Update Lambda
    lambda.trace[k] = exp(log(lambda.trace[k-1]) + gamma[k-1]*(exp(g_diff.all[k-1])-alpha.tar))
    sigma.all[[k]] = sigma.all[[k-1]] + gamma[k-1]*
                     (as.matrix(t(parm.new-mu.all[[k-1]]))%*%
                     (as.matrix(parm.new-mu.all[[k-1]])) - sigma.all[[k-1]])
    mu.all[[k]] = mu.all[[k-1]] + gamma[k-1]*(parm.new - mu.all[[k-1]])
    gamma[k] = 1/(500*k)

    # Sample mu
    sigmu = q[k]/((T-1)*(1-phi[k])^2+(1-phi[k]^2))
    muhat = sigmu*((1-phi[k]^2)*X[k-1,1]/q[k]+(1-phi[k])*(sum(X[k-1,2:T])-
    phi[k]*sum(X[k-1,1:(T-1)]))/q[k])
    mu[k] = muhat + sqrt(abs(sigmu))*rnorm(1)
    while(abs(mu[k]) > 2){
      mu[k] = muhat + sqrt(abs(sigmu))*rnorm(1)
    }

    # Run CPF-AS
    u = cpf_as_sv(y, phi[k], q[k], mu[k], N, X[k-1,])
    particles = u$x           # returned particles
    w = u$w                   # returned weight
    # Draw J (extract a particle trajectory)
    J = which( (runif(1)-cumsum(w[,T])) < 0 )[1]
    X[k,] = scale(particles[J,], center=TRUE, scale=FALSE)     # center X
    pr$step()
  }#end
}
```

```
bi = 1:burnin
acp = (dim(table(phi[-bi]))-1)/nmcmc*100
time2run10 = proc.time() - ptm

list(phi=phi[-bi], q=q[-bi], mu=mu[-bi], X=X[-bi,], time=time2run10, acp=acp)
}#end

#-------------------------------------------------------------------------

cpf_as_sv = function(y, phi, q, mu, N, X){
# Conditional particle filter with ancestor sampling
# Input:
#   y - measurements
#   phi - transition parameter
#   q - state noise variance
#   mu - leverage term
#   N - number of particles
#   X - conditioned particles

T = length(y)
x = matrix(0, N, T); # Particles
a = matrix(0, N, T); # Ancestor indices
w = matrix(0, N, T); # Weights
x[,1] = 0; # Deterministic initial condition
x[N,1] = X[1]

for (t in 1:T){
if(t != 1){
ind = resamplew(w[,t-1]);
ind = ind[sample.int(N)]; # default: no replacement
t1 = t-1
xpred = phi*x[, t1] + (1-phi)*mu # length N Vector
x[,t] = xpred[ind] + sqrt(q)*rnorm(N);
x[N,t] = X[t]; #Line 6

# Ancestor sampling
logm = -1/(2*q)*(X[t]-xpred)^2;
maxlogm = max(log(w[,t-1])+logm)
w_as = exp(log(w[,t-1])+logm - maxlogm)
w_as = w_as/sum(w_as);
ind[N] =  which( (runif(1)-cumsum(w_as)) < 0 )[1]

# Store the ancestor indices
a[,t] = ind;
}#end IF
# Compute importance weights
var_now = exp(x[,t]+mu)
logweights = -y[t]^2/(2*var_now) - 1/2*log(var_now)  # (up to an additive constant)
const = max(logweights); # Subtract the maximum value for numerical stability
weights = exp(logweights-const)
w[,t] = weights/sum(weights) # Save the normalized weights
}#end FOR

# Generate the trajectories from ancestor indices
ind = a[,T];
for(t in (T-1):1){
x[,t] = x[ind,t];
ind = a[ind,t];
}#end
list(x=x, w=w)
}#end

#-------------------------------------------------------------------

resamplew = function(w){
# multinomial resampling
N = length(w)
u = runif(N)    # uniform random number
```

```
cw = cumsum(w)  # Cumulative Sum
cw = cw/cw[N]
ucw = c(u,cw)
ind1 = sort(ucw, index.return=TRUE)$ix   # $ix will give the index
ind2 = which(ind1<=N)
i = ind2-(0:(N-1))
return(i)
}

#---------------------------------------------------------------

log_g_func = function(parms, mu_phi, sigma_phi, mu_q, sigma_q, rho, x, mu){
phi = parms[1]
q = parms[2]
Z = cbind(x[2:T]-mu, x[1:(T-1)]-mu)  # (T-1)*2 matrix
V = t(Z)%*%Z
term1 = -((phi-mu_phi)^2/(sigma_phi^2) + (q-mu_q)^2/(sigma_q^2)
        - 2*rho*(phi-mu_phi)*(q-mu_q)/(sigma_q*sigma_phi))/(2*(1-rho^2))
term2 = -(1-phi^2)*(x[1]-mu)^2/(2*q^2) - V[1,1]/(2*q^2) - V[2,2]*
        (phi^2)/(2*q^2) + V[1,2]*phi/(q^2)
g = .25*(log((1-phi^2)^2)) - .5*T*log((q)^2) + term1 + term2
return(g)
}
```

```
#########################################################################
##     Simulation study for 3-parms SV Model using PGAS_adaptive       ##
##      Please run pgasSV_binorm_ada(), cpf_as_sv(), resamplew()       ##
##             and log_g_func() first                                  ##
#########################################################################
library(nltsa)
require(plyr)
require(MASS)

# Generate data
set.seed(1989)
phi_sim = 0.85; q_sim = 0.2; mu_sim = 0.1
n = 1000
h_sim = rep(0, n); y_sim = rep(0, n)
h_sim[1] = mu_sim + sqrt(q_sim/(1-phi_sim^2))*rnorm(1)
for(i in 1:1000){
y_sim[i] = exp(mu_sim/2)*exp(h_sim[i]/2)*rnorm(1)
h_sim[i+1] = mu_sim + phi_sim*(h_sim[i]-mu_sim) + sqrt(q_sim)*rnorm(1)
}
plot.ts(y_sim)

#---------------------------------------------------------------
npart   = 20        # Number of particles used in pgas
nmcmc   = 33000     # Number of iterations in the mcmc samplers after burnin
burnin  = 3000      # Number of iterations to burn
mcmseed = 90210

#  Bi-normal prior and initial values
init = c(.95, 0.25, 0.15)
hyper = c(0.9, 0.25, 0.25, 0.25, -0.25)
sigma_MH = 0.026*matrix(c(1,0.25,0.25,1),nrow=2,ncol=2)
lambda_init = 2.8
alpha.tar = 0.20

# N=20 Run it
ptm <- proc.time()
u1 = pgasSV_binorm_ada(nmcmc, burnin, y_sim, init, hyper, sigma_MH,
                       npart, lambda_init, alpha.tar, parms_2 = F, mcmseed)
(time2run10 = proc.time() - ptm)
```

```
40
41 # Acceptance Rate
42 cat("The acceptance rate is", u1$acp, "%.")
43
44 ################  u1 N=20 ######################
45 ### Pretty pictures
46 parms = cbind(u1$phi, u1$q, u1$mu)
47 names = c(expression(phi), expression(sigma^2), expression(mu))
48 true_parms = c(phi_sim, q_sim, mu_sim)
49 nobs=length(y_sim)
50
51 #dev.new(height=5.5, width=9)
52 par(mfcol=c(3,3), mar=c(3,3,1,1), mgp=c(1.6,.6,0), oma=c(0,0,1.5,0))
53
54 # Parameters ...
55 for (i in 1:3){
56   plot.ts(parms[,i], ylab='trace')
57   mtext(names[i], 3, line=.25)
58   acf(parms[,i], 100, xlim=c(1.4,100))
59   hist(parms[,i], breaks=20,  prob=TRUE, main="", xlab="")
60   lines(density(parms[,i], adjust=1.5))
61   lp = quantile(parms[,i], .005)
62   up = quantile(parms[,i], .995)
63   abline(v=mean(parms[,i]), col=4, lwd=3, lty=1)
64   abline(v=true_parms[i], col=2, lwd=1, lty=1)
65   print(mean(parms[,i]))
66 }
67
68 # States
69 par(mfrow=c(2,1), mar=c(3,3,1.5,1), mgp=c(1.6,.6,0), oma=c(0,0,0,0))
70 mX = ts(apply(u1$X, 2, mean))
71 lX = ts(apply(u1$X, 2, quantile, 0.005))
72 uX = ts(apply(u1$X, 2, quantile, 0.995))
73 plot.ts(mX, ylab=expression(X[~t]), ylim=c(min(lX,mX)-.5,max(uX,mX)+.5))
74 xx=c(time(lX), rev(time(uX)))
75 yy=c(lX, rev(uX))
76 polygon(xx, yy, border=NA, col=rgb(0,.6,.3, alpha = .2))
77 legend("topleft",legend=c("Posterior Mean","99%CI"),
78         col=c(1,rgb(0,.6,.3, alpha = .4)),
79 lty=c(1,1), lwd=c(1,5), cex=.85)
80 plot.ts(y_sim, ylab=expression(Y[~t]))
81
82 par(mfrow=c(1,1))
83 z=kde2d(parms[,1],(parms[,2]))
84 plot(parms[,1],(parms[,2]), pch=20, col=4
85      , xlab=expression(phi), ylab=expression(sigma^2))
86 contour(z, drawlabels=FALSE, nlevels=15,  add=TRUE)
87
88 # Ineff
89 ineff = function(parm){
90   parmacf = acf(parm, nmcmc, plot = F)$acf
91   inef = 0
92   for(i in 1:(nmcmc/2)){
93     temp1 = parmacf[2*i-1]+parmacf[2*i]
94     if(temp1 > 0){inef = inef + temp1}else{break}
95   }
96   inef = 2*inef - parmacf[1]
97   return(inef)
98 }
99
100 ineff(parms[,1]);
101 ineff(parms[,2]);
102 ineff(parms[,3])
```

```
#########################################################################
##    SP500 SV Model using PGAS_adaptive                                ##
##       3 parameters SV model                                          ##
##       Please run pgasSV_binorm_ada(), cpf_as_sv(), resamplew()       ##
##             and log_g_func() first                                   ##
#########################################################################
library(nltsa)
data(sp500.gr)
require(plyr)
require(MASS)
y = 100*window(sp500.gr, start=2005)
#-----------------------------------------------------------------
npart   = 20       # Number of particles used in pgas
nmcmc   = 33000     # Number of iterations in the mcmc samplers after burnin
burnin  = 3000      # Number of iterations to burn
mcmseed = 90210

#  Bi-normal prior and initial values
init = c(.95, 0.25, 0)
hyper = c(0.9, 0.25, 0.25, 0.25, -0.25)
sigma_MH = 0.1*matrix(c(1,0.25,0.25,1),nrow=2,ncol=2)
lambda_init = 2
alpha.tar = 0.25

# N=20 Run it
ptm <- proc.time()
u1 = pgasSV_binorm_ada(nmcmc, burnin, y, init, hyper, sigma_MH
                       , npart, lambda_init, alpha.tar, parms_2 = T, mcmseed)
(time2run10 = proc.time() - ptm)
# Acceptance Rate
cat("The acceptance rate is", u1$acp, "%.")

################ Results ######################
parms = cbind(u1$phi, u1$q)
names = c(expression(phi), expression(sigma^2))
nobs=length(y)

par(mfcol=c(3,2), mar=c(3,3,1,1), mgp=c(1.6,.6,0), oma=c(0,0,1.5,0))
# Parameters ...
for (i in 1:2){
  plot.ts(parms[,i], ylab='trace')
  mtext(names[i], 3, line=.25)
  acf(parms[,i], 100, xlim=c(1.4,100))
  hist(parms[,i], breaks=20,  prob=TRUE, main="", xlab="")
  lines(density(parms[,i], adjust=1.5))
  lp = quantile(parms[,i], .005)
  up = quantile(parms[,i], .995)
  abline(v=mean(parms[,i]), col=4, lwd=3, lty=1)
  print(mean(parms[,i]))
}

# States
par(mfrow=c(2,1), mar=c(3,3,1.5,1), mgp=c(1.6,.6,0), oma=c(0,0,0,0))
mX = ts(apply(u1$X, 2, mean))
lX = ts(apply(u1$X, 2, quantile, 0.005))
uX = ts(apply(u1$X, 2, quantile, 0.995))
plot.ts(mX, ylab=expression(X[~t]), ylim=c(min(lX,mX)-.5,max(uX,mX)+.5))
xx=c(time(lX), rev(time(uX)))
yy=c(lX, rev(uX))
polygon(xx, yy, border=NA, col=rgb(0,.6,.3, alpha = .2))
legend("topleft",legend=c("Posterior Mean","99%CI"),col=c(1,rgb(0,.6,.3, alpha = .4)),
lty=c(1,1), lwd=c(1,5), cex=.85)
plot.ts(y, ylab=expression(Y[~t]))

par(mfrow=c(1,1))
z=kde2d(parms[,1],(parms[,2]))
plot(parms[,1],(parms[,2]), pch=20, col=4, xlab=expression(phi), ylab=expression(sigma^2))
```

```
contour(z, drawlabels=FALSE, nlevels=15,  add=TRUE)

# Ineff
ineff = function(parm){
  parmacf = acf(parm, nmcmc, plot = F)$acf
  inef = 0
  for(i in 1:(nmcmc/2)){
    temp1 = parmacf[2*i-1]+parmacf[2*i]
    if(temp1 > 0){inef = inef + temp1}else{break}
  }
  inef = 2*inef - parmacf[1]
  return(inef)
}
ineff(parms[,1]); ineff(parms[,2])
```

C.5 CODE FOR PROPOSED METHOD IN SECTION 5.3

```
##########################################################################
##     PGAS for Multivarite SV model.                                   ##
##     2 parameters: X(t) = phi * X(t-1) + W(t) ~ iid N(0,q);           ##
##                   Y(t) = beta*exp{X(t)/2}V(t) ~ iid N(0,1);          ##
##                   beta=exp(mu/2)                                     ##
##########################################################################

pgasMSV_binorm = function(nmcmc, burnin, y, init, beta_init, hyper, sigma_MH, npart, mcmseed){
# Input:
#   nmcmc - number of MCMC
#   burnin - number of burnin
#   y - measurements
#   init - initial value of (phi, q)
#   beta_init - initial value of beta_i, e.g. user inputs (beta_1, beta_2, beta_3)
#   npart - number of particles
#   hyper - hyperparameters for bivariate normal (phi, q),
#           user inputs (mu_phi, mu_q, sigma_phi, sigma_q, rho)
#   sigma_MH - covariance matrix for Random Walk Metropolis Hastings
#   mcmseed - seed for mcmc

# Output:
#   phi - sampled phi
#   q - sampled phi
#   beta - sampled beta_i
#   X - sampled hidden states
#   time - running time
#   acp - acceptence rate of Random Walk Metropolis Hastings

ptm <- proc.time()
numMCMC = nmcmc+burnin
N = npart
set.seed(mcmseed)
pr <- progress_text()         # displays progress (from plyr)
pr$init(numMCMC)
T = nrow(y)
p = ncol(y)
X = matrix(0,numMCMC,T)
q = rep(0,numMCMC)             # q = state variance ## W(t) ~ iid N(0,q)
phi = rep(0,numMCMC)           # phi-values
mu = rep(0,numMCMC)            # leverage term
beta = matrix(0,numMCMC,p)

# Initialize the parameters
phi[1] = init[1]
q[1] = init[2]
mu[1] = 0
for(i in 1:p){
  beta[1,i] = beta_init[i]
}

# hyperparameters
mu_phi = hyper[1]
mu_q = hyper[2]
sigma_phi = hyper[3]
sigma_q = hyper[4]
rho = hyper[5]  # negative correlation
mu_MH = c(0,0)  # tuning parameter
sigma_MH = sigma_MH # tuning parameter


# Initialize the state by running a PF
u = cpf_as_sv(y, phi[1], q[1], 0, N,  X[1,], beta[1,])
particles = u$x           # returned particles
```

```
w = u$w                    # returned weights
# Draw J
J = which( (runif(1)-cumsum(w[,T])) < 0 )[1]
X[1,] = scale(particles[J,], center=TRUE, scale=FALSE)


# Run MCMC loop
for(k in 2:numMCMC){
  # Sample the parameters (phi, sqrt_q) ~ bivariate normal RWMH
  parms = cbind(phi[k-1], sqrt(q[k-1]))
  parms_star = parms + mvrnorm(1, mu_MH, sigma_MH)
  while(parms_star[2]^2 < 2e-2|parms_star[1]>2){
  parms_star = parms + mvrnorm(1, mu_MH, sigma_MH)
  }

  g_star = log_g_func(parms_star, mu_phi, sigma_phi, mu_q
                      , sigma_q, rho, X[(k-1),], mu[k-1])
  g_old = log_g_func(parms, mu_phi, sigma_phi, mu_q, sigma_q
                     , rho, X[(k-1),], mu[k-1])
  g_diff = g_star - g_old
  if(is.na(g_diff)){
    phi[k] = parms[1]
    q[k] = parms[2]^2
  }else {
    if(log(runif(1)) < g_diff){
      phi[k] = parms_star[1]
      q[k] = parms_star[2]^2
    }else{
      phi[k] = parms[1]
      q[k] = parms[2]^2
    }
  }
  mu[k] = 0

  # Sample beta, prior unif, N(0,b)=N(0,inf)
  for (i in 1:p){
  atemp = T/2 - 1
  btemp = sum(y[,i]^2/exp(X[k-1,]))/2
  beta[k,i] = sqrt(1/rgamma(1, atemp, btemp))
  }

  # # Sample beta, prior IG(a,b)
  # a = .001
  # b = .001
  # for (i in 1:p){
  #   atemp = T/2 + a
  #   btemp = sum(y[,i]^2/exp(X[k-1,]))/2 + b
  #   beta[k,i] = sqrt(1/rgamma(1, atemp, btemp))
  # }

  # Run CPF-AS
  u = cpf_as_sv(y, phi[k], q[k], 0, N, X[k-1,], beta[k,])
  particles = u$x           # returned particles
  w = u$w                   # returned weight
  # Draw J (extract a particle trajectory)
  J = which( (runif(1)-cumsum(w[,T])) < 0 )[1]
  X[k,] = scale(particles[J,], center=TRUE, scale=FALSE)
  pr$step()
}#end

bi = 1:burnin
time2run = proc.time() - ptm
acp = dim(table(phi[-bi]))/nmcmc*100
list(phi=phi[-bi], q=q[-bi], beta=beta[-bi,], X=X[-bi,], time=time2run, acp=acp)
}#end

#-------------------------------------------------------------------------
cpf_as_sv = function(y, phi, q, mu, N, X, beta){
```

```
# Conditional particle filter with ancestor sampling
# Input:
#   y - measurements
#   phi - transition parameter
#   q - state noise variance
#   mu - leverage term
#   N - number of particles
#   X - conditioned particles

T = nrow(y)
pp= ncol(y)
x = matrix(0, N, T); # Particles
a = matrix(0, N, T); # Ancestor indices
w = matrix(0, N, T); # Weights
x[,1] = 0; # Deterministic initial condition
x[N,1] = X[1]

for (t in 1:T){
  if(t != 1){
  ind = resamplew(w[,t-1]);
  ind = ind[sample.int(N)]; # default: no replacement
  t1 = t-1
  xpred = phi*x[, t1] # length N Vector
  x[,t] = xpred[ind] + sqrt(q)*rnorm(N);
  x[N,t] = X[t]; #Line 6

  # Ancestor sampling
  logm = -1/(2*q)*(X[t]-xpred)^2;
  maxlogm = max(log(w[,t-1])+logm)
  w_as = exp(log(w[,t-1])+logm - maxlogm)
  w_as = w_as/sum(w_as);
  ind[N] =  which( (runif(1)-cumsum(w_as)) < 0 )[1]

  # Store the ancestor indices
  a[,t] = ind;
  }#end IF
  # Compute importance weights
  exp_now = exp(x[,t])
  temp = (y[t,]/beta)^2
  logweights = -sum(log(beta)) - (pp/2)*x[,t] - 0.5*(sum(temp)/exp_now)
  const = max(logweights); # Subtract the maximum value for numerical stability
  weights = exp(logweights-const)
  w[,t] = weights/sum(weights) # Save the normalized weights
}#end FOR

# Generate the trajectories from ancestor indices
ind = a[,T];
for(t in (T-1):1){
x[,t] = x[ind,t];
ind = a[ind,t];
}#end
list(x=x, w=w)
}#end
#-------------------------------------------------------------------
resamplew = function(w){
# multinomial resampling
  N = length(w)
  u = runif(N)    # uniform random number
  cw = cumsum(w)  # Cumulative Sum
  cw = cw/cw[N]
  ucw = c(u,cw)
  ind1 = sort(ucw, index.return=TRUE)$ix   # $ix will give the index
  ind2 = which(ind1<=N)
  i = ind2-(0:(N-1))
  return(i)
}
#-------------------------------------------------------------------
log_g_func = function(parms, mu_phi, sigma_phi, mu_q, sigma_q, rho, x, mu){
```

```
# Calculate acceptance probability for RWMH
  phi = parms[1]
  q = parms[2]
  Z = cbind(x[2:T]-mu, x[1:(T-1)]-mu)  # (T-1)*2 matrix
  V = t(Z)%*%Z
  term1 = -((phi-mu_phi)^2/(sigma_phi^2) + (q-mu_q)^2/(sigma_q^2) -
          2*rho*(phi-mu_phi)*(q-mu_q)/(sigma_q*sigma_phi))/(2*(1-rho^2))
  term2 = -(1-phi^2)*(x[1]-mu)^2/(2*q^2) - V[1,1]/(2*q^2) -
          V[2,2]*(phi^2)/(2*q^2) + V[1,2]*phi/(q^2)
  g = .25*(log((1-phi^2)^2)) - .5*T*log((q)^2) + term1 + term2
  return(g)
}
```

```
##############################################################################
##            MSV Model using PGAS for BOA, Citi and JPM data               ##
##      Please run pgasMSV_binorm(), cpf_as_sv(), resamplew()               ##
##             and log_g_func() first                                       ##
##############################################################################
require(plyr)
library(MASS)

# Load data
boaciti = read.csv("Boa&Citi&JPM.csv")
boaciti = boaciti[,-1]
boa = diff(boaciti[,1])/boaciti[1:3243,1]
citi = diff(boaciti[,2])/boaciti[1:3243,2]
jpm = diff(boaciti[,3])/boaciti[1:3243,3]
ytemp = ts(cbind(boa,citi,jpm), start=c(2005,1,3),frequency=260.25)
y = 100*window(ytemp, start=2007, end=2013)
par(mfrow=c(3,1), mar=c(3,3,1.5,1), mgp=c(1.6,.6,0), oma=c(0,0,0,0))
plot.ts(y[,1], ylab = "BOA", ylim=c(-40,60))
plot.ts(y[,2], ylab = "Citi", ylim=c(-40,60))
plot.ts(y[,3], ylab = "JPM", ylim=c(-40,60))

#-----------------------------------------------------------------------
npart   = 20        # Number of particles used in pgas
nmcmc   = 30000     # Number of iterations in the mcmc sampler after burnin
burnin  = 3000      # Number of iterations to burn
mcmseed = 90210

#  Initial values of phi, q, mu
init = c(0.95, 0.15)
beta_init = c(1.5, 1.5, 1.5)
hyper = c(0.95, 0.5, 0.075, 0.1, -0.25)
sigma_MH = 1.7*10^(-4) * matrix(c(100,-25,-25,100),nrow=2,ncol=2)

################  N=20 particles ######################
# Run it
ptm <- proc.time()
u_nd = pgasMSV_binorm(nmcmc, burnin, y, init, beta_init, hyper
                      , sigma_MH, npart, mcmseed)
(time2run = proc.time() - ptm)
# Acceptance Rate
cat("The acceptance rate is", u_nd$acp, "%.")

################  Results   ############################
### Pretty pictures
parms = cbind(u_nd$phi, u_nd$q, u_nd$beta[,1], u_nd$beta[,2], u_nd$beta[,3])
names = c(expression(phi), expression(sigma^2), expression(beta[1])
          , expression(beta[2]), expression(beta[3]))

#dev.new(height=5.5, width=9)
par(mfcol=c(3,5), mar=c(3,3,1,1), mgp=c(1.6,.6,0), oma=c(0,0,1.5,0))
```

```
51
52 # Parameters ...
53 for (i in 1:5){
54   plot.ts(parms[,i], ylab='trace')
55   mtext(names[i], 3, line=.25)
56   acf(parms[,i], 100, xlim=c(1.2,100))
57   hist(parms[,i], breaks=20,  prob=TRUE, main="", xlab="")
58   lines(density(parms[,i], adjust=1.5))
59   lp = quantile(parms[,i], .005)
60   up = quantile(parms[,i], .995)
61   abline(v=mean(parms[,i]), col=4, lwd=3, lty=1)
62   print(mean(parms[,i]))
63 }
64
65 # States ...
66 par(mfrow=c(4,1), mar=c(3,3,1.5,1), mgp=c(1.6,.6,0), oma=c(0,0,0,0))
67 mX = ts(apply(u_nd$X, 2, mean))
68 lX = ts(apply(u_nd$X, 2, quantile, 0.005))
69 uX = ts(apply(u_nd$X, 2, quantile, 0.995))
70 plot.ts(mX[-1], ylab=expression(X[~t]), ylim=c(min(lX,mX)-.5,max(uX,mX)+.5))
71 xx=c(time(lX), rev(time(uX)))
72 yy=c(lX, rev(uX))
73 polygon(xx, yy, border=NA, col=rgb(0,.6,.3, alpha = .2))
74 legend("topleft",legend=c("Posterior Mean","99%CI")
75        ,col=c(1,rgb(0,.6,.3, alpha = .35)),
76 lty=c(1,1), lwd=c(1,5), cex=0.75)
77 plot.ts(y[,1], ylab = "BOA")
78 plot.ts(y[,2], ylab = "Citi")
79 plot.ts(y[,3], ylab = "JPM")
80
81 # Ineff
82 ineff = function(parm){
83 parmacf = acf(parm, nmcmc, plot = F)$acf
84 inef = 0
85 for(i in 1:(nmcmc/2)){
86 temp1 = parmacf[2*i-1]+parmacf[2*i]
87 if(temp1 > 0){inef = inef + temp1}else{break}
88 }
89 inef = 2*inef - parmacf[1]
90 return(inef)
91 }
92 ineff(parms[,1]); ineff(parms[,2])
93 ineff(parms[,3]); ineff(parms[,4])
94 ineff(parms[,5])
```

BIBLIOGRAPHY

Christophe Andrieu and Johannes Thoms. A tutorial on adaptive mcmc. *Statistics and Computing*, 18(4):343–373, December 2008. ISSN 0960-3174. doi: 10.1007/s11222-008-9110-y.

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.

Manabu Asai, Michael McAleer, and Jun Yu. Multivariate stochastic volatility: a review. *Econometric Reviews*, 25(2-3):145–175, 2006.

Fischer Black and Myron Scholes. The pricing of options and corporate liabilities. *Journal of political economy*, 81(3):637–654, 1973.

Moulines Eric Cappé, Olivier and Tobias Ryden. *Inference in Hidden Markov Models*. Springer, 2005. ISBN 978-0-387-40264-2. doi: 10.1007/0-387-28982-8_6. URL http://dx.doi.org/10.1007/0-387-28982-8_6.

Marc Chesney and Louis Scott. Pricing european currency options: A comparison of the modified black-scholes model and a random variance model. *Journal of Financial and Quantitative Analysis*, 24(3):267–284, 1989.

Siddhartha Chib and Edward Greenberg. Markov chain monte carlo simulation methods in econometrics. *Econometric theory*, 12(3):409–431, 1996.

Pierre Del Moral. Non-linear filtering: interacting particle resolution. *Markov processes and related fields*, 2(4):555–581, 1996.

Arnaud Doucet, Simon Godsill, and Christophe Andrieu. On sequential monte carlo sampling methods for bayesian filtering. *Statistics and Computing*, 10(3):197–208, Jul 2000. ISSN 1573-1375. doi: 10.1023/A:1008935410038. URL https://doi.org/10.1023/A:1008935410038.

Darrell Duffie and Kenneth J Singleton. Simulated moments estimation of markov models of asset prices, 1990.

Charles J. Geyer. Practical markov chain monte carlo. *Statist. Sci.*, 7(4):473–483, 11 1992. doi: 10.1214/ss/1177011137. URL https://doi.org/10.1214/ss/1177011137.

Simon J Godsill, Arnaud Doucet, and Mike West. Monte carlo smoothing for nonlinear time series. *Journal of the American Statistical Association*, 99(465):156–168, 2004. doi: 10.1198/016214504000000151. URL https://doi.org/10.1198/016214504000000151.

Heikki Haario, Eero Saksman, and Johanna Tamminen. An adaptive metropolis algorithm. *Bernoulli*, 7(2):223–242, 04 2001. URL https://projecteuclid.org:443/euclid.bj/1080222083.

Andrew Harvey, Esther Ruiz, and Neil Shephard. Multivariate stochastic variance models. *The Review of Economic Studies*, 61(2):247–264, 1994.

Eric Jacquier, Nicholas G Polson, and Peter E Rossi. Bayesian analysis of stochastic volatility models. *Journal of Business & Economic Statistics*, 20(1):69–87, 1994.

R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35, 1960. doi: 10.1115/1.3662552.

R. E. Kalman and R. S. Bucy. New results in linear filtering and prediction theory. *Journal of Basic Engineering*, 83(1):95, 1961. doi: 10.1115/1.3658902.

Sangjoon Kim, Neil Shephard, and Siddhartha Chib. Stochastic volatility: likelihood inference and comparison with arch models. *The review of economic studies*, 65(3):361–393, 1998.

Fredrik Lindsten, Randal Douc, and Eric Moulines. Particle gibbs with ancestor sampling. *Journal of Machine Learning Research*, 15:2145–2184, 2014. ISSN 0303-6898. doi: 10.1111/sjos.12136.

Jun S Liu and Rong Chen. Sequential monte carlo methods for dynamic systems. *Journal of the American statistical association*, 93(443):1032–1044, 1998.

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.

Neil Shephard. Statistical aspects of arch and stochastic volatility. *Monographs on Statistics and Applied Probability*, 65:1–68, 1996.

Robert H Shumway and David S Stoffer. An approach to time series smoothing and forecasting using the em algorithm. *Journal of time series analysis*, 3(4):253–264, 1982.

Robert H. Shumway and David S. Stoffer. *Time series analysis and its applications: with R examples*. Springer, New York, 4rd edition, 2017. ISBN 1431875X;9783319524511;.

Stephen Taylor. Modelling financial time series (second edition). *World Scientific Publishing*, 12 2007.

Stephen J Taylor. Modeling stochastic volatility: A review and comparative study. *Mathematical finance*, 4(2):183–204, 1994.

Nick Whiteley, Christophe Andrieu, and Arnaud Doucet. Efficient bayesian inference for switching state-space models using discrete particle markov chain monte carlo methods. *arXiv preprint arXiv:1011.2437*, 2010.

James B Wiggins. Option values under stochastic volatility: Theory and empirical estimates. *Journal of financial economics*, 19(2):351–372, 1987.