THREE ESSAYS ON ECONOMETRICS: ASYMMETRIC EXPONENTIAL POWER DISTRIBUTION, ECONOMETRIC COMPUTATION, AND MULTIFACTOR MODEL

BY SHILIANG LI

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Economics

Written under the direction of

Hiroki Tsurumi

and approved by

_____

_____

_____

_____

New Brunswick, New Jersey

October, 2011

ABSTRACT OF THE DISSERTATION

Three Essays on Econometrics: Asymmetric Exponential Power Distribution, Econometric Computation, and Multifactor Model

by Shiliang Li

Dissertation Director: Hiroki Tsurumi

The dissertation consists of three independent essays, and they are put in as three chapters.

The goal of the first chapter is to develop an estimation procedure for financial time series models with the error terms following the Asymmetric Exponential Power Distribution (AEPD). The AEPD is the most general class of unimodal distributions. In addition to the usual location and scale parameters, it has skewness and kurtosis parameters. The kurtosis parameter is hard to estimate when the sample size is small and skewness is large. We show that when the skewness parameter is either close to zero or close to one the estimation of the kurtosis parameters are virtually unidentifiable unless the sample size is large. We analyze the nonlinear GARCH model (NGARCH) and an asset pricing model known as CKLS. We devise Bayesian Markov chain Monte Carlo (MCMC) algorithms.

In chapter 2, we focus on econometric computation and develop a method to speed up intensive computation. The combination of MATLAB, C/C++ and Graphic Processing Unit (GPU) is a method to put convenience and speed together. MATLAB is a high level computing language for econometrics. C/C++ language, on the other hand,

is more flexible and more powerful than interpreted languages such as GAUSS, MATLAB and SAS. But it requires more professional programming skills. There are three levels of speedup (within one personal computer). The lowest is simple C++ substitution. The faster level is parallel computing using multicore CPU. The fastest speed-up is parallel computing using GPU cards. The compiled codes which use GPU can run in MATLAB hundreds time faster than corresponding MATLAB script functions.

In chapter 3, we focus on the estimation of the multifactor CKLS model. The CKLS model provides a simple econometric framework to nest some popular term structure models such as the ones proposed by Merton (1973), Vasicek (1977), and Cox, Ingasol and Ross (CIR) (1985). In this chapter, we provide a new MCMC algorithm to estimate the multifactor CKLS model. Compared to the algorithm in Sowar (2005), our algorithm is more efficient andcan be applied to multifactor models whose dimension is more than two.

Acknowledgements

The writing of this dissertation is one of the most significant academic challenges for me. The dissertation could not be written without the inspiration, support, suggestion and guidance from many people. I would like to express a deep feeling of gratitude and appreciation to my advisor, Professor Hiroki Tsurumi. His wisdom, knowledge on various field and encouragement inspired and motivated me. He spent a lot of time on my research. His comments, advice and suggestion point into almost every sentence in this dissertation. I am grateful to Professor Roger Klein. He give me a lot of valuable suggestions on my second year paper that is a part of this dissertation. I would like to thank Professor Norman Swanson for his priceless suggestions on writing paper and his interesting lecture, time series. I am grateful to Professor John Landon-Lane for his inspiring lecture and his comments on this dissertation. I am greatly thankful to Professor Neil Sheflin for his comments on this dissertation. I would like to thank Professor Elena Goldman for serving as the outside committee member for my dissertation defense.

I would like to thank Professor Liuling Li for her comments and computation help on this dissertation. I thank Dorothy Rinaldi for her help on any question and any problem in last seven years. I thank the faculty and graduate students in the Economics at Rutgers for providing such an outstanding research and study environment.

Last but not least, I would like to thank my wife Li Sun. She support me financially and emotionally for last seven years. I would like to thank my Mom and Dad for their patience and emotional support.

Dedication

To my wife Li and my three angels: Kaiyi, Yuchen and Yujun

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This dissertation consists of three essays on The Asymmetric Exponential Power Distribution(AEPD), parallel computation and interest rate modeling respectively. They are independent and in three chapters.

The asymmetric Exponential Power Distribution (AEPD) is the most general class of unimodal distributions over the range of $-\infty$ to ∞. It includes many familiar distributions such as skewed or non-skewed normal or Laplace distributions. Many financial data have leptokurtic and slightly skewed distributions. In parametric statistical inference we should use a distribution that explains the data best. The goal of the first chapter is to develop an estimation procedure for financial time series models with the error terms following the Asymmetric Exponential Power Distribution (AEPD). In addition to the usual location and scale parameters, it has skewness and kurtosis parameters. The kurtosis parameter is hard to estimate when the sample size is small and skewness is large. We show that when the skewness parameter is either close to zero or close to one the estimation of the kurtosis parameters are virtually unidentifiable unless the sample size is large. We analyze the nonlinear GARCH model (NGARCH) and an asset pricing model known as CKLS. We devise Bayesian Markov chain Monte Carlo (MCMC) algorithms. The main contributions of this paper are: generalizing the method to draw AEPD random variate from density with all five parameters; deriving the another computation method of expectation and variance of AEPD; setting up a Bayesian algorithm for AEPD-NGARCH model, the algorithm save computation time and easily find random walk variance; showing that AEPD-NGARCH(1,1) can not explain the daily return on S&P500 composite index.

Chapter 2 is motivated from the slow computation rate when we use MCMC method

to do Bayesian inference. In this paper, we extend the simple combination of MATLAB and C++ to massive parallel computation method which using Graphic Processing Unit (GPU). The GPU parallel method speeds up intensive computation dramatically. In the paper, we illustrate that using a series of computation examples. In addition the combination of MATLAB and C++, we also compare several computation languages such as Gauss and R et. al. The combination of MATLAB, C/C++ and Graphic Processing Unit is a method to put convenience and speed together. MATLAB is a high level computing language for econometrics. It provides an interactive environment for data analysis, visual output and various toolboxes for special computation. However, MATLAB is a kind of script language and works too slow for certain classes of computation. C/C++ language, on the other hand, is more flexible and more powerful than interpreted languages such as GAUSS, MATLAB and SAS. But it requires more professional programming skills. There are three levels for the speedup (within one personal computer). The lowest is simple substitution. Some loop functions that can not be optimized with MATLAB are substituted by compiled C/C++ codes. The faster level is parallel computing using multicore CPU. In this method, we can write C/C++ codes with OpenMP library for convenience. The compiled C/C++ codes will work in MATLAB environment and automatically use multithreads to speed up computation. The fastest speed-up is parallel computing using GPU (graphics Processing Units) cards. The compiled codes which use GPU can run in MATLAB and hundreds time faster than corresponding MATLAB script funtions.

Chapter 3 also focus on Bayesian inference. In this paper, we set up a new Bayesian inference algorithm for the the multifactor CKLS model comparing to Sowar (2005). The CKLS model provides a simple econometric framework to nest some popular term structure models such as the ones proposed by Merton (1973), Vasicek (1977), and Cox, Ingasol and Ross (CIR) (1985). However, One-factor models, assume that changes in the interest rates of all maturities are driven by changes in a single underlying random factor which is the short rate. These kind of models have the same problem of perfect correlation of different maturities interest rates. In addition, the one-factor models can only accommodate yield curves that are monotonically increasing, monotonically

decreasing, or normally humped. So, several multifactor CKLS models arive. In this chapter, we provide a new MCMC algorithm to estimate the multifactor CKLS model from Brennan and Schward (1979). Compared to the algorithm in Sowar (2005), our algorithm is more efficient and can be applied to multifactor models whose dimension is more than two.

Chapter 2

Regression Models with the Error Term Following the Asymmetric Exponential Power Distribution

2.1 Introduction

The asymmetric Exponential Power Distribution (AEPD) is the most general class of unimodal distributions over the range of $-\infty$ to ∞. It includes many familiar distributions such as skewed or non-skewed normal or Laplace distributions. Many financial data have leptokurtic and slightly skewed distributions. In parametric statistical inference we should use a distribution that explains the data best. Zhu and Zinde-Walsh (2009) have demonstrated that the AEPD has desired statistical properties and that the location, scale and shape are represented by interpretable parameters. The moments has closed-form expression.

I will first present in Section 2.2 the AEPD distribution. Since the AEPD has five parameters, we will present how each parameter can be interpreted. We introduce the concept of the left and right kurtosis measure. Then I will discuss the Zhu and Zinde-Walsh's method of drawing random numbers from the AEPD distribution. Tadikamalla (1980) proposed accept-reject methods to draw random numbers from the exponential power (EPD) distribution. Zhu and Zinde-Walsh's method is better than Tadikamalla's because their method is to draw random numbers from the AEPD distribution that contains the EPD distribution as a special case and because they draw random numbers directly as transformation of gamma random numbers. The accept-reject methods are not as efficient as the direct method.

In Section 2.3 we estimate the nonlinear asymmetric GARCH (NGARCH) model with the error term distributed as an AEPD. I first derive the likelihood function and using daily data on S&P500 index I estimate the NGARCH model first by the maximum

likelihood estimation (MLE) method and evaluate the estimated regression. We point out that the estimated model does not track the real data well.

One of the key characteristics of the AEPD distribution is

$$\begin{cases} \Pr(X \leq \mu) &= \alpha \\ \Pr(X > \mu) &= 1-\alpha \end{cases}$$

where μ is the mode and $\alpha \in (0, 1)$ is the skewness parameter. If α is close to 0, we need a large sample size to estimate of p_1 and if α is close to 1, we need a large sample size to to estimate p_2. In the case of the NGARCH model, I will show that a sample size of over 60,000 to obtain e reliable estimate of p_2 when $\alpha = .99$.

After the MLE estimation of the NGARCH model I estimate the NGARCH model by Bayesian Markov Chain Monte Carlo (MCMC) methods. We use generated data and We derive Jeffrey's invariance prior. We transform the parameters so that the MCMC draws attain convergence quickly.

In the last subsection 3.4 I estimate the CKLS model with the AEPD error term following an ARMA-GARCH process. The model is estimated by MCMC methods. This model is presented as a possible alternative to the NGARCH model.

2.2 AEPD Distribution and Random Number Drawing

The probability density may be denoted as AEPD$(\alpha, p_1, p_2, \mu, \sigma)$ and it is given by

$$f_{AEP}(x|\beta) = \begin{cases} \left(\frac{\alpha}{\alpha^\star}\right) \frac{1}{\sigma} K_{EP}(p_1) \exp\left(-\frac{1}{p_1}\left|\frac{x-\mu}{2\alpha^\star\sigma}\right|^{p_1}\right), & \text{if } x \leq \mu \\ \left(\frac{1-\alpha}{1-\alpha^\star}\right) \frac{1}{\sigma} K_{EP}(p_2) \exp\left(-\frac{1}{p_2}\left|\frac{x-\mu}{2(1-\alpha^\star)\sigma}\right|^{p_2}\right), & \text{if } x > \mu \end{cases} \tag{2.1}$$

where $\beta = (\alpha, p_1, p_2, \mu, \sigma)^T$ is the parameter vector. $K_{EP}(p)$ is the normalizing constant and is defined by

$$K_{EP}(p) \equiv \frac{1}{2\, p^{1/p} \Gamma(1+1/p)}.$$

where $\alpha^\star$ is a function of parameters and defined by

$$\alpha^\star = \frac{\alpha K_{EP}(p_1)}{\alpha K_{EP}(p_1) + (1-\alpha) K_{EP}(p_2)}.$$

The parameters of the distribution, $\alpha,\ p_1,\ p_2,\ \mu$, and σ have the following interpretations:

1. $\mu \in R$ is the location of the mode.

2. $\sigma > 0$ is the density scale having the property that

$$\sigma = 1/2\{[E(|X-\mu|^{p_1}|X \leq \mu)]^{1/p_1} + [E(|X-\mu|^{p_2}|X \geq \mu)]^{1/p_2}\}.$$

3. $p_1 > 0$ and $p_2 > 0$ are the left and right tail parameters, respectively. However, as we discuss below, rather than saying that p_1 and p_2 are tail parameters, it is better to call them the kurtosis parameters.

4. $\alpha \in (0,1)$ is the skewness parameter with the properity:

$$P(X \leq \mu) = \alpha.$$

That is, the location μ is the α-quantile. For example, if $\alpha = .5$ the density is symmetric and the mode, μ, is the 50%-tile.[1]

Before interpreting p_1 and p_2 as the kurtosis parameters, which is the function of the fourth moment, we should discuss the first and second moments, or the mean and variance. Using the integral equation

$$\int_0^\infty x^{v-1}\exp(-\mu x^p)\mathrm{dx} = \frac{1}{p}\,\mu^{-v/p}\Gamma\left(\frac{v}{p}\right), \quad for \quad \mu > 0,\ v > 0,\ p > 0 \tag{2.2}$$

we can get the mean and variance of the random variable X which follow equation (2.1).

$$\omega = \mathrm{E(X)} = \frac{\sigma}{B}\left[(1-\alpha)^2\frac{p_2\Gamma(2/p_2)}{\Gamma^2(1/p_2)} - \alpha^2\frac{p_1\Gamma(2/p_1)}{\Gamma^2(1/p_1)}\right] + \mu \tag{2.3}$$

$$\begin{aligned}\delta^2 = \mathrm{Var(X)} &= \frac{\sigma^2}{B^2}\left[(1-\alpha)^3\frac{p_2^2\Gamma(3/p_2)}{\Gamma^3(1/p_2)} + \alpha^3\frac{p_1^2\Gamma(3/p_1)}{\Gamma^3(1/p_1)}\right] \\ &- \frac{\sigma^2}{B^2}\left[(1-\alpha)^2\frac{p_2\Gamma(2/p_2)}{\Gamma^2(1/p_2)} - \alpha^2\frac{p_1\Gamma(2/p_1)}{\Gamma^2(1/p_1)}\right]^2\end{aligned} \tag{2.4}$$

where B is a constant that is defined as

$$B = \alpha K_{EP}(p_1) + (1-\alpha)K_{EP}(p_2).$$

[1]It is worthwhile to note that μ is not $E(X)$ and σ is not $Var(X)$ neither.

We see that the mean ω is a function of all the five parameters $\alpha,\ p_1,\ p_2,\ \sigma$ and μ. The variance δ^2 is a function of four parameters $\alpha,\ p_1,\ p_2$ and σ.

The skewness measure η is customarily defined using the third moment:

$$\eta = \frac{\mathrm{E}(X - \mathrm{E}X)^3}{\delta^3},$$

and we see that η is a function of four parameters $\alpha,\ p_1,\ p_2$ and σ. Rather than using the skewness parameter η, it is more convenient to use α as the skewness measure. Since

$$\Pr(X \leq \mu) = \alpha$$

we see

$$\begin{cases} \alpha < .5 \implies & \text{right skewed} \\ \alpha = .5 \implies & \text{symmetric} \\ \alpha > .5 \implies & \text{left skewed.} \end{cases}$$

The kurtosis measure κ is usually defined using the fourth moment as

$$\kappa = \frac{\mathrm{E}(x - \mathrm{E}x)^4}{\delta^4}.$$

For an asymmetric distribution such as the AEPD, this conventional definition of kurtosis needs to be modified. Mineo (1989) defines the generalized kurtosis as

$$\frac{\mathrm{E}|X - \mu|^{2r}}{(\mathrm{E}|X - \mu|^r)^2}.$$

for any $r > 0$. Mineo and Ruggieri (2005) use the generalized kurtosis for the exponential power distribution. Zhu and Zinde-Welsh (2009) adopt the generalized kurtosis and define the left and right generalized kurtosis as

$$\mathrm{kur}_L(r) = \Gamma\left(\frac{1}{p_1}\right)\Gamma\left(\frac{2r+1}{p_1}\right)\Big/\Gamma^2\left(\frac{r+1}{p_1}\right),$$

$$\mathrm{kur}_R(r) = \Gamma\left(\frac{1}{p_2}\right)\Gamma\left(\frac{2r+1}{p_2}\right)\Big/\Gamma^2\left(\frac{r+1}{p_2}\right).$$

for any $r > 0$. If we set $r = 2$ then we get the kurtosis measures that correspond to

the conventional definition of kurtosis:

$$\text{kur}_L(2) = \Gamma\left(\frac{1}{p_1}\right)\Gamma\left(\frac{5}{p_1}\right)\Big/\Gamma^2\left(\frac{3}{p_1}\right),$$

$$\text{kur}_R(2) = \Gamma\left(\frac{1}{p_2}\right)\Gamma\left(\frac{5}{p_2}\right)\Big/\Gamma^2\left(\frac{3}{p_2}\right).$$

The left and right generalized kurtosis measures are functions only of p_1 and of p_2. Accordingly it is better to call p_1 and p_2 as the measures of kurtosis. If $p_1 = p_2 = p$ the left and right kurtosis measures are equal. If $p = 2$ then the kurtosis is 3 and it is the kurtosis of a normal distribution, and it is mesokurtic. If $p = 1$, then the left and right kurtosis measures are 6, which is the kurtosis of the Laplace distribution. The larger is the kurtosis measure, the sharper is the peakness of the distribution and the fatter is the tail distribution. The kurtosis measure is used in relation to the peakness of the normal distribution that has the kurtosis of 3, and it is called mesokurtic.

The table below presents the left and right kurtosis measures for different values of p_1 and p_2. When p_1 (p_2) is larger than 2, then the left (right) peakness is platikurtic, and when p_1 (p_2) is less than 2, the left (right) peakness is leptokurtic. The stylized fact of financial return data is that it is leptokurtic and slightly skewed.

Left and Right Kurtosis, $\text{k}_L(2)$ and $\text{k}_R(2)$			
p_1	p_2	left kurtosis $\text{k}_L(2)$	right kurtosis $\text{k}_R(2)$
.5	1	25.2	6.0
1	1	6.0	6.0
2	2	3.0	3.0
3	.7	2.42	11.06
4	1.5	2.19	3.76

In order to draw a random variate from AEPD with equation (2.1), we define

$$y = \begin{cases} \dfrac{1}{p_1}\left|\dfrac{x-\mu}{2\alpha^\star\sigma}\right|^{p_1}, & \text{if } x \le \mu \\ \dfrac{1}{p_2}\left|\dfrac{x-\mu}{2(1-\alpha^\star)\sigma}\right|^{p_2}, & \text{if } x > \mu \end{cases} \tag{2.5}$$

The random variable, y, can be drawn as follows. First we transform x in equation (2.1) into y using equation (2.5):

$$f(y|\beta) = \begin{cases} \alpha\,\Gamma\left(\frac{1}{p_1}\right)^{-1}\ y^{1/p_1-1}\exp(-y), & \text{with probability } P(X \le \mu) = \alpha \\ (1-\alpha)\,\Gamma\left(\frac{1}{p_2}\right)^{-1}\ y^{1/p_2-1}\exp(-y), & \text{with probability } P(X > \mu) = 1-\alpha \end{cases} \tag{2.6}$$

From equation (2.6) we see that the random variable Y is distributed as a mixture of the gamma distribution. Hence, in order to geneate an AEPD variate, X, first we draw three random numbers:

$$\begin{aligned} U &\sim \text{U}(0,\,1) \\ Y_1 &\sim \Gamma\left(\frac{1}{p_1}\right) \\ Y_2 &\sim \Gamma\left(\frac{1}{p_2}\right) \end{aligned}$$

where U(0,1) is the uniform distribution on (0,1), and $\Gamma(s)$ is the gamma distribution with parameter s. Then using the inverse of equation (2.5);

$$X = \begin{cases} \mu - \dfrac{\alpha\sigma}{B\Gamma(1+1/p_1)}Y_1^{1/p_1}, & \text{if } U < \alpha \\ \mu + \dfrac{(1-\alpha)\sigma}{B\Gamma(1+1/p_2)}Y^{1/p_2}, & \text{if } U > \alpha \end{cases} \tag{2.7}$$

the AEPD random variable X is drawn by

$$X = \mu + \frac{\alpha\sigma\,[\text{sign}(U-\alpha)-1]}{2B\Gamma(1+1/p_1)}Y_1^{1/p_1} + \frac{(1-\alpha)\sigma\,[\text{sign}(U-\alpha)+1]}{2B\Gamma(1+1/p_2)}Y_2^{1/p_2} \tag{2.8}$$

It is easier to obtain the mean and variance of X if we use Y. Let us obtain E(X) in equation (3) as follows: First we obtain

$$\begin{aligned} E(Y^{1/p}) &= \int_0^\infty y^{1/p}\Gamma(1/p)^{-1}y^{1/p-1}exp(-y)dy \\ &= \int_0^\infty \Gamma(1/p)^{-1}y^{2/p-1}exp(-y)dy \\ &= \frac{\Gamma(2/p)}{\Gamma(1/p)}\int_0^\infty y^{2/p}\Gamma(2/p)^{-1}y^{1/p-1}exp(-y)dy \\ &= \frac{\Gamma(2/p)}{\Gamma(1/p)} \end{aligned} \tag{2.9}$$

and $\Gamma(1+1/p) = 1/p\Gamma(1/p)$. Now using equation (2.8) and (2.9) we obtain

$$\mathrm{E(X)} \quad = \quad \frac{\sigma}{B}\left[(1-\alpha)^2\frac{p_2\Gamma(2/p_2)}{\Gamma^2(1/p_2)} - \alpha^2\frac{p_1\Gamma(2/p_1)}{\Gamma^2(1/p_1)}\right] + \mu$$

which is equation (2.3).[2] The variance of X that is given in (2.4) is similarly obtained.

Let us show that the AEPD random variables drawn using equation (2.8) yield the probability density function (2.1). Figure 2.1 presents the exact density and the kernel density estimate. We have drawn 5,000 AEPD random variables.

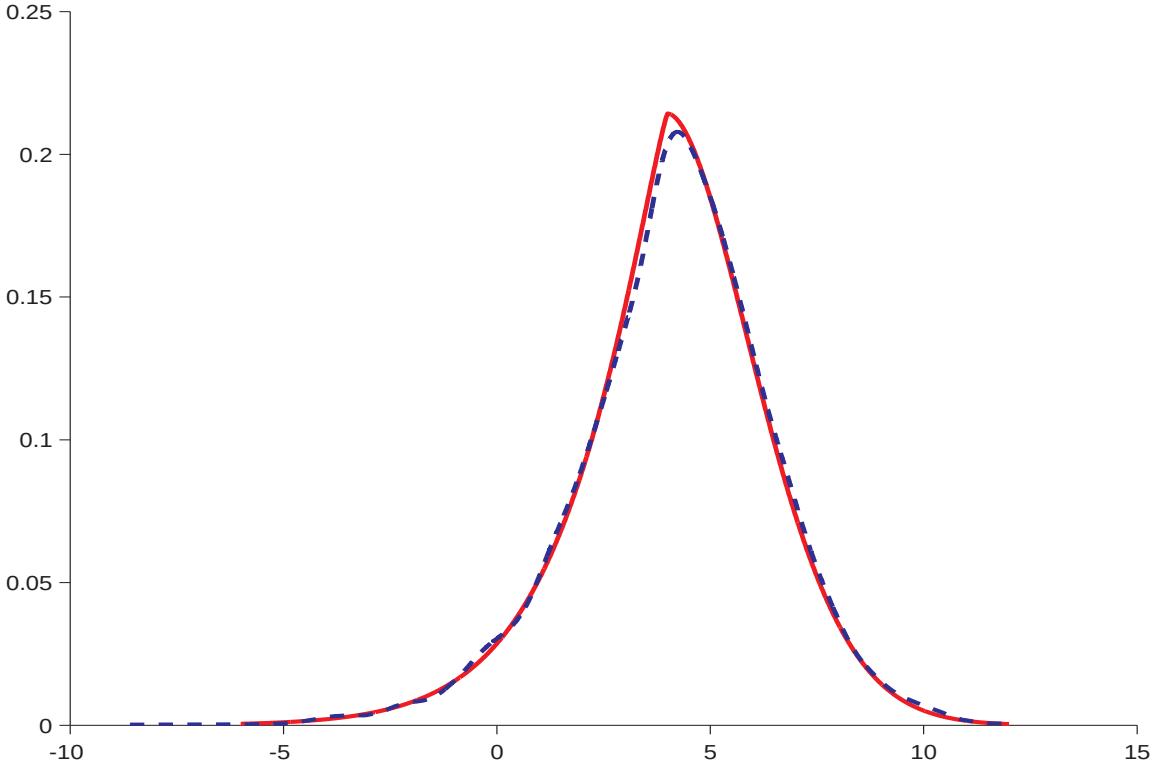

Figure 2.1: AEPD true density(red solid) and density estimation (blue dash)

[2]If $\alpha = .5$ and $p_1 = p_2$ then $\mathrm{E}X = \mu$.

The parameter settings for Figure 2.1 are

$$\begin{cases} \alpha &= .45 \\ p_1 &= 1.2 \\ p_2 &= 1.8 \\ \mu &= 4 \\ \sigma &= 2 \end{cases}$$

The following table compare the true mean and variance with those obtained by the draws of X:

	True	Sample
Mean of X	4.1357	4.1440
Variance of X	5.2069	5.3751

Nots True means true values computed by equations (2.3) and (2.4).

Sample means sample mean and variance from the draws of X by equation (2.8).

We see from Figure 2.1 and table above that the draws of X produces the density of X fairly well.

Table 2.2 below presents the parameters of the AEPD distribution that yield the normal, asymmetric Laplace, skewed exponential poweer and exponential power distributions.

Distribution	density function	parameters in APED
Normal	$\frac{1}{\sqrt{2\pi}\sigma} exp(-\frac{(x-\mu)^2}{2\sigma^2})$	$p_1 = p_2 = 2, \alpha = 1/2$
ALD	$\begin{cases} \frac{1}{2\sigma}\exp\left(-\left\|\frac{x-\mu}{2\alpha\sigma}\right\|\right), \text{if } x \le \mu \\ \frac{1}{2\sigma}\exp\left(-\left\|\frac{x-\mu}{2(1-\alpha)\sigma}\right\|\right), \text{if } x > \mu \end{cases}$	$p_1 = p_2 = 1$
SEPD	$\begin{cases} \frac{1}{\sigma}K_{EP}(p)\exp\left(-\frac{1}{p}\left\|\frac{x-\mu}{2\alpha\sigma}\right\|^p\right), \text{if } x \le \mu \\ \frac{1}{\sigma}K_{EP}(p)\exp\left(-\frac{1}{p}\left\|\frac{x-\mu}{2(1-\alpha)\sigma}\right\|^p\right), \text{if } x > \mu \end{cases}$	$p_1 = p_2$
EPD	$\frac{1}{\sigma}K_{EP}(p)\exp\left(-\frac{1}{p}\left\|\frac{x-\mu}{\sigma}\right\|^p\right)$	$p_1 = p_2, \alpha = 1/2$

Table 2.1: Denisties nested in AEPD

Figure 2.2 shows the four distributions: the normal, asymmetric Laplace (ALD), skewed exponential power (SED) and asymmetric exponential power distributions.

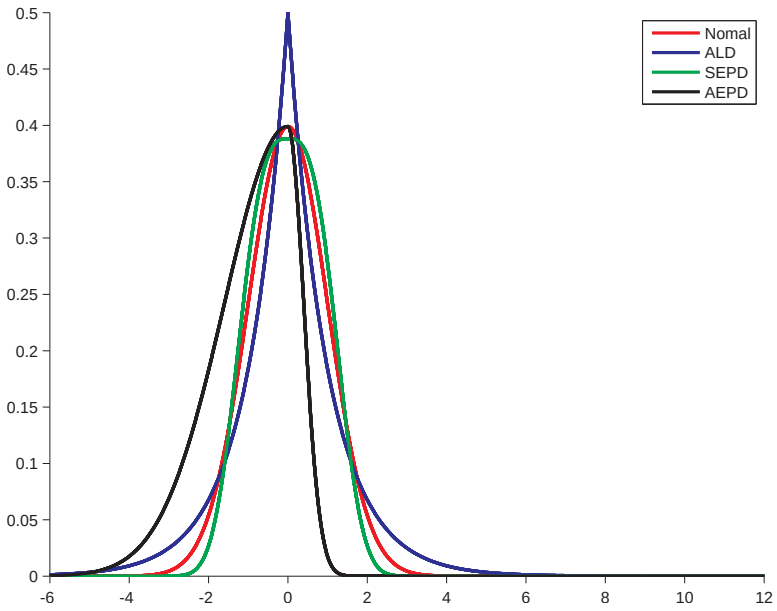

Figure 2.2: The graphs are for densities which are nested in AEPD

2.3 Estimation of AEPD-NGARCH Model by MLE

Zhu and Zinde-Walsh (2009) conduct Monte Carlo simulation experiments for the parameters in equation (1):

$$\mu,\ \sigma,\ \alpha,\ p_1,\ \text{and}\ \ p_2$$

under the following parameter settings:

Sample size T	500, 1,000, 2,000, 8,000
Parameters	
μ	0
σ	1
α	.3, .5
p_1	.7, 1.0, 1.5, 2.5
p_2	.7, 1.0, 1.5, 2.5

Notes: There are 32 combinations of parameters. Zhu and Zinde-Walsh report 16 combinations.

They use the MLE estimation procedure and make 2,000 replications ($N = 2,000$) for each of the 16 combinations of the parameters and for each sample size. They use the mean, M, and the standard deviation,STE, of the parameter estimates:

$$\begin{aligned} M(\widehat{\theta}) &= \frac{1}{N}\sum_{i=1}^{N}\widehat{\theta}^i \\ STD(\widehat{\theta}) &= \left(\frac{1}{N}\sum_{i=1}^{N}(\widehat{\theta}^i - M(\widehat{\theta}))^2\right)^2 \end{aligned}$$

where $\widehat{\theta}^i$ is the MLE of the i-th replication, $i = 1, \cdots, N$, and θ is the true value of the parameter θ.[3] Zhu and Zinde-Walsh conclude that the Monte Carlo experiment results are good: the MLE procedure works well. Instead of using equation (1) for our simulation experiments, we will use the nonlinear asymmetric GARCH(NGARCH) model of Engle and Ng (1993).

[3]Zhu and Zinde-Walsh should have used the the mean absolute deviation,$MADE$,

$$MAD = \frac{1}{N}\sum_{i=1}^{N}\left|\widehat{\theta}^i - \theta\right|$$

or the MSE

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(\widehat{\theta}^i - \theta\right)^2$$

rather than $M(\widehat{\theta})$ and $STD(\widehat{(\theta)})$.

2.3.1 AEPD-GARCH Model

The AEPD-NGARCH(1,1)model is given by

$$\begin{aligned} r_t &= m + \sigma_t z_t,\ z_t \sim i.i.d.\ \text{AEPD}(0,1) \\ \sigma_t^2 &= b_0 + b_1\sigma_{t-1}^2 + b_2\sigma_{t-1}^2(z_{t-1} - c)^2 \\ &= b_0 + b_1\sigma_{t-1}^2 + b_2(r_{t-1} - m - c\sigma_{t-1})^2 \end{aligned} \tag{2.10}$$

where r_t is asset return observations. z_t is distributed with AEPD(0,1) with $E(z_t) = 0$; $Var(z_t) = 1$. Model parameters are

$$\phi = (m, b_0, b_1, b_2, c, \alpha, p_1, p_2).$$

We exclude μ, and σ since they are not needed if z_t is distributed with zero mean and unit variance . Theoretically, the process r_t is a stationary process if the parameters in the state equation satisfy stationary conditions. i.e. $E(\sigma_t) = c, < \infty$. Define $\sigma_{t+j|t}^2 \equiv E_t(\sigma_{t+j}^2)$, we get the following equation for the conditional volatility in the model.

$$\sigma_{t+1|t}^2 = b_0 + b_1\sigma_t^2 + b_2(r_t - m - c\sigma_t)^2 \tag{2.11}$$

$$\sigma_{t+j|t}^2 = b_0 + [b_1 + b_2(1 + c^2)]\sigma_{t+j-1|t}^2, \quad j \geq 2 \tag{2.12}$$

Since σ_t cannot be negative, we put constraints on the parameters:

$$b_0 > 0;\ b_1 > 0;\ b_2 > 0$$

. and the conditional volatility process is stable process. We get

$$\text{E}(\sigma_{t+j|t}^2) = \text{E}(\sigma_{t+j-1|t}^2) = \frac{b_0}{1 - b_1 - b_2(1 + c^2)},$$

and $b_1 + b_2(1 + c^2) < 1,\ \text{E}(r_t) = m; \text{Var}(r_t) = \dfrac{b_0}{1 - b_1 - b_2(1 + c^2)}$.

2.3.2 Likelihood function for AEPD-NGARCH(1,1) Model

In order to get the likelihood function of the model, we need to derive the density of r_t. There is no closed form of the AEPD density with zero mean and unit variance. Suppose $X \sim AEPD(\alpha, p_1, p_2, 0, 1)$. i.e.

$$f_{AEP}(x|\beta) = \begin{cases} \left(\frac{\alpha}{\alpha^\star}\right) K_{EP}(p_1) \exp\left(-\frac{1}{p_1}\left|\frac{x}{2\alpha^\star}\right|^{p_1}\right), & \text{if } x \leq 0 \\ \left(\frac{1-\alpha}{1-\alpha^\star}\right) K_{EP}(p_2) \exp\left(-\frac{1}{p_2}\left|\frac{x}{2(1-\alpha^\star)}\right|^{p_2}\right), & \text{if } x > 0 \end{cases} \tag{2.13}$$

we know that

$$z_t = \frac{X-\omega}{\delta} \quad \text{where} \quad \omega = E(X), \quad \delta^2 = Var(X) \tag{2.14}$$

$$r_t = m + \sigma_t z_t \tag{2.15}$$

So, we get

$$X = \omega + \frac{r_t - m}{\sigma_t}\delta \quad and \quad \left|\frac{\partial X}{\partial r_t}\right| = \frac{\delta}{\sigma_t} \tag{2.16}$$

The likelihood function is derived from equation 2.16 as follows:

$$\begin{aligned} l(\phi; r) &= \prod_{t=1}^{T} \left[\left(\frac{\alpha}{\alpha^\star}\right)\frac{\delta}{\sigma_t} K_{EP}(p_1) \exp\left(-\frac{1}{p_1}\left|\frac{x(r_t)}{2\alpha^\star}\right|^{p_1}\right)\right]^{\mathrm{I}(x(r_t)\leq 0)} \\ &\times \prod_{t=1}^{T} \left[\left(\frac{1-\alpha}{1-\alpha^\star}\right)\frac{\delta}{\sigma_t} K_{EP}(p_2) \exp\left(-\frac{1}{p_2}\left|\frac{x(r_t)}{2(1-\alpha^\star)}\right|^{p_2}\right)\right]^{\mathrm{I}(x(r_t)>0)} \end{aligned} \tag{2.17}$$

where $r = (r_1, r_2, \cdots, r_T)^T$ is observation vector. $x(r_t)$ is a function of r_t as same as in 2.16. $\mathrm{I}(\cdot)$ is a indicator function. ϕ is defined as parameters group $(m, b_0, b_1, b_2, c, \alpha, p_1, p_2)$ which are to be estimated. $\omega, \delta,$ and $\alpha^\star$ are functions of parameters same as defined in previous part. σ_t is state variable defined in equation 2.10. From equation 2.17, the

log-likelihood function is obtained as follow.

$$
\begin{aligned}
L(\phi; r) &= \mathrm{T}\,\log(\delta) - \sum_{t=1}^{T} \log(\sigma_t) && (2.18)\\
&+ N\,(\log(\alpha) - \log(\alpha^*)) && (2.19)\\
&+ (\log(1-\alpha) - \log(1-\alpha^\star))\\
&+ \sum_{t=1}^{T} \left(-\frac{1}{p_1} \left| \frac{x(r_t)}{2\alpha^\star} \right|^{p_1} \right) \mathrm{I}(x(r_t) \le 0)\\
&+ \sum_{t=1}^{T} \left(-\frac{1}{p_2} \left| \frac{x(r_t)}{2(1-\alpha^\star)} \right|^{p_2} \right) \mathrm{I}(x(r_t) > 0) && (2.20)
\end{aligned}
$$

where N is the number of r_t that $x(r_t) \le 0, t = 1, 2, 3, \cdots, T$. i.e. N= $\sum_{t=1}^{T} \mathrm{I}(x(r_t) \le 0)$. $M = T - N$.

Zhu and Zinde-Walsh (2009) use daily returns on the S&P500 composite index from January 2 1990 to December 31 2002. Let us also use the S&P500 composite index data. We have downloaded the data from www.yahoo.com. The sample size is 3,280 (T=3,280). The following table compares their MLE estimates with ours:

$\beta =$	m	b_0	b_1	b_2	c	α	p_1	p_2
(1)	0.0254	0.0089	0.8918	0.0583	0.8802	0.4000	1.1820	1.8200
(2)	0.0252	0.0089	0.8918	0.0585	0.8765	0.4012	1.1831	1.8148

Table 2.2: Estimation results from S&P500 composite index for the period from January 2, 1990 to December 31, 2002 on NGARCH-AEPD model. (1) is estimation result in Zhu and Zinde-walsh (2009) and (2) is the estimation result from our ML estimation.

The slight differences between their estimates and ours may be due to slight differences in data entry as well as to the MLE algorithms. We used MATLAB function, *fmincon*, that is a flexible optimization function.[4]

There are some problems with the estimated NGARCH-ARCH model. They are

1. Since r_t is stationary process, E(r_t) should be close to m. However, the sample

[4]MATLAB *fmincon* also provides an access method to different Hessian function.

mean of daily returns on the S&P500 composite index for the period from January 2, 1990 to December 31, 2002 is 0.0328. The estimator of m is 0.0254. The difference is significant. The estimator of m can not be the predictor of daily returns in the future.

2. The unconditional variance of r_t should be close to

$$Var(r_t) = \frac{b_0}{1 - b_1 - b_2(1 + c^2)}$$

that is given in section (3.1). However, the sample variance of daily returns on the S&P500 composite index is 1.1046. The theoretical mean of the conditional variance from the estimation is 1.8808. The difference is also significant. The unconditional volatility does not work as the measure of future risk.

3. We have obtained the simulated data using the MLE parameter estimates. The simulated data differ from the real data dramatically. In the simulation example, we have set the unconditional variance

$$\frac{b_0}{1 - b_1 - b_2(1 + c^2)}$$

as the one step the first 2,000 simulation data and obtained 3,280 simulation data. Comparison of two sets of data is in Figure 2.3.2.

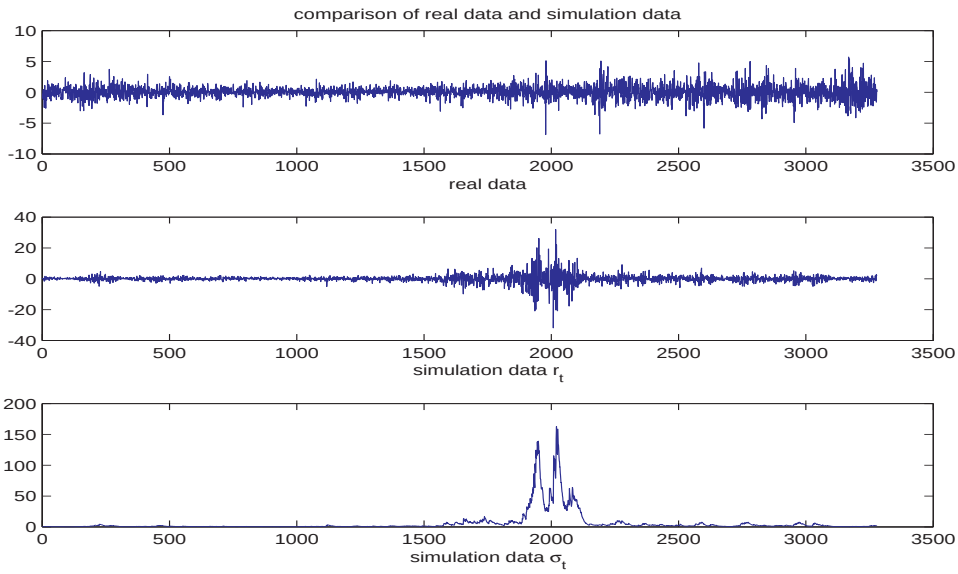

Figure 2.3: comparison of real data and simulation data that are generated from the MLE estimators

In the first panel of Figure the real data is plotted, while the simulated (*i.e.* fitted) data is presented in the second panel. The conditional volatility σ_t is plotted in the third panel. The sample mean of the simulated data is -0.0116 and the sample mean of the conditional volatility is 6.0343. Both of them are far away from the theoretical values.

One conclusion we may draw is that although the MLE procedure using the S&P500 data converges for the NGARCH-AEPD model, the NGARCH-AEPD model may not be an appropriate specification for the S&P500 data. As an alternative model specification we will introduce the CKLS-ARMA-GARCH model in the next section, albeit we use different data set.

2.3.3 MLE Estimation when the skewness parameter α is either close to 1 or close to 0

In equation (6) we have noted that

$$\begin{cases} Pr(X \leq \mu) &= \alpha \\ Pr(X > \mu) &= 1 - \alpha \end{cases}$$

Suppose that $\alpha = .99$. Then the probability of X is less than or equal to μ is .99. The AEPD distribution is highly skewed to the left. Examining the pdf of X we see that there is a very small number of observations to estimate the kurtosis parameter p_2. This means that p_2 is virtually unidentifiable even with a reasonable number of observations.

Let us demonstrate how large the sample size has to be so that we can obtain a reliable MLE estimate of p_2. We have made Monte Carlo experiments by setting the true parameters to be

$$\begin{cases} m &= 0.0254 \\ b_0 &= 0.0089 \\ b_1 &= 0.8918 \\ b_2 &= 0.0583 \\ c &= 0.8802 \\ \alpha &= 0.99 \\ p_1 &= 1.182 \\ p_2 &= 1.820 \end{cases}$$

We vary the sample size from 500 to 63,280. Given a sample size we replicate the sample N times. As the sample size increases we reduced the replication size to obtain the Monte Carlo experiments within a manageable time. The following table presents the results.

β	true value	estimation with sample size(T)				
		500(N=2000[1])	3280(2000)	23280(100)	43280(50)	63280(50)
m	0.0254	0.0539	0.0338	0.0289	0.0260	0.0270
b_0	0.0089	0.0100	0.0486	0.0085	0.0095	0.0088
b_1	0.8918	0.8922	0.8916	0.8918	0.8924	0.8918
b_2	0.0583	0.0550	0.0564	0.0576	0.0578	0.0611
c	0.8802	0.8595	0.8860	0.8777	0.8782	0.8278
α	0.99	0.9912	0.9779	0.9842	0.9877	0.9846
p_1	1.182	1.2512	1.2305	1.2022	1.2011	1.1823
p_2	1.820	13.3882	7.5564	3.1094	2.0805	1.8895

Table 2.3: parameter values used for the simulation experiments

2.3.4 Bayesian Estimation of the NGARCH-AEPD Model

In this section, we design Bayesian algorithms for the NGARCH-AEPD model.We conduct a series of estimation experiments on simulated data. First, we set the posterior function for the model. Then we use the probability integral transformation (PIT) method and Metropolis-Hastings method to estimate the parameters of the NGARCH-AEPD model.

Our experiments show that the estimation results are similar to those obtained by the MLE procedure. When the sample size is small, the bias of the estimation is significant. When the skewness is too large, the accuracy of the estimation increases as the sample size increase. We have also found that when the skewness parameter α is in the polar region (*i.e.* close to 0 o 1), the PIT method can not locate the inverse the CDF at the point of a unit random variate. In this case, a large number of grids is necessary to increase integration accuracy. However, because of the increase of the number of grids and because of the large sample size, computation time increases dramatically.

In our Bayesian algorithm, we use Jeffreys' noninformative prior for α, p_1 and p_2.

[1]N is the number of MLE estimation replications

Zhu and Zinde-Walsh(2009) give the Fisher information matrix for the AEPD model, and it is given by

$$p(\alpha,p_1,p_2)=\begin{vmatrix} \frac{p_1+1}{\alpha}+\frac{p_2+1}{1-\alpha} & -\frac{1}{p_1} & \frac{1}{p_2} \\ -\frac{1}{p_1} & \frac{\alpha}{p_1^3}(1+\frac{1}{p_1})\Psi'(1+\frac{1}{p_1}) & 0 \\ \frac{1}{p_2} & 0 & \frac{1-\alpha}{p_2^3}(1+\frac{1}{p_2})\Psi'(1+\frac{1}{p_2}) \end{vmatrix} \tag{2.21}$$

The noninformative reference prior distributions for other parameters are.

$$\begin{aligned} m \sim & N(\mu_m,\sigma_m^2) \\ \mathbf{log}(b_0) \sim & N(\mu_{b_0},\sigma_{b_0}^2) \\ (b_1,b_2(1+c^2)) \sim & Dirichlet(\gamma_1,\gamma_2,\gamma_3) \\ \textbf{i.e. } p(b_1,b_2,c) = & b_1^{\gamma 1}(b_2(1+c^2))^{\gamma 2}(1-b_1-b_2(1+c^2))^{\gamma 3} \end{aligned} \tag{2.22}$$

where μ_m,σ_m^2, μ_{b_0}, $\sigma_{b_0}^2$, γ_1, γ_2 and γ_3 are hyperparameters that are assumed to be known.

In order to have the unbounded support for the parameters, we re-parameterize them:

$$\lambda=\begin{bmatrix} \ddot{m} \\ \ddot{b_0} \\ \ddot{b_1} \\ \ddot{b_2} \\ \ddot{c} \\ \ddot{\alpha} \\ \ddot{p_1} \\ \ddot{p_2} \end{bmatrix} = \begin{bmatrix} m \\ \mathbf{log}(b_0) \\ \mathbf{log}(\frac{b_1}{1-b_1-b_2(1+c^2)}) \\ \mathbf{log}(\frac{b_2}{1-b_1-b_2(1+c^2)}) \\ c \\ \mathbf{log}(\frac{\alpha}{1-\alpha}) \\ \mathbf{log}(p_1) \\ \mathbf{log}(p_2) \end{bmatrix} \tag{2.23}$$

So we get the ϕ as a function of λ:

$$\phi = \begin{bmatrix} m \\ b_0 \\ b_1 \\ b_2 \\ c \\ \alpha \\ p_1 \\ p_2 \end{bmatrix} = \mathbf{g}(\lambda) = \begin{bmatrix} m \\ \mathbf{exp}(\ddot{b_0}) \\ \frac{\mathbf{exp}(\ddot{b_1})}{\mathbf{exp}(\ddot{b_2})+\mathbf{exp}(\ddot{b_1})+\mathbf{exp}(\ddot{b_2})\ddot{c}^2+1} \\ \frac{\mathbf{exp}(\ddot{b_2})}{\mathbf{exp}(\ddot{b_2})+\mathbf{exp}(\ddot{b_1})+\mathbf{exp}(\ddot{b_2})\ddot{c}^2+1} \\ \ddot{c} \\ \frac{\mathbf{exp}(\ddot{\alpha})}{1+\mathbf{exp}(\ddot{\alpha})} \\ \mathbf{exp}(\ddot{p_1}) \\ \mathbf{exp}(\ddot{p_2}) \end{bmatrix} \tag{2.24}$$

So, we get Jacobian transformation matrix:

$$\frac{\partial \phi}{\partial \lambda} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{exp}(\ddot{b_0}) & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{\partial b_1}{\partial \ddot{b_1}} & \frac{\partial b_1}{\partial \ddot{b_2}} & \frac{\partial b_1}{\partial \ddot{c}} & 0 & 0 & 0 \\ 0 & 0 & \frac{\partial b_2}{\partial \ddot{b_1}} & \frac{\partial b_2}{\partial \ddot{b_2}} & \frac{\partial b_2}{\partial \ddot{c}} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{\mathbf{exp}(\ddot{\alpha})}{(1+\mathbf{exp}(\ddot{\alpha}))^2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{exp}(\ddot{p_1}) & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{exp}(\ddot{p_2}) \end{bmatrix} \tag{2.25}$$

where

$$\begin{aligned} \frac{\partial b_1}{\partial \ddot{b_1}} &= \frac{\mathbf{exp}(\ddot{b_1})(\mathbf{exp}(\ddot{b_2}) + \mathbf{exp}(\ddot{b_2})\ddot{c}^2 + 1)}{(\mathbf{exp}(\ddot{b_2}) + \mathbf{exp}(\ddot{b_1}) + \mathbf{exp}(\ddot{b_2})\ddot{c}^2 + 1)^2} \\ \frac{\partial b_1}{\partial \ddot{b_2}} &= -\frac{\mathbf{exp}(\ddot{b_1})(\mathbf{exp}(\ddot{b_2}) + \mathbf{exp}(\ddot{b_2})\ddot{c}^2)}{(\mathbf{exp}(\ddot{b_2}) + \mathbf{exp}(\ddot{b_1}) + \mathbf{exp}(\ddot{b_2})\ddot{c}^2 + 1)^2} \\ \frac{\partial b_1}{\partial \ddot{c}} &= -\frac{2\mathbf{exp}(\ddot{b_1})\mathbf{exp}(\ddot{b_2})\ddot{c}^2}{(\mathbf{exp}(\ddot{b_2}) + \mathbf{exp}(\ddot{b_1}) + \mathbf{exp}(\ddot{b_2})\ddot{c}^2 + 1)^2} \\ \frac{\partial b_2}{\partial \ddot{b_1}} &= -\frac{\mathbf{exp}(\ddot{b_1})\mathbf{exp}(\ddot{b_2})}{(\mathbf{exp}(\ddot{b_2}) + \mathbf{exp}(\ddot{b_1}) + \mathbf{exp}(\ddot{b_2})\ddot{c}^2 + 1)^2} \\ \frac{\partial b_2}{\partial \ddot{b_2}} &= \frac{\mathbf{exp}(\ddot{b_2})(\mathbf{exp}(\ddot{b_1}) + 1)}{(\mathbf{exp}(\ddot{b_2}) + \mathbf{exp}(\ddot{b_1}) + \mathbf{exp}(\ddot{b_2})\ddot{c}^2 + 1)^2} \\ \frac{\partial b_2}{\partial \ddot{c}} &= -\frac{2\mathbf{exp}(2\ddot{b_2})\ddot{c}^2}{(\mathbf{exp}(\ddot{b_2}) + \mathbf{exp}(\ddot{b_1}) + \mathbf{exp}(\ddot{b_2})\ddot{c}^2 + 1)^2} \end{aligned} \tag{2.26}$$

The determinant of the Jacobian transformation matrix is

$$\left|\frac{\partial\phi}{\partial\lambda}\right| = \mathbf{exp}(\ddot{b_0})\frac{\mathbf{exp}(\ddot{b_1})\mathbf{exp}(\ddot{b_2})}{(\mathbf{exp}(\ddot{b_2})+\mathbf{exp}(\ddot{b_1})+\mathbf{exp}(\ddot{b_2})\ddot{c}^2+1)^3}\frac{\mathbf{exp}(\ddot{\alpha})}{(1+\mathbf{exp}(\ddot{\alpha}))^2}\mathbf{exp}(\ddot{p_1})\mathbf{exp}(\ddot{p_1}) \quad (2.27)$$

From Equation 2.21, Equation 2.22, Equation 2.27 and the log-likelihood function in Equation 2.20, we get the posterior:

$$p(\lambda|r) = L(\psi;r)p(\alpha,p_1,p_2)p(m)p(b_0)p(b_1,b_2,c)\left|\frac{\partial\phi}{\partial\lambda}\right| \quad (2.28)$$

In our experiments, we use both Metropolis-Hastings algorithm and PIT method to analysis simulated data. Both methods work well.

In the Metropolis-Hastings method, we use the independent random walk. If parameter α is not close to 0 or 1, we choose information matrix valuated at MLE estimators time a constant C as the proposal variance as in Equation 2.29. If the parameter α is close to 0 or 1, MLE estimators of parameter p_1 or p_2 are biased far from their true value. $\hat{\Sigma}$ is not appropriate for the proposal variance. In this case, we have to tune proposal variance according optimal accept rate.

$$\lambda^{(i)} \sim N(\lambda^{(i-1)}, C\hat{\Sigma}) \quad (2.29)$$

$$\hat{\Sigma} = -\left\{\left[\frac{\partial^2\mathbf{log}p(\lambda|r)}{\partial\lambda\partial\lambda'}\right]_{\lambda=\hat{\lambda}}\right\}^{-1} \quad (2.30)$$

where $\hat{\lambda}$ is the MLE estimator of λ. Since it is hard to get the close form of the Hessian matrix of λ in this model, in computation practice, we can get the approximation by differentiating the posterior.

The Bayesian estimation results are in Table 2.4 and in Figure 2.4 and Figure 2.5.

$\beta =$	m	b_0	b_1	b_2	c	α	p_1	p_2
true value	0.0254	0.0079	0.8918	0.0583	0.8802	0.5000	1.1820	1.8200
estimation	0.0338	0.0104	0.8959	0.0614	0.8759	0.5074	1.2385	1.6289

Table 2.4: MCMC simulation example with $\alpha = 0.5$

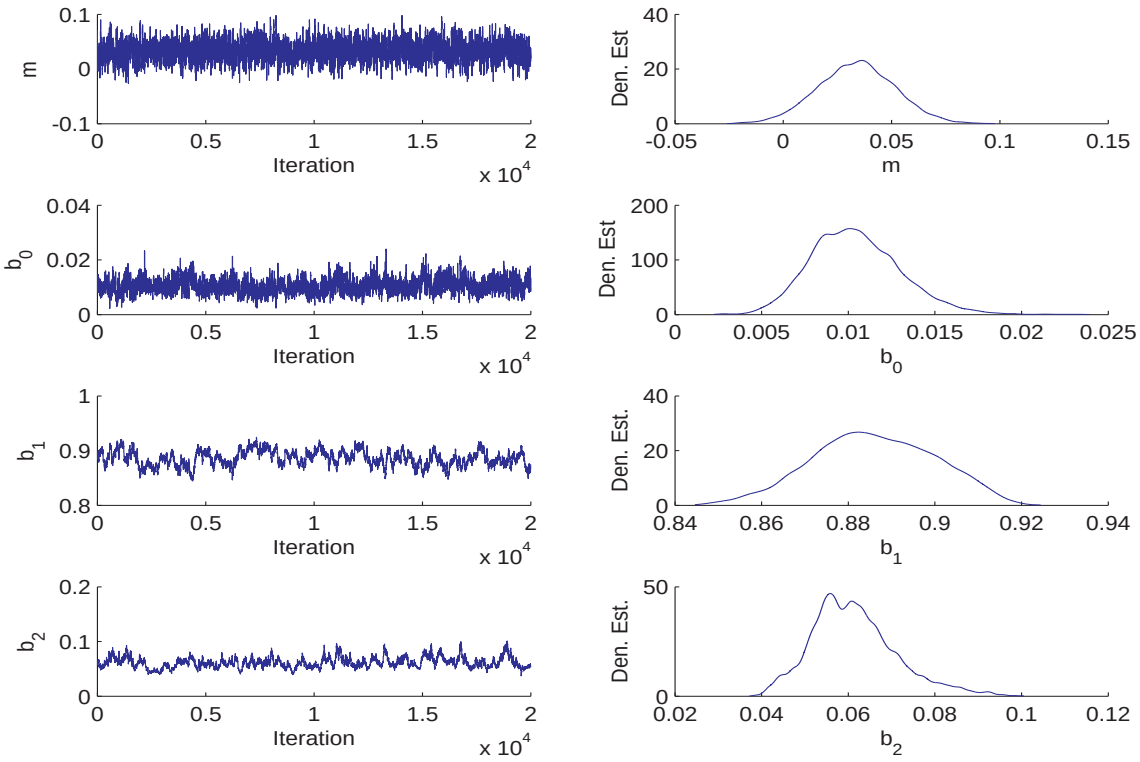

Figure 2.4: graph for estimators of c, α, p_1, p_2 from MCMC estimation on NGARCH-AEPD model with α=0.5.

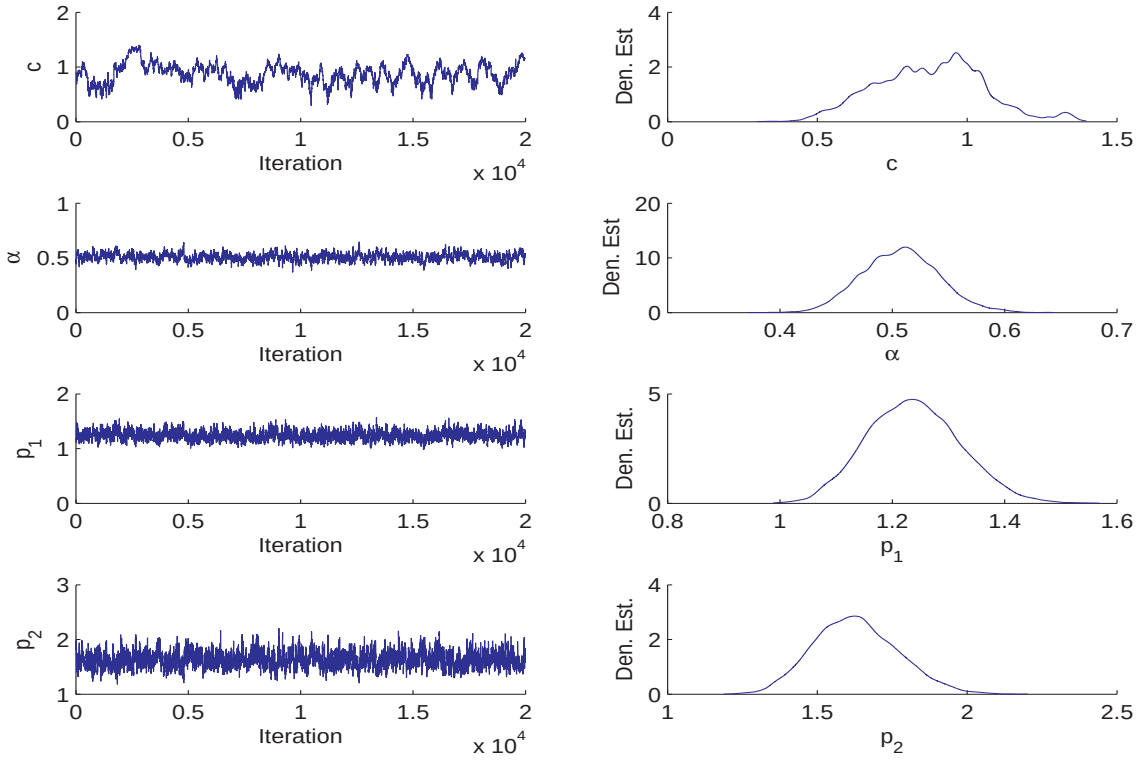

Figure 2.5: graph for estimators of c, α, p_1, p_2 from MCMC estimation on NGARCH-AEPD model with α=0.5.

The advantage of the PIT method is that the optimal proposal variance is not needed. However, it is hard to get the closed form of the CDF of posterior density. We have to resort to a numerical method such as Simpson's rule. That requires a lot of computation time to locate the inverse of the CDF function that equals to a uniform random variate. In simulation experiment, our experience show that if skewness parameter α is close to 1, we have to increase the number of grids in integration by Simpson's rule. Otherwise, the integrable area becomes smaller and smaller and the PIT algorithm crashes. In one of the examples we set $\alpha = 0.97$ to simulate the data. When we used the PIT algorithm, the number of grids for p_2 is set at 2000 (200 for other parameters) and the sample size is 20,000. The estimation of p_2 is close to the true value. However, the computation time is about 12 hours when the number of MCMC draws is 5,000. In our simulation experiments, we also find that when the sample size is small (for example, T=500), all of the three estimations from three methods: the MLE,

Metropolis-Hastings and PIT have big biases from the true value. The problem lies in the filtration of the conditional volatility. The filtering is started from the unconditional variance. When the sample size is small, the unconditional variance is not close to its unconditional one. Then all the filtered σ_t^2s deviate from the true values.

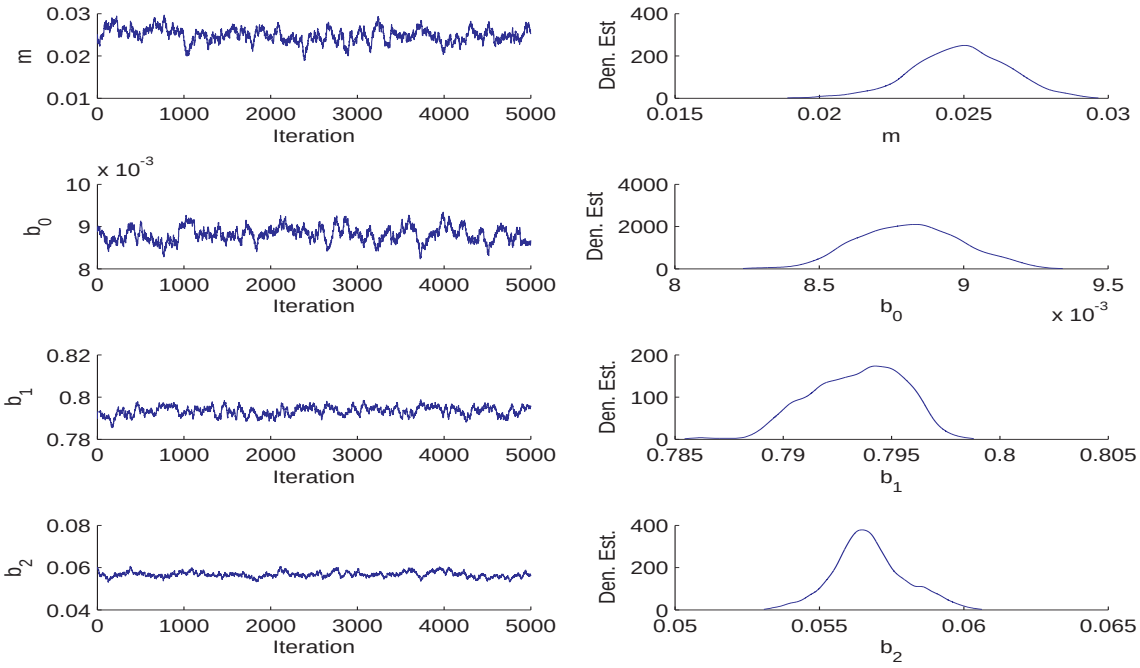

Figure 2.6: graph for estimators of m, b_0, b_1, b_2 from PIT estimation on NGARCH-AEPD model with α=0.97.

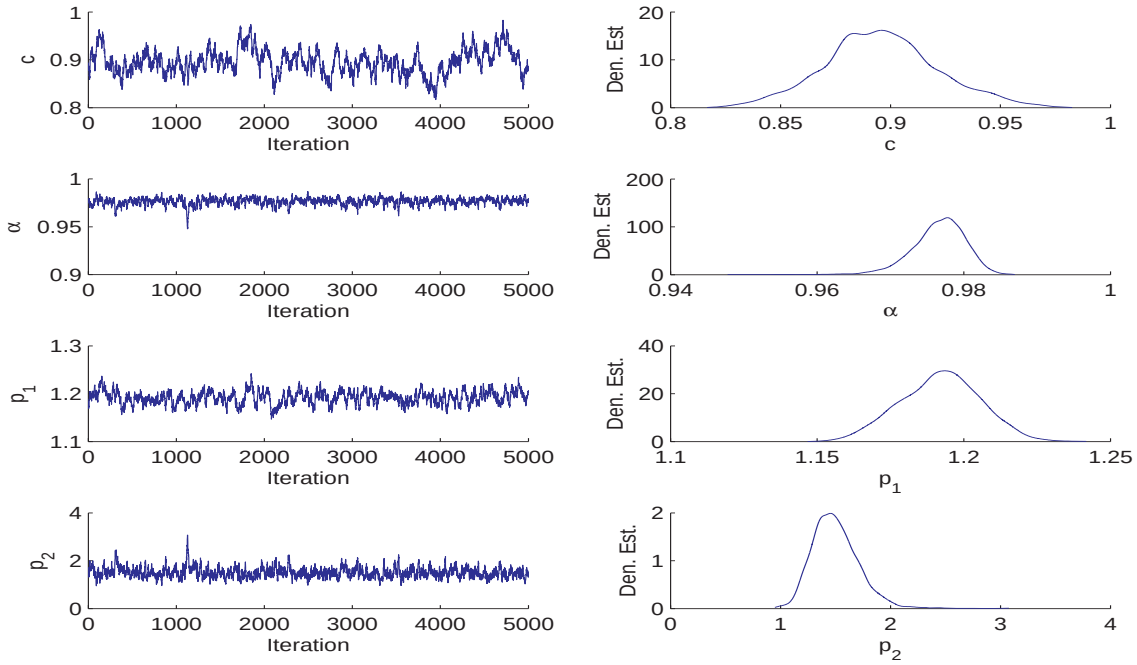

Figure 2.7: graph for estimators of c, α, p_1, p_2 from PIT estimation on NGARCH-AEPD model with α=0.97.

2.4 CKLS-ARMA-GARCH-AEPD Model

In the previous section we pointed out that the NGARCH-AEPD model may not be a reasonable specification to employ. In this section we illustrate that there is another way of specifying a GARCH model. The model we illustrate is the CKLS model with an ARMA-GARCH error term.

In her dissertation Liuling Li (2009) estimated the CKLS model with an ARMA-GARCH error term. In her dissertation the error term is assumed to be normal or symmetric exponential power distribution. Since the financial return data are usually leptokurtic and slightly skewed, it will be better to use the AEPD distribution for the error term. Accordingly, we use the AEPD distribution and call the model CKLS-ARMA-GARCH-AEPD Model.

The CKLS-ARMA-GARCH-AEPD model is given by

$$
\begin{aligned}
r_t &= a + b\, r_{t-1} + r_{t-1}^{c} u_t \\
u_t &= \sum_{j=1}^{p} \phi_j u_{t-j} + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} + \sigma_t \epsilon_t \\
\sigma_t^2 &= \alpha_0 + \sum_{j=1}^{r} \alpha_j \epsilon_{t-j}^2 + \sum_{j=1}^{s} \beta_j \sigma_{t-j}^2 \qquad (2.31) \\
\alpha_0 &> 0, \;\; \alpha_j \geq 0, \; j = 1, \cdots, r \quad \beta_j \geq 0, \;\; j = 1, \cdots, s \\
1 &\geq \sum_{j=1}^{\max(r,s)} (\alpha_j + \beta_j)
\end{aligned}
$$

where ϵ_t follows an AEPD distribution.

The posterior pdf of the CKLS-ARMA-GARCH-AEPD model is given by

$$
p(\boldsymbol{\Delta}|\text{data}) \propto p(\boldsymbol{\Delta}) \prod_{t=1}^{n} r_{t-1}^{-c}\, \sigma_t^{-1} \begin{cases} \exp\left\{ -\frac{1}{p_1} \left| \frac{\epsilon_t}{2\,\alpha\, K(p_1)} \right|^{p_1} \right\}, & \text{if } \epsilon_t \leq 0 \\ \exp\left\{ -\frac{1}{p_2} \left| \frac{\epsilon_t}{2\,(1-\alpha)\, K(p_2)} \right|^{p_2} \right\}, & \text{if } \epsilon_t > 0 \end{cases} \qquad (2.32)
$$

where

$$
\begin{aligned}
\boldsymbol{\Delta} &= \left\{ a,\; b,\; c,\; \boldsymbol{\Phi}^{,},\; \boldsymbol{\Theta}^{'},\; \boldsymbol{\Lambda}^{'},\; \alpha,\; p_1,\; p_2 \right\} \\
\boldsymbol{\Phi} &= (\phi_1, \cdots, \phi_p)^{'} \\
\boldsymbol{\Theta} &= (\theta_1, \cdots, \theta_2)^{'} \\
\boldsymbol{\Gamma} &= (\alpha_1, \cdots, \alpha_r)^{'} \\
\boldsymbol{\Lambda} &= (\beta_1, \cdots, \beta_s)^{'} \\
\epsilon_t &= \frac{1}{\sigma_t} \left[u_t - \sum_{j=1}^{p} \phi_j u_{t-j} - \sum_{j=1}^{q} \theta_j \epsilon_{t-j} \right] \\
u_t &= \frac{r_t - a - b\, r_{t-1}}{r_{t-1}^{c}},
\end{aligned}
$$

$t = 1, \cdots, n$, and we set $\epsilon_0 = \cdots = \epsilon_{-q} = 0$. We set the prior, $p(\mathbf{\Delta})$, as

$$\begin{aligned} p(\mathbf{\Delta}) &= N(\overline{a}, \overline{\sigma}_a^2) \times N(\overline{b}, \overline{\sigma}_b^2) \times N(\overline{c}, \overline{\sigma}_c^2) \times N(\overline{\Phi}, \overline{\sigma}_\phi^2 I) \\ &\quad \times N(\overline{\Theta}, \overline{\sigma}_\theta^2 I) \times N(\overline{\alpha}_0, \overline{\sigma}_{\alpha_0}^2)\, I(\alpha_0 > 0) \times N(\overline{\Gamma}, \overline{\sigma}_\Gamma^2 I)\, I(\Gamma > 0) \\ &\quad \times N(\overline{\Lambda}, \overline{\sigma}_\Lambda^2 I)\, I(\Lambda > 0) \times N(\overline{\alpha}, \overline{\sigma}_\alpha^2)\, I(0 < \alpha < 1) \\ &\quad \times N(\overline{p}_1, \overline{\sigma}_{p_1}^2)\, I(p_1 > 0) \times N(\overline{p_2}, \overline{\sigma}_{p_2}^2)\, I(p_2 > 0) \end{aligned}$$

We make a simulated data of sample size $n = 1,000$. Table 2.5 presents the posterior means and posterior standard deviations of the parameters for different numbers of MCMC draws. The major features of the MCMC algorithms we employ are

1. We use the PIT algorithms for drawing the parameters of the AEPD distribution: c, α, p_1, and p_2.

2. We use the efficient jump draws for parameters a, b, Θ and Λ rather than the nonlinear maximization procedure used by Nakatsuma (2000).

3. To increase computational speed, we use MATLAB to call C++ program.

Table 2.5: Summary Statistics of MCMC Draws

parameter	true value	m=1,000 pm (psd)	m=30,000 pm (psd)	m=50,000 pm (psd)	m=80,000 pm (psd)
a	.2	.2975	.3207	.3201	.3079
		(.1292)	(.1399)	(.1314)	(.1295)
b	.4	.4078	.4011	.4004	.4613
		(.2359)	(.2623)	(.2228)	(.2803)
ϕ_1	.4	.5977	.5918	.5846	.6171
		(.2591)	(.2473)	(.2905)	(.2572)
θ_1	.4	.5766	.5843	.5636	.5699
		(.2803)	(.2777)	(.2792)	(.2896)
α_0	.7	.8541	.8212	.8440	.8232
		(.3895)	(.3819)	(.3621)	(.4146)
α_1	.7	.7356	.7400	.7426	.7460
		(.1684)	(.1507)	(.1675)	(.1648)
β_1	.2	.1317	.1292	.1526	.1292
		(.1208)	(.1084)	(.1413)	(.1183)
α	.4	.4404	.4399	.4406	.4500
		(.0474)	(.0559)	(.0570)	(.0536)
p_1	1	.8677	.8731	.8682	.8745
		(.1661)	(.1803)	(.1731)	(.1792)
p_2	1.2	.9308	.9457	.9491	.(231
		(.1642)	(.1553)	(.1536)	(.1578)
c	.5	.5412	.5349	.5380	.5341
		(.0326)	(.0415)	(.0401)	(.0379)

Notes pm = posterior mean; psd = posterior standard deviation;
m = number of MCMC draws after the burns.

Table 2.6 present the Kolmogorov-Smirnov test (KST)and filtered Kolmogorov-Smirnov test (FKST) for convergence of the MCMC draws. These tests are given in Goldman, Valieva and Tsurumi (2008). They advise that the convergence of the MCMC draws should be judged not only by the Kolmogorov-Smirnov tests, but also by the plot of the MCMC draws of a parameter. We checked the plots and we see that for m $\geq$ 30,000, the plots exhibit convergence patterns of random fluctuations along the posterior means.

From Tables 2.5 and 2.6 we observe

1. With m=1,000 both the KST and FKST reject the null hypothesis of convergence. For m=30,000 KST accept the null hypothesis except for α and p_1 if we set the significance level to be .05. The FKST accepts the null hypothesis for m $\geq$ 30,000.

2. the posterior means and standard deviations are more or less the same for all m. Even as little as m=1,000, they are not different from the posterior means and standard deviations of m $\geq$ 30,000. Goldman, Valieva and Tsurumi (2008) state that even the FKST tends to be conservative and that if we are only getting the summary statistics of posterior means and standard deviations, the MCMC draws often need not be large. As soon as the plot of MCMC draws of a parameter show random fluctuations around the mean, we may stop MCMC drawing. Judged from the plots, we conclude that m=30,000 is large enough a number of draws.

3. The posterior means of the parameters are fairly close to the true values.

Table 2.6: P-values of Kolmogorov-Smirnov Test

	m=1,000		m=30,000		m=50,000		m=80,000	
	KST	FKST	KST	FKST	KST	FKST	KST	FKST
a	.0001	.1497	.5812	.9922	.9883	.9135	.9938	.9062
b	.0146	.1979	.9510	.7260	.9135	.7591	.9062	.6994
ϕ_1	0	0	.5812	.9510	.7591	.9883	.4676	.9938
θ_1	0	0	.9510	.8580	.9135	.9135	.9938	.9062
α_0	.0009	.0241	.3272	.6571	.5727	.9999	.6994	.9062
α_1	.0002	.0012	.8580	1.0	.9883	.9883	.9938	.9938
β_1	0	0	.2323	.8512	.1641	.9135	.2809	.9938
α	0	0	.0159	.3272	.7591	.7591	.4676	.2800
p_1	.2262	.1497	.0438	.7260	.7591	.9883	.9062	.6994
p_2	0	0	.9998	.9510	.9883	.9883	.0783	.2809
c	.5085	0	.8580	.9922	.9135	.7591	.9938	.9938

Notes: KST= Kolmogorov-Smirnov test

FKST= Filtered Kolmogorov-Smirnov test

2.5 Concluding Remarks

In this chapter we discussed the regression model whose error term follows the asymmetric exponential power distribution (AEPD). We presented a straightforward way to draw random numbers from the AEPD. Using the daily return data on the S&P500 composite index from January 2 1990 to December 31 2002, we estimated the AEPD-NGARCH(1,1) model by the maximum likelihood procedure.

We computed the fitted variable over the sample period and found that the AEPD-NGARCH(1,1) does not explain the daily S&P500 composite index. As an alternative to AEPD-NGARCH(1,1) model we introduced the CKLS-ARMA-GARCH-AEPD model. The ARMA process tends to smooth out the volatility and thus this model may explain

the data better than the AEPD-NGARCH(1,1) model.

Using Monte Carlo experiments we showed that when the skewness parameter α is either close to 1 or close to 0, the estimation of the kurtosis parameters, p_1 and p_2 is virtually unidentifiable unless the sample size is extremely large: in our experiments the sample size of 70,000 is necessary to obtain a reliable estimate of p_1 or p_2.

Chapter 3

Combining MATLAB with C/C^{++} and with Graphic Processor Unit (GPU): Examples of Matrix Multiplication and Kernel Density Estimation

3.1 Introduction

In discussing the CKLS-ARMA-GARCH-AEPD model we used MATLAB coupled with C^{++} to reduce computational time considerably. In this chapter we will dicuss methods to speed up computation using MATLAB coupled with C^{++} and graphic processing unit (GPU)[1].

Economic models are becoming more and more complex and available data sets are getting larger and larger. Complex models need powerful computing to mine useful information. Although computing power has increased dramatically in the last decades, the CPU has reached its clock speed limitation due to power and heat restriction as well as the physical limit of CPU.

The computer manufacturers have been forced to change CPU architecture to increasing computer performance. They provide processors with two or more computing cores instead of one. The different core can take different task at the same time. There is almost no desktop computer in the market containing a single computing core. Most of them have four-core CPU. Most of laptop computers have two cores CPU. However, most programming languages such as GAUSS, Ox, MATLAB and SAS, which are widely used in economics, finance and engineering, use by default only one core for computation. In contrast, many models based on numerical techniques are notoriously computation intensive and require weeks of CPU time on powerful computers. So, many

[1]An introductory explanation of GPU is available for example in Wikipedia.

high level programming languages provide interface to use multi-core or multi-node for intensive computation.

There is a third party package parallel for GAUSS or GAUSS GE. The most recent GAUSS 11 provides multi-thread functions for matrix operation such as LU and Cholesky factorization. Ox and MATLAB provide not only for multi-core (openMP)programming but also multi-node(MPI) parallel computing. JACKET from AccelerEyes and GPUmat from GP-YOU are third party packages for GPU parallel computing in MATLAB.

Doornik et.al(2004)[11] gives a elaborate introduction on Ox. Parallel Computing Toolbox 5.0 [32] in MATLAB provides interfaces for GPUs, multicore computers and computer clusters. Kepner(2009)[24] offer a textbook on MATLAB parallel computing. However, in the high-level programming, the parallel computing is not as smooth as serial programming. They have to shift between the serial environment and parallel environment. For example, in JACKET and GPUmat, the matrix in MATLAB has to change into GPU data storage format for GPU computation.

When the GPU computation finish, the result has to be transferred to MATLAB storage format. If the data is too big for GPU memory, there would be memory leak and result in MATLAB shut down. Another disadvantage for high-level parallel is that the parallel computing only can be carried out outside of the MATLAB function. For instance, function mle is a maximum likelihood estimation function.

If we want this function to estimate a model by parallel computing, we have to rewrite a function in MATLAB. The low-level programming languages such as C/C++ and Fortran have more flexible control on memory and programming style. They also provide efficient speed when the program is compiled into machine language.

All the high level programming languages have an interface that enable them to interact with low level efficient language(API). For example, MATLAB has such an API and communicates with C/C++, Fortran and Java. The capability of combination with low level language enable high level programming language to exploit more potential power in a computer.

This chapter starts with an introduction on the combination of MATLAB and

C/C++. The combination dramatically accelerate the speed on some intensive computation programs.

There are three levels on speeding up. The first one is simple substitution for the MATLAB script code. This kind of combination is fit for the computation which has big size loop and can not be parallelized. For example, recursive iteration filter computation, which are commonly used in Bayesian estimation and very slow in high level programming language.

The second combination is multicore parallelizing. Most computers have multicore cpu but high level programming language use only one of them for computation by default. Although there are several paralleling methods in MATLAB script code such as Kepner(2009), the speed gains are not significant. In contrast, the combination method can accelerate about 5 times efficiency on general matrix computation (Goto and Van De Geign 2008) .

The most efficient method is massive parallel programming using Compute Unified Device Architecture(CUDA) by NVidia [38]. In the combination with MATLAB, CUDA as well as companied libraries work as a general C/C++ library but achieve hundreds time faster compare to the same functional MATLAB script codes.

The organization of the chapter is as follow. Section 3.2 provides a brief introduction to most popular programming languages in Econometrics and Bayesian estimation. We will explain why we choose MATLAB as a platform to combine low level programming language. In section 3.3, we will explain the easiest way to call C++ function from MATLAB code. we also illustrate by an example how to build an efficient Mex function by different ways. In section 3.4, we provide a canonical example to show efficiency obtained from a three level combination. Section 3.5 is a conclusion on further research direction on Bayesian inference using massive parallel computing.

3.2 Programming Languages for Econometrics

Econometricians have been using a variety of programming languages for their research. There are three groups of computing languages for data analysis. The first group is the

high-level packaged programming languages such as SAS, SPSS, EVIEWS and STATA. These languages provide a set of statistical toolboxes for most commonly used data analysis functions. For example, in EVIEWS, built-in models can be chosen to analysis the data. It is not necessary to write any code for the models. This built-in toolbox carry out the analysis in an efficient way since the functions are compiled into machine language. They are commonly used by practitioners in industry and in college for business, teaching and research.

However, the price for this convenience is that the toolboxes are fixed and can not be extended to the latest econometric models and analysis methods. We have to wait for the release of a new version that include new techniques. Although these high level programming packages have a little programming capability, their programming grammar differs from the traditional and friendly style and make it hard to learn.

The second group includes matrix-oriented language like MATLAB, GAUSS, S-PLUS(R) and OX. This group presents a more flexible programming environment for developing new models or new analysis methods. Each of them also has special application packages. For example, MATLAB has financial toolbox for time series data. MATLAB also has econometric toolbox, optimization toolbox, fixed-income toolbox and so on. GAUSS has Maxlik module and CML module for optimization computation. GAUSS also has a financial module for time series data. R has a giant number of packages. There would be a package for any special application in econometrics or statistics. OX has a lot of free third part packages including ARFIMA, Financial, Maximize, MultiGARCH, time series Modeling and so on.

The third group consists of low level programming languages. They are JAVA, Fortran and C/C++ which can access to hardware control and have much more computational power than high level programming language. The low level programming languages do not have friendly interface like high level language. They need more input and output for a computation task. Most of the time, we have to write a lot of codes for a simple operation in low-level language. Fortunately there are many libraries for mathematical operation in low level programming languages. For example, Fortran has IMSL math library. C/C++ has MKL, ATLAS and GotoBLAS that not only make the

programming easy but also make computing optimal.

There are many references on the comparisons of programming languages. The comparisons are usually based on runtime efficiency, readability, ease of learning and reliability. Rust(1993) compared MATLAB and Gauss. Cribari-Neto (1999) compared C with other high-level programming languages. Belsley (1999) introduced Mathematica as a computing tool in economics and econometrics. Prechelt(2006) compared C, C++, Java, Perl, Pychon, Rexx and Tcl. Cribari-Neto(2003) compared OX with other high-level programming languages. However, these comparisons are based on old software versions. Some programming languages are changed dramatically. For example, MATLAB uses C instead of Fortran to write its function and use Just-In-Time techniques to speed its script codes. OX and GAUSS adapt to parallel computing in their new version. Hence, those comparisons that are presented above cannot serve as a criteria to choose computing tool for research.

Because the data size becomes large and the model becomes complex without corresponding increase in computing clock, it is essential for all high-level programming languages to use parallel techniques for the computing power demand. So, one more criteria should be added to choose an appropriate programming language.

Among all the high-level programming, MATLAB provides the most convenient way to interact with low level programming languages to speed up computation. A summary of the advantages that MATLAB has compared to other high-level languages are the following.

- The function profile in MATLAB can help programmers to track program's execution time. This function make debugging code easy. When a part of the codes are checked as big time consumer, we can substitute them for compiled low-level codes or parallelize them.

- The combination of MATLAB with a low-level programming language such Fortran and C/C++ can hide parallelization from the end front users. For example, when A and B are big size matrices, C=A*B would be very slow. We can write an parallelizing Mex function named MatrixMultiply and use it as

$C = MatrixMultiply(A, B)$. So, the combination make the programm to run with an interface that is identical to the interface of an equivalent serial version. The new function is faster, more concise, readable and friendly.

- In MATLAB, we can change the BLUS and Lapack library that are used for matrix operation to speed up computation by automatically paralleling codes. Automatically Tuned Linear Algebra Softwar (ATLAS) [44] and GotoBLAS are algebra operation low-level programming language libraries. These libraries are complied and optimized according to computer hardware. We can use them instead of Math Kernel Library (MKL) which is from Intel. Since GotoBLAS [17] can parallelizes the matrix operation automatically, the programm can run faster by a 2–5 factors depend on computing hardware.

- In the latest released version, MATLAB can use GPU to process massively parallel compute time-expensive problems. In the parallelizing trend, almost all the high level programming languages use multi-thread for intensive computation. But they just use multi-thread in CPUs. Only MATLAB has embedded GPU parallelizing into the release version.

- There is a handy large community that uses MATLAB and exchanges programming experience. A program beginner can easily find helper in the community forum. For example, Iversen(2010)[2] posts a program for publication quality graphic output. Buehren(2010)[3] posts a package for multiple cores programming. That would be helpful for programming beginners.

- MATLAB provides a friendly help file system. The built-in help file includes not only function usage but also the background theory. Since compiled mex file can be used with text help file, we can write the same name file with m extension to explain the mex function. When the mex function is looked for help as other

[2]www.mathworks.com/matlabcentral/fileexchange/7943-freezecolors-unfreezecolors

[3]www.mathworks.com/matlabcentral/fileexchange/13775-multicore-parallel-processing-on-multiple-cores

build-in function is searched in command window. The text in m file shows up to explain it.

So, we can regard MATLAB as an intermediate which combine friendly and concise programming interface with efficient computation routines.

3.3 Combining MATLAB with C/C++

3.3.1 Matrix multiplication example

We explain how the multiplication of large dimensional matrices can be speeded up if we combine MATLAB and C/C++ using GPU. There are many ways to combine MATLAB with C/C++ [33]. One is calling C/C++ function in MATLAB. another is to call a MATLAB function from C/C++. Since there are a lot of matrix operation libraries for the C/C++ user to do matrix multiplication, there is little benefit to use matrix library in C/C++ programming. This chapter focuses on calling C/C++ function in MATLAB which is to speed up the function. First, we use a simple example to illustrate how MATLAB works in processing matrix multiplication of 5,000×5,000 matrices of random numbers drawn from the standard normal distribution.

```
1 X=randn(5000,5000); % Generate a 5000x5000 matrix .
2 Y=randn(5000,5000); % Generate another 5000x5000 matrix .
3 Z=X*Y;              %  X multiply Y.
4                     % It takes about 24 seconds
```

Listing 3.1: a simple example in MATLAB

The codes generate two matrices and multiply them to get the third matrix. Behind the computer screen, the computer does a lot of work for this several lines of program. A simply summary of operation is as follows:

- 1. Set up a memory space for X whose size is 5000×5000×double type memory size;
- 2. Generate random numbers from Gaussian distribution using MATLAB built-in function randn and put them into the memory space;

- 3. Same operation for matrix Y;
- 4. Set up a memory space for Z whose size is 5000×5000×double type memory size;
- 5. Multiply X with Y using MATLAB built-in function $*$ and put the result into the Z memory space.

In Listing 3.1, randn is a MATLAB built-in function to generate random samples from the standard normal distribution. Star $*$ also is a MATLAB built-in function for matrix multiplication. Although MATLAB routine that optimizes matrix multiplication, it uses only one CPU core for computation. The function $\star$ takes a long time to produce matrix Z. In the computer with CPU core i7, 6G memory using MATLAB 2009b, it takes about 23 seconds. If we want to speed up this operation, we can use a multi-thread algorithm for matrix operation. We can write a C function named MatrixMultipy which compute parallel. Then Z can be obtained by Z=MatrixMultipy(X,Y) which will be faster than function $*$. The simplified C/C++ function with comment is as following[4].

```
1  // mexFunction is fixed and to specify the input
2  // and output of a mex function.
3  void mexFunction (int nlhs, mxArray *plhs[],
4                    int nrhs, const mxArray *prhs[])
5  {
6      // locate the address of input Matrix X;
7      double *X = mxGetPr(prhs[0]);
8
9      //check the size of the input Matrix X;
10     int X_Mrows = (int)  mxGetM(prhs[0]);
11     int X_Ncols = (int) mxGetN(prhs[0]);
12
13     //locate the address of input matrix Y;
14     double *B = mxGetPr(prhs[1]);
15     //get the size of input matrix Y;
16     int Y_Mrows = (int) mxGetM(prhs[1]);
```

[4]the complete code is attached in appendix

```
17      int Y_Ncols = (int) mxGetN(prhs[1]);
18
19      //Check if X and Y match;
20      if ( Y_Mrows==X_Ncols)
21          mexErrMsgTxt("X's columns must match Y's rows");
22
23      // Generate a Matrix for output;
24      plhs[0] = mxCreateDoubleMatrix(N,1,mxREAL);
25
26      //Name the address of output;
27      double *C=mxGetPr(plhs[0]);
28      // computing for C;
29      C/C++_routine_for_matrix_multiply(A,B,C);
30      return;
31  }
```

Listing 3.2: a simplified C/C++ program to illustrate the combination of MATLAB and C/C++

In Listing 3.2, a simplified C/C++ programm is used to combine MATLAB with C/C++. The keyword mexFunction is used to specify the input and output of a mex function which is compiled from C/C++ codes. Other keywords such as mxGetpr, mxCreateDoubleMatrix and mxGetM are reserved for the combination of MATLAB and C/C++. Figure 3.1 provides a general visual illustration that how MATLAB and C/C++ work together.

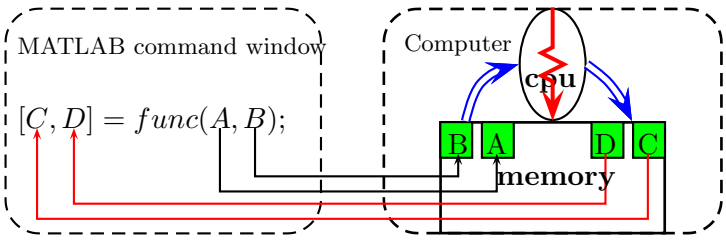

Figure 3.1: illustration of combination of MATLAB and C/C++

In Figure 3.1, there are two parts. The left part is command window in MATLAB. It is a front interface for MATLAB programmer to implement a function. The right

part is the backyard of the front interface. It shows how computer uses C/C++ or other low-level programming languages to work out for the function in front interface. In Figure 3.1 A and B are input variables C and D are output variables of the function func which is compiled from the C/C++ program. A and B are pre-built matrices or vectors by MATLAB. C and D are created output by function func. When we run the function, it first locates the address of A and B in computer memory (green part in right part). Then, the properties of A and B are checked. According to the A and B's properties and programmer's requirements, the function generates the memory space C and D. The last step is to implement the computing process using low-level computing routine and put the result into the memory space C and D.

In our real C/C++ program, we use routine DGEMM in GotoBLAS library to replace C/C++_routine_for_matrix_multiply in Listing 3.2. The compiled mex function named MatrixMultiply works as a MATLAB built-in function but with a faster speed depending on hardware in computer (refer to Listing 3.3). In the computer with core i7 that has 4 cores and can compute with 8 threads, the function MatrixMultiply takes about 7 seconds, which is much faster than the 23 seconds in Listing 1.

```
1 X=randn(5000,5000);
2 Y=randn(5000,5000);
3 Z=MatrixMultiply(X,Y); % It takes about 7 seconds.
```

Listing 3.3: the new function hides parallel programming and run using multi-thread

From Listing 3.3, we can see that the combination of MATLAB with C/C++ hides complicated parallel programming work and provides a concise, friendly interface for the end user.

Since there are a lot of matrix or vector operations such as inverse, LU factorize and QR factorize, do we have to write a mex function which run parallel for each of them? The answer is no if we have GotoBLAS2 library. It is well known that MATLAB use MKL library for algebra operation. MKL library is an optimized BLAS and LAPACK library for algebra operation from Inter Company. However, it is only for

series computation. GotoBLAS2 is developed by Goto in Texas Advanced Computing Center of the University of Texas at Austin. The library is compiled optimized according to hardware. It uses cache in CPU to speed up computing algebra problems. Even use one thread (without parallel), GotoBLAS2 is about 30% faster than MKL[44]. It is simple to use GotoBLAS2 for parallel computation. After library GotoBLAS2 is compiled, copy it to the folder where MKL library is in and change the environment variable to use GotoBLAS2 in the file blas. Then, when the program in Listing 3.1 run, it take the same time as program in Listing 3.3 since it run multi-thread parallel.

From the above description we see that it is not hard to combine MATLAB and C/C++. It is worthwhile to write a C/C++ function for a time consuming problem if it can save more run time than developing time. In addition, there are a lot of free or commercial libraries that make C/C++ programming pretty easy and save a lot of developing time. For example, BLAS and LAPACK are for algebra operation, Opt++ is for optimization, libquant is for time series. However, most of them are developed in Unix system and it is not easy to be compiled in the Windows system. For example, ATLAS and GotoBLAS2 are a little hard to be compiled into 64-bit libraries in Windows system. Our experience is to use Cygwin+Wingw_W64(refer to web reference).

3.3.2 Matrix multiplication using GPU

Since C/C++ is a low-level programming language which can manage and control hardware, we can use not only a faster C/C++ library but also other hardware for computation. A kind of graphic card (usually named Graphics Processing Unit GPU) from Nvidia offers a general computation capacity. This kind of card works like massive core CPU and provides up to ten thousand thread for parallel computing. The GPU's main work processes are to transfer the data into GPU memory, then parallel compute using up to ten thousand threads and return the computing results back to computer memory. Another character in GPU computation is that there are shared memory in GPU. The shared memory works as cache in CPU and enable the difference thread block to share data at a very fast rate. An example introduction by Sanders and Kandrot(2010)[40] is very helpful for a beginner. Kirk and Hwu (2010)[25] provides an

introduction on hardware with GPU programming. Those two feathers provide GPU great computing power for an intensive time consuming problem. The extended version of Figure 3.1 is Figure 3.2 which include the GPU part.

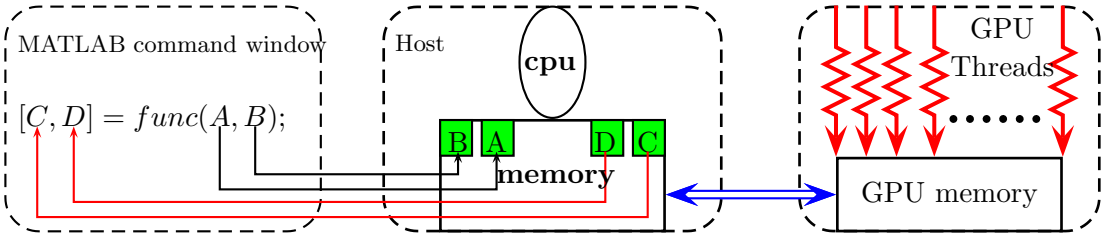

Figure 3.2: An illustration of MATLAB,C/C++ and GPU working together

The new function from combination of MATLAB, C/C++ and GPU named Matri-Multiply_GPU run very fast. In our test program, it takes 0.75 second.

```
1 X=randn(5000,5000); % Generate a  5000x5000 matrix.
2 Y=randn(5000,5000); % Generate a  5000x5000 matrix.
3 Z=MatrixMultiply_GPU(X,Y);  % It takes about 0.75 seconds
```

Listing 3.4: a simple example for GPU computation in MATLAB

The combination of MATLAB and C++ provides an efficient, concise and friendly implementation of time-consuming computing. The compiled mex function works as MATLAB built-in function and hide complex C/C++ codes. In order to compare their efficiency, Table 3.1 provides runtime in second.

X=5000 × 5000 Y=5000 × 5000	**runtime(sec.)**	**speed rate**
$X * Y$	23	
MatrixMulitply(X,Y)	7	×3.3
MatrixMulitply_GPU(X,Y)	0.75	×30.7

Table 3.1: computing time in second for matrix multiply functions

3.4 A Canonical Example: Kernel Density Estimation

Matrix operation is one kind of the most intensive computations, there are many existing computation libraries for speeding them up. These computation libraries include BLAS, LAPACK, ATLAS, GotoBLAS, IMSL and MKL. These libraries not only speed up computation with dramatically rate but also make programming easier. In GPU computation, there is a optimized library CUBLAS also makes the matrix operation fast and easy. In this part, we introduce another example which needs a lot of computation and we need to write codes by ourselves. This example involves density estimation. This example illustrates how the GPU parallel tool speeds the intensive computation problem.

3.4.1 Introduction to the density estimation

The nonparametric kernel density estimation is an important tool in statistical data analysis. It is widely used in various inference procedures in machine learning, data mining and pattern recognition. In Bayesian inference, it is used to summarize the Bayesian posteriors. The key point in the kernel density estimation is to select the optimal bandwidth of the kernel.

The basic theory on the optimal bandwidth was summarized in Wand and Jones (1995)[43] and Silverman (1986) [41]. Suppose $\mathcal{S} = \{X_i, i = 1, 2, ..., n\}$ is a set of sample data from the density $f(x)$ and $\mathcal{T} = \{x_j, i = 1, 2, ..., m\}$ are the points at which we want to estimate the density. The kernel density estimator at point x_j is an average of kernel functions on n observations $X_1, X_2, ..., X_n$.

$$\hat{f}(x_j; h) = (nh)^{-1} \sum_{i=1}^{n} K((x_j - X_i)/h), \quad x_j \in \mathcal{T}, \quad X_i \in \mathcal{S} \tag{3.1}$$

where h is the bandwidth and the kernel $K(x)$ is a function satisfying $\int K(x)dx = 1$ and $K(x) \geq 0$[5]. The closer the density estimator to the true density, the better is the estimator. We define the distance between the true density and the estimator as

[5]some higher order kernels can not satisfy $K(x) \geq 0$.[22]

Average Sum of Squared Error(ASSE) in equation (3.2):

$$\mathbf{ASSE}\{\hat{f}(x;h), f(x)\} \equiv \frac{1}{m}\sum_{j=1}^{m}\{\hat{f}(x_i;h) - f(x_i)\}^2, \quad x_j \in \mathcal{T} \tag{3.2}$$

If $\mathcal{T}$ is all the support of $f(x)$ or the whole real line $\mathcal{R}$ instead of the finite discrete set, the distance between the true density and the estimator is called Integrated Squared Error(ISE) which is given by equation (3.3).

$$\mathbf{ISE}\{\hat{f}(x;h), f(x)\} = \int_{\mathcal{R}}\{\hat{f}(x;h) - f(x)\}^2 dx \tag{3.3}$$

However, ISE depends on the sample data $\mathcal{S}$ and is not adequate for more than one sample set. If there are a lot of data sets $\mathcal{S}_1$,$\mathcal{S}_2$... from the density $f(x)$, an average of the distance should be used to check the performance of the density estimator. Therefore, the generalized distance between two functions are provided by Mean Integrated Squared Error(MISE) in equation (3.4). It is an appropriate criterion to analyze the closeness of the density estimator and the true density.

$$\mathbf{MISE}\{\hat{f}(x;h), f(x)\} = E[\int\{\hat{f}(x;h) - f(x)\}^2 dx] \tag{3.4}$$

From equation (3.1), it is known that the performance of the kernel density estimator $\hat{f}(x;h)$ depends on the choice of kernel function $K(x)$ and bandwidth h. In order to examine the effect of both elements on the MISE, we can substitute equation (3.1) into the equation (3.4) and get a function of $K(x)$ and bandwidth h in equation (3.5).

$$\begin{aligned}\mathbf{MISE}\{\hat{f}(x;h), f(x)\} = (nh)^{-1}\int K^2(x)dx \\ +(1-n^{-1})\int (K_h * f)^2(x)dx \\ -2\int (K_h * f)(x)dx + \int f^2(x)dx\end{aligned} \tag{3.5}$$

where $K_h(x)$ is the abbreviated expression of $h^{-1}K(x/h)$. Equation (3.5) is a function of bandwidth h. We wish to minimize MISE with respect to h but it seems hard to minimize the function and find the optimal bandwidth h. Fortunately, if we make some assumptions on the kernel and make it regular enough, we may use the Central Limit Theory. Then, an approximation of MISE is available and it has a simpler expression

and is easier to handle. This approximation is defined as Asymptotic Mean Integrated Squared Error (AMISE) as in equation (3.6):

$$\begin{aligned} \mathbf{MISE}\{\hat{f}(x;h), f(x)\} &\approx \mathbf{AMISE}\{\hat{f}(x;h), f(x)\} \\ &= (nh)^{-1}R(K) + \frac{1}{4}h^4\mu_2(K)^2R(f'') \qquad (3.6) \\ where\ \ R(K) = \int K(x)^2dx, &\qquad \mu_2(K) = \int x^2K(x)dx \end{aligned}$$

In equation (3.6), the first term is the integrated variance and the second term is the integrated squared bias. Since AMISE is both an increasing monotonic function of h and a decrease monotonic function of h, this lead to the famous bias-variance tradeoff in the density estimation. The optimal bandwidth which minimizes the AMISE is easily derived from equation (3.6) which is genearally denoted as $h^*_{\mathbf{AMISE}}$ and has a closed form as in equation (3.7):

$$h^*_{\mathbf{AMISE}} = \left[\frac{R(K)}{(\int x^2K)^2R(f'')n}\right]^{1/5} \qquad (3.7)$$

Although this expression is simple, the optimal bandwidth cannot be calculated directly. The reason is that $R(f'')$ is a function of the second order derivative of the true density function $f(x)$ which is unknown.

Since the most data sets in practice are Gaussian or asymptotically Gaussian(by the Central Limit Theory), it is natural to assume the function $f(x)$ is Gaussian distribution with standard deviation σ. If the kernel is also Gaussian, this assumption makes equation (3.7) is simple as in equation (3.8).[41][2]

$$h = \left(\frac{4}{3n}\right)^{1/5}\sigma \qquad (3.8)$$

Here σ is the standard deviation of the density $f(x)$. In practice, in order to get the optimal bandwidth, the sample standard deviation $\hat{\sigma}$ is used in the equation (3.8). This choice can accommodate long tailed distributions and outerliers. It is a robust estimator of σ.

However, the normal density is one of the smoothest distributions. The value h would be too large for non-normal data and it will tend to induce over-smoothing.

There are variations that have been proposed to find the optimal h for non-Gaussian data[21]. One of the most popular methods is solve-the-equation plug-in approach. The main steps are as following.

- define $\psi_r = \int f^{(r)}(x) f(x) dx$.
- the estimator of ψ_r, $\hat{\psi}_r = \frac{1}{n}\sum_{i=1}^{n} \hat{f}^{(r)}(x_i; h_r)$. where h_r is the pilot bandwidth to estimate the rth derivative of the density $f^{(r)}$.
- The optimal bandwidth h_r which is used to estimate ψ_r by asymptotic mean square error criteria is

$$h_r^{AMSE} = \left[\frac{-2k^{(r)}(0)}{\mu_2(K)\psi_{r+2} n} \right]^{1/(r+3)} \tag{3.9}$$

- So, $R(f''(x)) = \int [f''(x)]^2 dx = \psi_4$.(using integration by parts)
- So, the estimator of ψ_4 is

$$\hat{\psi}_4 = \frac{1}{n}\sum_{i=1}^{n} \hat{f}^{(4)}(x_i; h_4) = \frac{1}{n^2 p^5} \sum_{i=1}^{n}\sum_{j=1}^{n} K^{(4)}\left(\frac{x_i - x_j}{h_4}\right)$$

  . Substitute it into equation (3.7), and obtain equation (3.10).

$$h = \left[\frac{R(K)}{(\int x^2 K)^2 \hat{\psi}_4 n} \right]^{1/5} \tag{3.10}$$

- the optimal pilot bandwidth h_4 to estimate $\psi_4(R(f''))$ is abtained from equation (3.9) directly.

$$h_4 = \left[\frac{-2K^{(4)}(0)}{n\mu_2(K)\hat{\psi}_6} \right]^{1/7} \tag{3.11}$$

- The relationship between bandwidth h and pilot bandwidth p is derived from the equation (3.10) and (3.11):

$$p(h) = \left[\frac{-2K^{(4)}(0)\mu_2(K)\hat{\psi}_4}{R(K)\hat{\psi}_6} \right]^{1/7} h^{5/7} \tag{3.12}$$

- We finally get the fixed-point equation of h.

$$h = \left[\frac{R(K)}{(\int x^2 K(x)dx)^2 R(\hat{f''}(x; p(h)))n} \right]^{1/5} \tag{3.13}$$

The summary of the path of computing the optimal h is in Figure 3.3. In order to solve the h function, the parameter include the $\hat{\psi}_4$ and $\hat{\psi}_6$ has to be computed first. However, both of them require the optimized bandwidths (p_4,p_6 in the figure) for estimation. Further more, the optimized pilot bandwidths need higher order density derivatives for the calculation. This process will continue iteratively. The practical strategy for this problem is to stop it at some stage where the density estimation converge with a good rate. For example, use a non-optimal approximations of $\hat{\psi}_6$ and $\hat{\psi}_8$ in Figure 3.3.

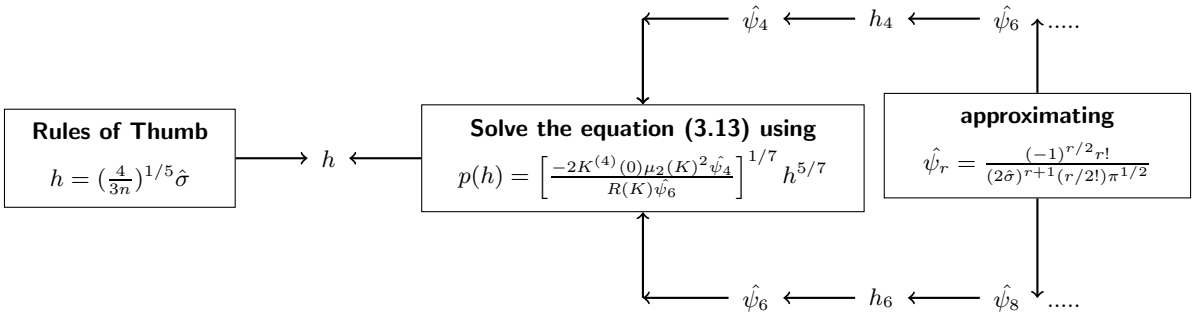

Figure 3.3: the path of computing h. h is the optimal bandwidth of density estimation. ψ_r is the mean of the rth derivative of density. p_r is the pilot optimal bandwidth of estimation of rth density derivative.

3.4.2 Computation of the Kernel Density Estimation

The focus of this section is not on finding the new or more optimal bandwidth (see Botev et al (2010) [1]). Our focus is to obtain speedy computation of a data driven optimal bandwidth.

As we have discussed in section 4.1. There are two methods of the Kernel Density Estimation. The simpler one uses simple bandwidth as in equation (3.8). Since the bandwidth is based on the assumption that the data is from Gaussian density, this method works well for the data which is Gaussian or close to Gaussian. However, the simple bandwidth is not good for non-Gaussian data. In this case, we have to to compute the optimal bandwidth from the data. The more accurate method drives the optimal bandwidth from data as in the right part of Figure 3.3, then calculate the

kernel density.

In order to compare the fitness of the two methods, we write two MATLAB functions. Function kden uses the simple bandwidth and kden_optimizing_h uses the data driven optimal bandwidth. Both functions take a vector as input. There are two outputs for them. The first one is sorted input. That means the function computes the density for every point in input. The second output is the correspondent density for the first output. In our simulation, the random samples are fifteen Gaussian mixture densities from Marron and Wand(1992) [31]. In order to illustrate the algorithm, we put function kden in Listing 3.5.

```
1  %kernel density estimation
2  function [sorted_X,estimated_density]=kden(X)
3  [Nrows,Mcols]=size(X);
4  if min(Nrows,Mcols)>1
5      error('input must be a N x 1 or 1 x N vector');
6  end
7  N=max(Nrows,Mcols);
8  h=1.06*std(X)/N^0.2; %h = (4/3n)^{1/5} σ̂
9  Ed=zeros(N,1); %preallocation
10 for j=1:N
11     Ed(i)=mean(normpdf((X-X(j))/h))/h;%f̂(x;h) = (nh)^{-1} Σ_{i=1}^{n} K((x - X_i)/h)
12 end
13 if (Nrows>Mcols)
14     temp=cat(2,X,Ed);
15 else
16     temp=cat(2,X',Ed);
17 end
18 ret=sortrows(temp);
19 sorted_X=ret(:,1);
20 estimated_density=ret(:,2);
```

Listing 3.5: MATLAB function for kernel density estimation with simple Gaussian bandwidth

In Listing 3.5, the input vector X is checked first. Then it calculates the simple bandwidth. After that, the kernel density is evaluated at each point using the simple bandwidth. Finally, it arrange the output vectors. The program in Listing 3.6 is a simulation example of function kden. In the program, 10,000 variates are generated from function Gaussian_Mixture_Sampler[6]. Then we use kden to estimate the kernel density. Finally, the true density and estimator are compared in a figure. This process is repeated for different densities. There are 15 graphs for 15 different densities. They are put together in Figure 3.4.

```
% Generate the random samples from the Gaussian Mixture Density
    N=10000; %sample size
    Y=linspace(-5,5,N);
     % 10 is the number to choose density. There are 15 different
     % densities as in figure 4
    [actual_density,X]=Gaussian_Mixture_Sampler(10,Y,N);
%Kernel density estimation using function kden.
    [sorted_X,D]=kden(X);
    %[sorted_X,D]=kden_c(X);
    %[sorted_X,D]=kden_c_para(X);
    %[sorted_X,D]=kden_c_gpu(X);
    %[sorted_X,D]=kden_opti_h(X);
    %[sorted_X,D]=kden_c_opti_h(X);
    %[sorted_X,D]=kden_c_para_opti_h(X);
    %[sorted_X,D]=kden_c_gpu_opti_h(X);
    plot(Y,actual_density);
    hold on;
    plot(sorted_X,D)
```

Listing 3.6: MATLAB program for simulation example

From Figure 3.4, if the true density is close to Gaussian, the estimated kernel density is close to the true density. When the density is far away from Gaussian, there is oversmoothing in estimation such as Smooth Comb, Claw and Discrete Comb. In one

[6]The function code is from Vikas.C.Raykar (2005)

density, the estimation function works well in the area where is smoothing and more like Gaussian. The fitness is very bad in the area that is no smoothing such as in Smooth Comb and Discrete Comb.

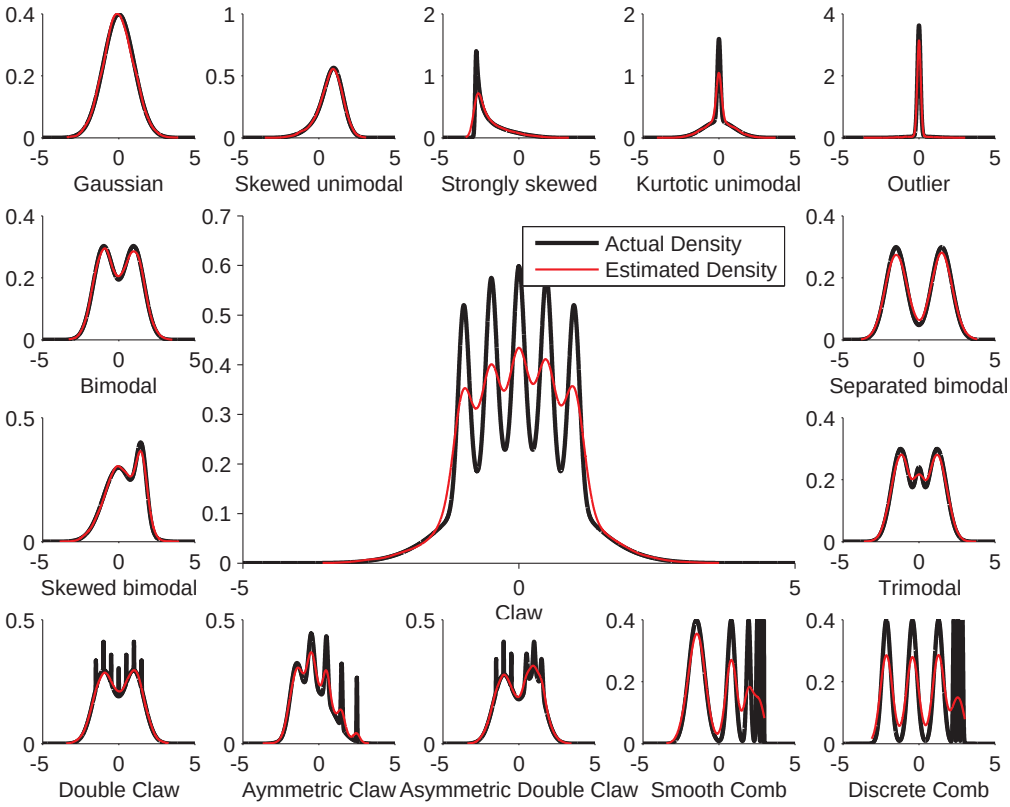

Figure 3.4: Kernel Density Estimation using simple Gaussian bandwidth for Gaussian mixture densities

Besides the presence of over-smoothing, function kden is very slow. When the sample size is $10,000$, the time used to estimate the density Discrete Comb is about 6 seconds.[7] when the sample size is $30,000$, the time is about 50.53 seconds. When the sample size is $50,000$, the cost is 137.24 seconds. The time cost increase by $O(N^2)$. That is what is indicated in the second column in Table 3.2. So, the kernel density estimating function in Listing 3.5 is not good enough for no-Gaussian density. It is too slow for big sample data set. In Figure 3.5, we see that kernel density estimation function kden works well for most functions except a few odd densities. So, our first step

[7]The computing hardware is core^{TM} i7-920 with 2.66GHz and 6G RAM. The computing software is MATLAB 2010A under x64 windows 7.

is to speed it up. kden is a MATLAB script function. It is very slow because it has loop which can not be vectorized. Using the combination method of MATLAB and C/C++ we have introduced in section 3, we write three new functions for speed up. Function kden_c(X) is a compiled mex function from C/C++ program which is a series version . Function kden_c_para(X) is similar to kden_c(X) but using parallel computation with CPU multi-thread. Function kden_c_gpu(X)[8] also is a C/C++ program which uses GPU multi-thread for kernel computation. All three new functions have the same inputs and outputs as function kden. Table 3.2 presents their performance for a series of sample sizes. The simple C/C++ substitution does not make significant improvement. Function kden_c only run faster at about 1.8 factor when the sample size is big. The C/C++ parallel and GPU parallel code run much faster than MATLAB script function. The most impressive is that GPU can make a 423 time faster.

DataSize	**kden(X)**	**kden_c(X)**	**kden_c_para(X)**	**kden_c_gpu(X)**
1000	0.14	0.03(×4.67)	0.01(×14)	0.0034(×41.18)
5000	1.72	0.80(×2.15)	0.21(×8.19)	0.011(×156.67)
10000	6.15	3.22(×1.90)	0.73(×8.42)	0.020(×307.5)
30000	50.53	28.91(×1.74)	6.08(×8.31)	0.16(×315.81)
50000	137.24	80.22(×1.71)	15.67(×8.76)	0.35(×392.11)
80000	355.96	205.44(×1.73)	38.26(×9.30)	0.84(×423.76)

Table 3.2: computing time in seconds for density estimation using simple bandwidth. kden is a MATLAB script function ;kden_c is a C++ compiled function without parallel; kden_c_para is a C++ compiled function with parallel; kden_c_gpu is a C++ compiled function uing GPU for parallel computation.

[8]The GPU cards is GTX 470 from NVidia which has 448 processor cores, 1280M RAM and can run up to $65,535$ threads, the software for the card is CUDA 3.2 RC under windows 7 x64

In order to estimate the density for non-Gaussian data, the driven optimal bandwidth should be computed first. So, the kernel estimation with driven optimal bandwidth h is a little harder than that with simple Gaussian bandwidth. In kden_opti_h , MATLAB function lsqnonlin is used to solve fixed point equation 3.13.

In our C/C++ programs, we use Lev-Mar [29] library to solve it. Function kden_opti_h is much slower than kden since it computes density derivative several times as well as solve a optimization equation. The second column in Table 3.3 is the run time of function kden_opti_h for different sample size. In order to speed it up, we also write three C/C++ programs using CPU series, CPU parallel and GPU parallel. In order to recognize them easily, they are named as kden_c_opti_h, kden_c_para_opti_h and kden_c_gpu_opti_h respectively. Each has input X and two outputs same as in function kden.

Their estimation results are in Figure 3.5. Comparing Figure 3.4 and Figure 3.5, we can see that functions which drive optimal bandwidth then compute density work better than functions which use simple Gaussian bandwidth if data is non-Gaussian. For example, claw density is a multimodal. When we use simple Gaussian bandwidth, the estimator is serious overshooting. There is a big difference between true density and estimator in bump area. The same phenomena appear in other densities which is not close to Gaussian density(refer to Figure 3.4). When we use data driven optimal bandwidth, the estimators are very close to the true densities (in Figure 3.5).

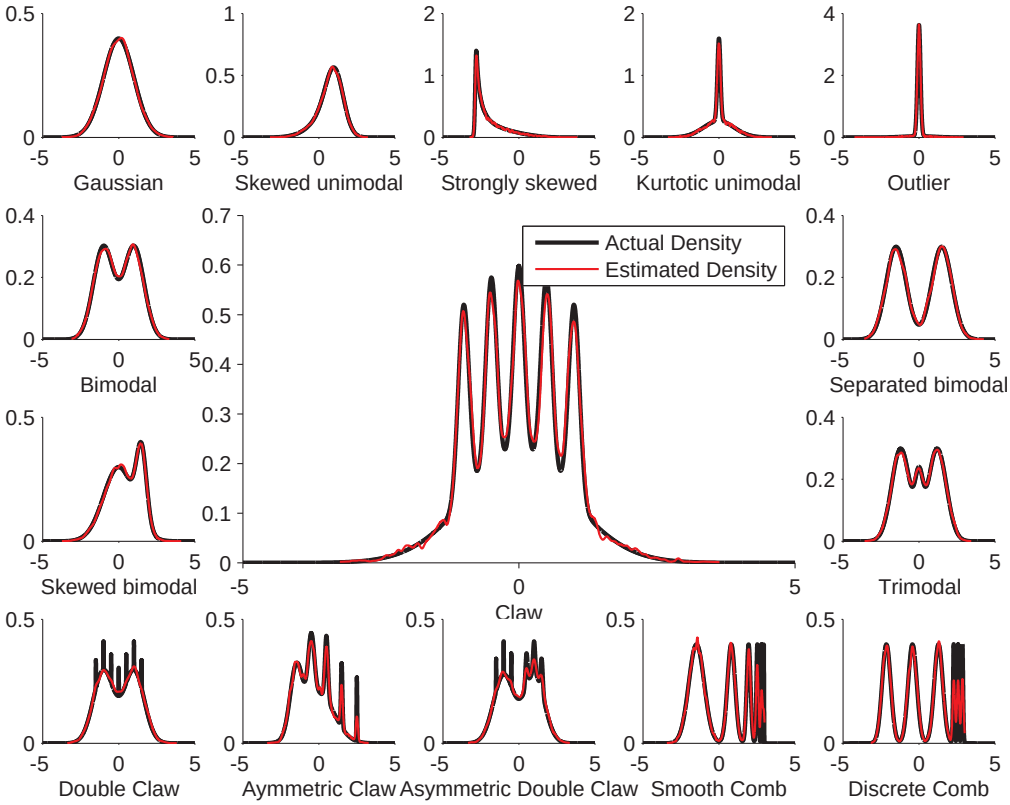

Figure 3.5: Kernel Density Estimation using data-driven asymptotic optimal bandwidth for Gaussian mixture densities

Table 3.3 compares run times of those three new functions which use kernel density estimation by driving optimal bandwidth. When samples size becomes big. MATLAB script function uses too much time for the estimation. The simple C/C++ code substitution also doesn't offer much help. The parallel functions in CPU and GPU dramatically speed up the estimation. GPU version make it at 455 speed rate.

DataSize	kden _opti_h(X)	kden_c _opti_h(X)	kden_c_para _opti_h(X)	kden_c_gpu _opti_h(X)
1000	4.16	0.86(×4.83)	0.23(×18.09)	0.053(×78.49)
5000	48.82	21.53(×2.27)	4.86(×10.05)	0.20(×244.10)
10000	166.94	86.93(×1.92)	17.80(×9.38)	0.39(×428.05)
30000	1292.38	798.02(×1.62)	162.28(×7.96)	3.35(×385.79)
50000	3378.80	2268.71(×1.49)	470.20(×7.19)	7.41(×455.98)

Table 3.3: computing time in second for density estimation by driving the optimal bandwidth. kden_opti_h is a MATLAB script function ;kden_c_opti_h is a C++ compiled function without parallel; kden_c_para_opti_h is a C++ compiled function with parallel; kden_c_gpu_opti_h is a C++ compiled function using GPU for parallel computation.

3.5 Concluding Remarks and Possible Application

Bayesian inference and MCMC are time consuming estimation methods. Although computer's capability has increased during last decade, MCMC algorithms still need much more time than other estimation methods such as maximum likelihood estimation. Parallel computing, especially massive parallel in GPU will play an important role to speed MCMC computation in the near future since graphics cards is affordable hardware to most computer users. Some GPU computation libraries such as CULA, CUDPP and Thrust also make the GPU parallel programming easy and practicable for researchers who are not majored in computer science. This chapter is an introduction to CPU and GPU parallel computation. There are many ways of implementing the parallel computation. The combination of MATLAB and C/C++ provides a concise and friend usage for function end users. Since MATLAB contain Just-In-Time acceleration techniques, the simple C/C++ substitution does not attain much help in term of speed. GPU parallel computation would speeds up intensive computation tremendously.

A possible application of the kernel density estimation by using GPU is the semi-parametric discrete response model.

name	URL	Comment
Aptech GAUSS	www.aptech.com/g11_timetrials.html	parallel
parallel for GAUSS	www.aptech.com/pdf/Parallel_1.1.pdf	The third party for GAUSS
JACKET	www.accelereyes.com	The third party for MATLAB
ATLAS	http://math-atlas.sourceforge.net	a new algebra library
Gogoblas2	http://www.tacc.utexas.edu/tacc-projects	GotoBLAS2 include lapack
CUDA	www.nvidia.com/object/io_1209386593154.html	MATLAB mathematica, python
Cygwin	http://www.cygwin.com/	a visual Unix environment in Windows
Wingw_W64	http://mingw-w64.sourceforge.net	GCC for Windows x64
Lev-Mar c++ library	www.ics.forth.gr/ lourakis/levmar	for nonlinear optimization
CULA	http://www.culatools.com	GPU computation library
CUDPP	http://gpgpu.org/developer/cudpp	CUDA Data Parallel Primitives Library
Thrust	http://code.google.com/p/thrust	GPU template library

Table 3.4: Web references

Chapter 4

Multi-Factor CKLS Model and Estimation

4.1 Introduction

Term structure of interest rates have been extensively studied both from theoretical and empirical points. Modeling interest rates have been one of the most popular research issues in financial economics. The purpose of interest rate modeling is to price and hedge interest rate products through understanding interest rate behavior. The best model is to predict future movements in interest rate and prices of interest rate products. Unfortunately, there is no reliable method to accurately predict interest rates. The stochastic nature of interest rates is recognized and was modeled as a random walk by earlier researchers. However,this simple model can not fit the stylized facts of the observant data.

The organization of the paper is as follows. In section 4.2 we present an overview of theoretical interest rate models. In section 4.3 we discuss the construction of seemingly unrelated regression model for discrete two factor CKLS model and our Markov Chain Monte Carlo (MCMC) algorithm. In section 4.4 we provide estimation results using our MCMC algorithms. The estimation result for the simulated data shows that our algorithm work efficiently: convergence of the MCMC draws is relatively quick and draws are not so autocorrelated. Section 4.5 summarizes the main results and suggests for future research.

4.2 Overview of Interest Rate Models

Modeling interest rate have been one of the most popular research issues in financial economics. Campbell (1995) provides an excellent introduction. Rebonato (2003) gives

a deeper and timely review. James and Webber (2000) and Brigo and Mercurio (2006) cover detailed discussion of the topic. James and Webber (2000) give a simple definition of the term structure of interest rates: "The term structure of interest rates is the set of interest rates for different investment periods or maturities." Most term structures are calculated from the observed prices of government securities, such as Treasury bonds and bills in the United states, which are generally regard as default free. Once the term structure of interest rates is known, the price of a bond with yearly coupons is given by equation (4.1):

$$P_0 = \sum_{t=1}^{N} \frac{C_t}{(1+R_t)^t} + \frac{D}{(1+R_N)^N} \tag{4.1}$$

where D denote bond's face value, C_t the coupon payment at time t, N the time to bond's maturity. R_t is the interest rate (continuously-compounded spot rates) prevailing at time 0 for the maturity t.

The continuous counterpart of equation (4.1) is equation (4.2):

$$P_0 = \int_{t=1}^{N} e^{-tr_t} C_t dt \tag{4.2}$$

where r_t is the instantaneous interest rate (short rate) and C_t the coupon payment at time t.

However,the shape of the term structure is continuously changing. Most of the time, the term structure is upward sloping, which means that yields on the long-term bonds are higher than those on short-term bonds. From time to time we have observed inversions of the interest rates. The inversion means that the long-term rate is lower that the short term rate as it is in the United States in 1973 and the early 1980s. Yan (2001) gives another example: "For much of 2000, the term structure was hump shaped: yields on intermediate-term notes (2-5years) were higher than yield on both long-term(10-30years) bonds and short-term (up to 1 year) bills ". The characteristics of yield curve suggest that the prices of interest rate derivatives are not fixed but continuously changing too. One would have to consider more risk premiums associated with these rates. So, the dynamics of the term structure of interest rates play an pivotal role in predicting prices of interest rate contingent claims and risk management. The need to model the dynamics of interest rate arises.

The stochastic nature of interest rates is recognized and was modeled as a random walk by earlier researchers. However,this simple model can not fit the stylized facts of the observated data. More complicated models were proposed to explain term structure's behaviors and are used to value interest rate contingent claims. Among the models, the continuous-time diffusion process has been given the most attention, which is described by equation (4.3).

$$dr_t = \mu(r_t, t)dt + \sigma(r_t, t)dW_t \tag{4.3}$$

where r_t is instantaneous interest rate, $\mu(r_t, t)$, which is called the drift, is the function of r_t and t. $\sigma(r_t, t)$, which is called diffusion, is also the function of r_t and t. W_t is the standard Brownian motion.

There are several reasons to explain the popularity of the continuous-time models. they link the price of interest rate derivatives to the differential equations, which are easier to work with than difference equations. Maes (2003) state that "The diffusions are attractive for empirical researches because they are fully characterized by their instantaneous conditional mean and variance".

The pioneering continuous-time setting was proposed by Vasicek (1977). He introduced a general no arbitrage framework for bonds and proposed an Ornstein-Uhlenbech (O-U) process:

$$dr_t = \kappa(\bar{r} - r_t)dt + \sigma dW(t) \tag{4.4}$$

In equation (4.4) the short rate is mean reverting: κ, which is greater than zero, measures the speed of mean reversion, $\bar{r}$ is also greater than zero and is the long-run mean to which the short rate is reverting.σ is the instantaneous volatility of the short rate which are assumed to be constant.

The success of this model is mainly due to its possibility of pricing analytically bonds and bond options. If we generalize equation (4.2) to the situation that r_t is a O-U process of (4.4), we get equation (4.5).

$$P_{t,T} = E_t\left\{e^{-\int_t^T r(s)ds}\right\} \tag{4.5}$$

where $P_{t,T}$ is the price of unit amount of zero-coupon-bond at time t for the maturity T. $P_{t,T}$ can be derived as equation (4.6). [1]

$$P_{t,T} = A(t,T)e^{-B(t,T)r(t)} \tag{4.6}$$

where

$$\begin{aligned} A(t,T) &= exp\left\{\left(\theta - \frac{\sigma^2}{2k^2}\right)[B(t,T) - T + t] - \frac{\sigma^2}{4k}B(t,T)^2\right\} \\ B(t,T) &= \frac{1}{k}\left[1 - e^{-k(T-t)}\right] \end{aligned}$$

From equation (4.6), the continuously-compounded spot rates [2]are given by (4.7).

$$\begin{aligned} R(t,T) &= -\frac{ln(A(t,T)}{T-t} + \frac{B(t,T)}{T-t}r_t \\ &= a(t,T) + b(t,T)r_t \end{aligned} \tag{4.7}$$

Using equation (4.7), we can calculate a long-term interest rate $R(t,T_1)$ and a short-term interest rate $R(t,T_2)$ at time t. T_1 and T_2 are maturity time that time T_1 is longer than time T_2. Their correlation is calculated by equation (4.8).

$$Corr(R(t,T_1), R(t,T_2)) = Corr(a(t,T_1) + b(t,T_1)r_t, a(t,T_2) + b(t,T_2)r_t) = 1 \tag{4.8}$$

So, by the above discussion, Brigo and Mercurio (2006) state that under the Vasicek model, "at every time instant rates for all maturities in the curve are perfectly correlated. ... This means that a shock to the interest rate curve at time t is transmitted equally through all maturities ... and curve moves almost rigidly in the same direction." In the market data, there is no such perfect-correlation feature of the model. In fact, interest rates are known to exhibit non-perfect correlation. In addition, this model is subject to generating negative interest rates because of the Gaussian distribution. However, there is no negative interest rate in practice. So, more satisfactory model of interest rates need to be found.

---

[1]For the detail of calculation, refer to (Vasicek 1977). This equation is from (Brigo and Mercurio 2006).

[2]there is a definition of this term in (Brigo and Mercurio 2006) and it is sometime used as interest rate.

Cox, Ingersoll, and Ross (CIR, 1985) use a square root process for the short rate and preclude negative short rates .

$$dr_t = (\alpha + \beta r_t)dt + \sigma\sqrt{\gamma(t)}dWt \tag{4.9}$$

Other less popular models are summarized in the table (4.1).

Table 4.1 Here

All of them can be nest by the most general Chan, Karolyi, Longstaff and Sanders (CKLS 1992) model which is becoming attractive because of it's flexibility. CKLS model used the following stochastic process to specify the general form of the short-term interest rate.

$$dr = (\alpha + \beta r)dt + \sigma r^{\gamma} dW_t \tag{4.10}$$

(CKLS 1992) found that the models that best describe the dynamics of interest rates over time are those that allow the conditional volatility of the changing in interest rates to be highly dependent on the level of the interest rate, i.e. $\gamma \geq 1$.

However, One-factor models, such as the Vasicek, CIR and CKLS model, assume that changes in the interest rates of all maturities are driven by changes in a single underlying random factor which is the short rate. These kind of models have the same problem of perfect correlation of different maturities interest rates. In addition, the one-factor models can only accommodate yield curves that are monotonically increasing, monotonically decreasing, or normally humped (*i.e.* $\frown$ -shaped) (Kan, 1992). An inversely humped (*i.e.* $\smile$-shaped) or any other yield curve cannot be generated. These problems indicate the necessity for multifactor models of the term structure.

There are issues involving the multifactor interest models. How many factors should be chosen and how to identify the observable factors in the implementation of models are two main issues. In the two-factor model of Duffie and Kan (1996) , factors are identified by any two bonds yields. In the models of Ho, Stapleton and Subrahmanyam (1993), two factors are short rate and forward rate. In Longstaff and Schwartz (1992), short rate and conditional volatility are two state variables. Brennan and Schward

(1979) developed a two-factor model based on dynamics of two yields on the curve. The two factors are represented by the short term yield and long term yield. Sorwar (2005) identify the two factors that are the same as Brennan and Schward, based on a different dynamic process. He uses CKLS dynamics to control the evolution of each factor.

$$\begin{aligned} r_t &= x_t + y_t \\ dx(t) &= (\beta_1 + \beta_2 x(t))dt + \sigma_1 x^{\gamma_1}(t) dW_1(t) \\ dy(t) &= (\beta_3 + \beta_4 y(t))dt + \sigma_2 y^{\gamma_2}(t) dW_2(t) \end{aligned} \tag{4.11}$$

where $dW_1(t)$ and $dW_2(t)$ are correlated Brownian Motion such that

$$dW_1(t)dW_2(t) = \rho dt$$

The main drawback of the CKLS model is that it does not have a closed-form of price of bonds and other interest derivatives. The key step for numerical valuation is getting the right parameters. Since the model involve nonlinear function of parameters, the Bayesian inference method is a good candidate for estimation. This paper combine the seemingly unrelated regressions model and MCMC method to estimate the two-factor CKLS model.

4.3 Estimation of Two-Factor CKLS Model

In this section, we construct a new MCMC algorithm to estimate the two-factor CKLS model. The advantage of this algorithm is that it has fewer blocks and has closed-form of conditional density function for more parameters compared to the Sorwar (2005) method.

Both the short rate and the long rate are assumed to follow the CKLS process where the conditional mean and variance are dependent on the prevailing rate. It is assumed the risk-adjusted factors are generated by processes given in equation (4.11). For parameter estimaiton purposes, the first order Euler approximation of equations

(4.11) are

$$x_{t+1} - x_t = (\alpha_1 + \beta_1 x_t)\triangle t + \sigma_1 x_t^{\gamma_1} \varepsilon_{1,t+1}\sqrt{\triangle t}$$
$$y_{t+1} - y_t = (\alpha_2 + \beta_2 y_t)\triangle t + \sigma_2 y_t^{\gamma_2} \varepsilon_{2,t+1}\sqrt{\triangle t} \tag{4.12}$$

Rearranging equations (4.12) by the method used by (Eraker 2001) gives the following matrix equations:

$$\frac{x_{t+1} - x_t}{x_t^{\gamma_1}\sqrt{\triangle t}} = \frac{\sqrt{\triangle t}}{x_t^{\gamma_1}}\beta_1 + \frac{x_t\sqrt{\triangle t}}{x_t^{\gamma_1}}\beta_2 + \eta_1$$
$$\frac{y_{t+1} - y_t}{y_t^{\gamma_1}\sqrt{\triangle t}} = \frac{\sqrt{\triangle t}}{y_t^{\gamma_1}}\beta_3 + \frac{y_t\sqrt{\triangle t}}{y_t^{\gamma_1}}\beta_4 + \eta_2 \tag{4.13}$$

where

$$\begin{pmatrix} \eta_1 \\ \eta_2 \end{pmatrix} = \begin{pmatrix} \sigma_1\epsilon_1 \\ \sigma_2\epsilon_2 \end{pmatrix} \sim N\left(\mathbf{0}\ ,\ \ \Sigma \right)$$

By the assumption about two Brownian Motions, we have

$$\Sigma = \begin{pmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{pmatrix}$$

Simplifying the equation (4.13), we get

$$Y_t^1 = X_{1,t}^1\beta_1 + X_{2,t}^1\beta_2 + \eta_1$$
$$Y_t^2 = X_{1,t}^2\beta_3 + X_{2,t}^2\beta_4 + \eta_2 \tag{4.14}$$

where

$$\begin{array}{lllllll} Y_t^1 & = & \frac{x_{t+1}-x_t}{x_t^{\gamma_1}\sqrt{\triangle t}}, & X_{1,t}^1 & = & \frac{\sqrt{\triangle t}}{x_t^{\gamma_1}}, & X_{2,t}^1 = \frac{x_t\sqrt{\triangle t}}{x_t^{\gamma_1}} \\ Y_t^2 & = & \frac{y_{t+1}-y_t}{y_t^{\gamma_1}\sqrt{\triangle t}}, & X_{1,t}^2 & = & \frac{\sqrt{\triangle t}}{y_t^{\gamma_1}}, & X_{2,t}^2 = \frac{y_t\sqrt{\triangle t}}{y_t^{\gamma_1}} \end{array} \tag{4.15}$$

Equation (4.14) is a seemingly unrelated model:

$$\begin{bmatrix} Y_t^1 \\ Y_t^2 \end{bmatrix} = \begin{bmatrix} X_{1,t}^1 & X_{2,t}^1 & 0 & 0 \\ 0 & 0 & X_{1,t}^2 & X_{2,t}^2 \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix} + \begin{bmatrix} \eta_1 \\ \eta_2 \end{bmatrix}$$

The compact equation for the model can be writen as equation (4.16).

$$Y = X\beta + \eta \tag{4.16}$$

where Y is of dimension $(N \times 2)$,X is of dimension of $(2N \times 4)$. The distribution of η is given by

$$\eta \sim N(0, \Sigma \otimes I_N) \tag{4.17}$$

Using $f(\cdot)$ as generic notation for a probability density function, the likelihood function for $\beta = (\beta_1, \beta_2, \beta_3, \beta_4)'$, Σ, γ_1 and γ_2 can be writen as

$$f(data|\beta, \Sigma, \gamma_1, \gamma_2) = (2\pi)^{-2N/2}|\Sigma|^{-N/2}\exp\{-\frac{1}{2}(Y - X\beta)'(\Sigma^{-1} \otimes I_N)(Y - X\beta)\} \tag{4.18}$$

where Y and X are function of data and parameters because they are transformed by equation (4.15) from the original data.

Using the result (9) on page 42 of L*ü*tkepohl (1996), the likelihood function can be written as equation (4.19)

$$f(data|\beta, \Sigma, \gamma_1, \gamma_2) = (2\pi)^{-2N/2}|\Sigma|^{-N/2}\exp\{-\frac{1}{2}tr(A\Sigma^{-1})\} \tag{4.19}$$

where $A = (Y - X^*B)'(Y - X^*B)$ and Y is the $(N \times 2)$ matrix $Y = (Y^1, Y^2)$, X^* is the $(N \times 4)$ matrix $X^* = (X^1, X^2)$, and B is the (4×2) matrix

$$B = \begin{bmatrix} \beta_1 & 0 \\ \beta_2 & 0 \\ 0 & \beta_3 \\ 0 & \beta_4 \end{bmatrix} \tag{4.20}$$

Following Griffiths (2001), we use the conventional conditional noninformative prior for β and Σ:

$$f(\beta, \Sigma, |\gamma_1, \gamma_2) = f(\beta)f(\Sigma) \propto |\Sigma|^{-(M+1)/2} \tag{4.21}$$

where M is the size of matrix A, in this case $M = 2$. So, the joint conditional posterior pdf for β, Σ based on that γ_1, γ_2 are known is

$$\begin{aligned} f(\beta, \Sigma|\gamma_1, \gamma_2, data) &\propto f(y|\beta, \Sigma, \gamma_1, \gamma_2)f(\beta, \Sigma|\gamma_1, \gamma_2) \\ &\propto |\Sigma|^{-(N+M+1)/2}\exp\{-\frac{1}{2}(Y - X\beta)'(\Sigma^{-1} \otimes I_N)(Y - X\beta)\} \\ &= |\Sigma|^{-(N+M+1)/2}\exp\{-\frac{1}{2}tr(A\Sigma^{-1})\} \end{aligned} \tag{4.22}$$

Conditional Posterior pdf for $(\beta|data, \Sigma, \gamma_1, \gamma_2)$ **:**

From the second line in equation (4.22) , we can easily derive the conditional posterior pdf for $(\beta|data, \Sigma, \gamma_1, \gamma_2)$:

$$f(\beta|\Sigma, \gamma_1, \gamma_2, data) \quad \propto \quad \exp\{-\frac{1}{2}(\beta - \hat{\beta})' X'(\Sigma^{-1} \otimes I_N) X(\beta - \hat{\beta})\} \tag{4.23}$$

where

$$\hat{\beta} = \left[X'(\Sigma^{-1} \otimes I_N)X\right]^{-1} X'(\Sigma^{-1} \otimes I_N)Y \tag{4.24}$$

From equations (4.23) and (4.24), we can conclude that β has a multinormal distribution with the mean $\hat{\beta}$ and variance $\left[X'(\Sigma^{-1} \otimes I_N)X\right]^{-1}$.

Conditional Posterior pdf for $(\Sigma|data, \beta, \gamma_1, \gamma_2)$ **:**

From equation (4.22), we can see that the conditional posterior for Σ is the Inverted Wishart distribution

$$f(\Sigma|\beta, \gamma_1, \gamma_2, data) \quad \propto \quad |\Sigma|^{(-N+M+1)/2} \exp\{-\frac{1}{2} tr(A\Sigma^{-1})\} \tag{4.25}$$

Conditional Posterior pdf for $(\gamma_i|data, \beta, \Sigma), i = 1, 2$ There is no closed form for conditional posterior pdf for γ_1 and γ_2. So, we use a Metropolis-Hastings Algorithm to sample γ_1 and γ_2 using normal distribution as the proposal density. From equation (4.23), Eraker (2001) gives the conditional posterior density of γ_1 by equation (4.26) and that of γ_2 by equation (4.27).

$$\begin{aligned} f(\gamma_1|\beta, \Sigma, data) \quad \propto \quad & \prod_{i=1}^{N} \frac{1}{\sigma_1 x_{i-1}^{\gamma}} \\ & \times \exp\left\{-\frac{1}{2}\frac{(x_i - x_{i-1} - (\beta_1 + \beta_2 x_{i-1})\triangle t)^2}{\sigma^2 x_{i-1}^{2\gamma_1} \triangle t}\right\} \end{aligned} \tag{4.26}$$

$$\begin{aligned} f(\gamma_2|\beta, \Sigma, data) \quad \propto \quad & \prod_{i=1}^{N} \frac{1}{\sigma_2 y_{i-1}^{\gamma}} \\ & \times \exp\left\{-\frac{1}{2}\frac{(y_i - y_{i-1} - (\beta_3 + \beta_4 y_{i-1})\triangle t)^2}{\sigma^2 y_{i-1}^{2\gamma_2} \triangle t}\right\} \end{aligned} \tag{4.27}$$

Eraker(2001) also give the proposal density to sampling γ_1 and γ_2. It is equation (4.28).

$$N(\gamma_i^{(h-1)} - \frac{g'(\gamma_i^{(h-1)})}{g''(\gamma_i^{(h-1)})}, -(g''(\gamma_i^{(h-1)}))^{-1}), \quad i = 1, 2 \tag{4.28}$$

where $g(\gamma_i) := logf(\gamma_i|\beta, \Sigma, data)$, $g'(\gamma_i)$ is the first derivative of g respect to γ_i and $g''(\gamma_i)$ is the second derivative of g respect to γ_i, $i = 1, 2$.

Sampling Algorithm : According above discussion about the two-factors CKLS model, we now design the Sampling Algorithm for the model:

1. Initialize all unknown parameters.
2. Transforms the data by equation (4.15)
3. Draw $\beta^{(h)}$ from multivariate normal distribution by equation (4.23).
4. Draw $\Sigma^{(h)}$ from the Inverted Wishart distribution in term of equation (4.25) [3].
5. Draw $\gamma_1^{(h)}$ using Metropolis-Hastings using normal distribution as proposal density which is given by equation (4.28) after updating the transformation of data.
6. Draw $\gamma_2^{(h)}$ using same method of drawing $\gamma_1^{(h)}$.
7. return to step 2.

where the superscript (h) denotes the $h - th$ draw.

This sampling algorithm has some advantages compare to that of Sorwar (2005). First, it has closed-form pdf for β. The vector of β can be drawn from a multinormal distribution by one step. The prior in this SUR method is constant. In Sorwar (2005), although each β_i has conditional normal pdf, there are prior parameters included. Sorwar (2005) did not give the algorithm to estimate the parameters in prior or give the value of them. In addition, Sorwar's (2005) method draws β_i for each equation respectively, i.e. in two steps. Secondly, the SUR method also has closed-form pdf for Σ. It can be draw from the invert Wishart distribution in one step. In contrast, Sorwar (2005) have to draw σ_i^2 separately and draw ρ using the truncated normal distribution,

[3]There are several algorithms to draw from the Inverted Wishard Distribution. This paper transforms the draws from the Wishart distribution, which is a function in OX. The details of transformation is in $http://en.wikipedia.org/wiki/Inverse-Wishart_distribution$

which is not an efficient way. Sorwar's way of drawing ρ works for the two factor model. For more than two factors, we have more than one ρ. Drawing ρ's become quickly untenable since we have to make sure that Σ always positive definite.

4.4 Examples of Application

Example 1: numerical illustration In this section, we present a empirical example showing that our MCMC algorithm works for the two factor model. First, we choose parameters and get the simulated data. The sample size is 2000. The path of them is presented in figure 4.1.[4]

Figure 4.1 here

We estimate the two-factor CKLS using the method from the section 2 and result of the estimation is in the table 4.2.

Table 4.2 here

The estimated density of draws are in figure 4.2, 4.3, 4.4 and 4.5. The number of draw is 15000 and burn first 5000.

Figure 4.2 here

Figure 4.3 here

Figure 4.4 here

Figure 4.5 here

The estimation result show that estimation method works very well. The table 4.2 show that estimators of parameters are very close to their true value. All of them are in their 90% highest posterior density interval.

[4]The program used for this paper is OX. The density graphs for all parameters are generated by the DrawDensity function in OX

Example 2 : Application of Real Data In this example, we apply the estimation on the data of 3-month treasury bill interest rates and 1-year treasury bond constant maturity interest rates, which are weekly data and from 7/17/1959 to 8/24/2001. the sample size is 2198. The data is presented in figure 4.6.

Figure 4.6 here

The sample size for this period is 2192. In estimation by MCMC, the number of draws is 15000 and drop off first 5000 draws, the estimation result is in the table 4.3 and visual results are shown in figure 4.7, 4.8, 4.10 and 4.9.

Talbe 4.3

Figure 4.7 here

Figure 4.8 here

Figure 4.10 here

Figure 4.9 here

The estimators of β_1 and β_3 are positive,β_2 and β_4 are negative. That satisfy the key assumptions of mean reverting about interest rates process. The estimators of γ_1 and γ_2 are 1.1290 and 1.1518 respectively. They are close to 1.5 and the result of CKLS (1992), which suggest that the conditional volatility is higher than that in the CIR model. The estimator of ρ, correlation coefficient is 0.744222, which is calculated from Σ. The result mean that the short term yields and the long term yiels are highly correlated and this result coincide with the visual data in figure 4.6.

4.5 Conclusions

The CKLS model nests many popular models for the interest rates. The flexility of the CKLS model has the cost in that it has no close form for the price of the interest rate derivative. The estimation of CKLS model is the key step to value the interest

rate derivative. This paper provides a new method to estimate the two-factor CKLS model. We transform the two-factor CKLS model into a seemingly unrelated regression model and use MCMC to estimate the parameters. Such method can easily derive the conditional posterior density of covariance matrix Σ and the conditional posterior densities of vector β. In the simulation example, most of the parameters are in the 90% highest posterior density interval. When it is applied in real data, the result satisfy the some assumption about interest rate. Although this method can be extend to more than two factor models, as sample size increase, the computation time increase dramatically.

1. Metron (1973)	$dr_t = \alpha dt + \sigma dW_t$	$beta = 0, \gamma = 0$
2. Vasicek (1977)	$dr_t = (\alpha + \beta r_t)dt + \sigma dW_t$	$\gamma = 0$
3. Cox, Ingersoll, and Ross (1985)	$dr_t = (\alpha + \beta r_t)dt + \sqrt{\sigma} dW_t$	$\gamma = 1/2$
4. Dothan (1978)	$dr_t = \sigma r_t dW_t$	$\alpha = \beta = 0, \gamma = 1$
5. Geometric Brownian Motion	$dr_t = \beta r_t dt + \sigma r_t dW_t$	$\alpha = 0, \gamma = 1$
6. Brennan and Schwartz (1980)	$dr_t = (\alpha + \beta r_t)dt + \sigma r_t dW_t$	$\gamma = 1$
7. Cox, Ingersoll, and Ross (1980)	$dr_t = \sigma \gamma^{2/3} dW_t$	$\alpha = \beta = 0, \gamma = 2/3$
8. Constant Elasticity of Variance	$dr_t = \beta r_t dt + \sigma r_t^{\gamma} dW_t$	$\alpha = 0$

Table 4.1: Interest Rate Models nested by CKLS, This table is from Brigo and Mercurio (2006)

parameter	True Value	Est. Value (mean)	Std. Dev.	AR(1) Coe.	Quantitle(5%,95%)
β_1	0.0014390	0.0019385	0.00036197	0	(0.0013316,0.0025214)
β_2	-0.019200	-0.027656	0.0060572	0	(-0.037422,-0.017549)
β_3	0.001310	0.0011527	0.00029353	0	(0.00068393,0.0016440)
β_4	-0.015200	-0.015307	0.0045583	0	(-0.022856,-0.0079612)
σ_1^2	0.017500	0.013926	0.0035131	0	(0.0090043,0.021552)
σ_2^2	0.01800	0.0025399	0.00061640	0	(0.0016646,0.0036613)
σ_{12}	0.003500	0.015306	0.0039530	0	(0.0098026,0.022396)
γ^1	1.2312	1.1866	0.045861	0	(1.1120,1.2606)
γ^2	1.3599	1.3257	0.047040	0	(1.2469,1.4027)

Table 4.2: Estimation Result for Exam 1

Para.	Est.Val.(mean)	Std. Dev.	AR(1) Coe.	Quantitle(5%,95%)
β_1	0.0081649	0.0053864	0	(-0.00085771 , 0.016899)
β_2	-0.0010579	0.0012184	0	(-0.0030455 , 0.00095806)
β_3	0.0093077	0.0046221	0	(0.0017178 , 0.016987)
β_4	-0.0010929	0.0011685	0	(-0.0030025 , 0.00083652)
σ_1^2	0.00036871	4.2845e-005	0	(0.00030347 , 0.00044411)
σ_{12}^2	0.00031867	4.2845e-005	0	(0.00027219 , 0.00036895)
σ_2^2	0.00049727	4.4954e-005	0	(0.00042865 , 0.00057636)
γ^1	1.1290	0.031983	0	(1.0758 , 1.1812)
γ^2	1.1518	0.025080	0	(1.1099 , 1.1925)

Table 4.3: Estimation Result for Exam 2

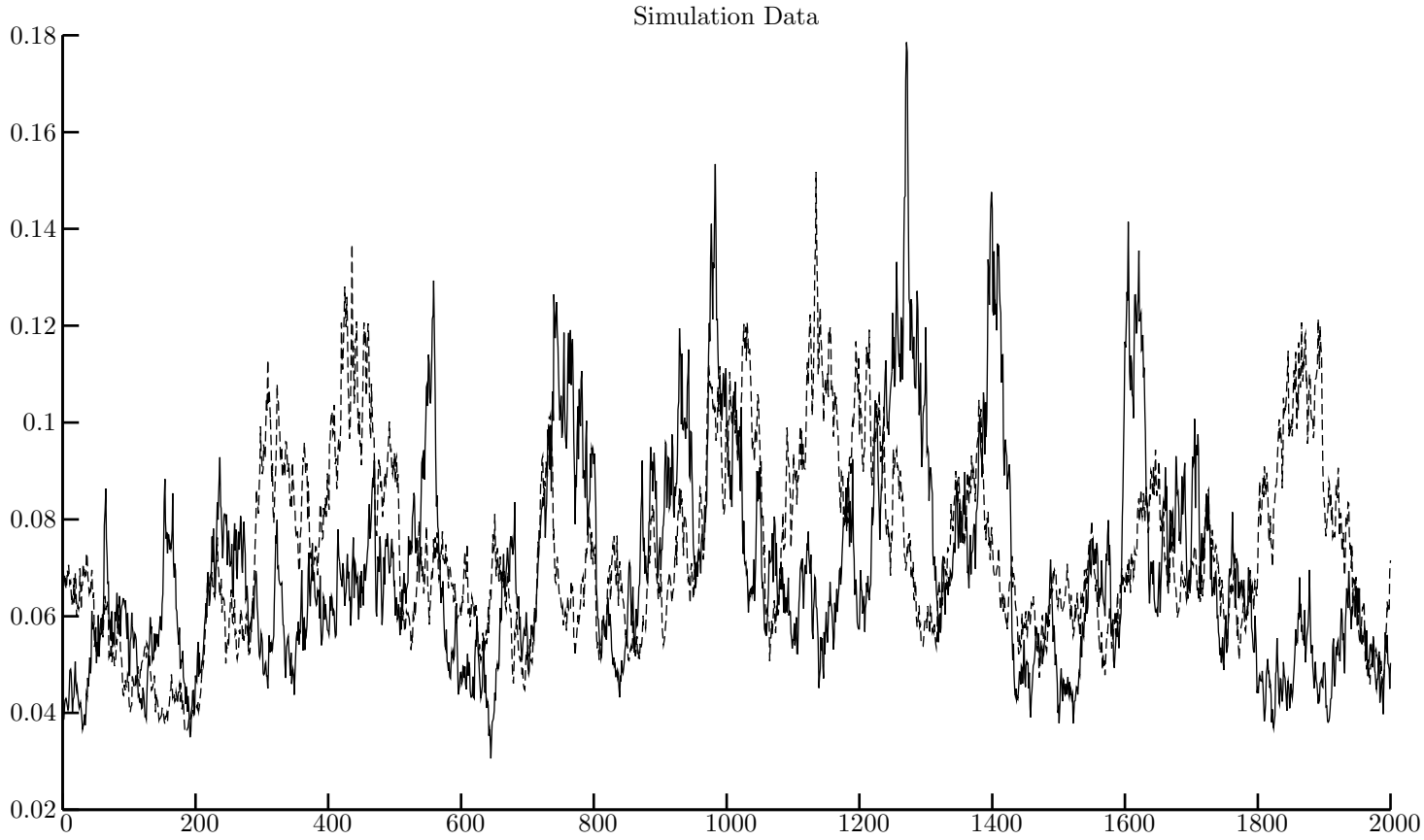

Figure 4.1: Simulation Data for Exam 1

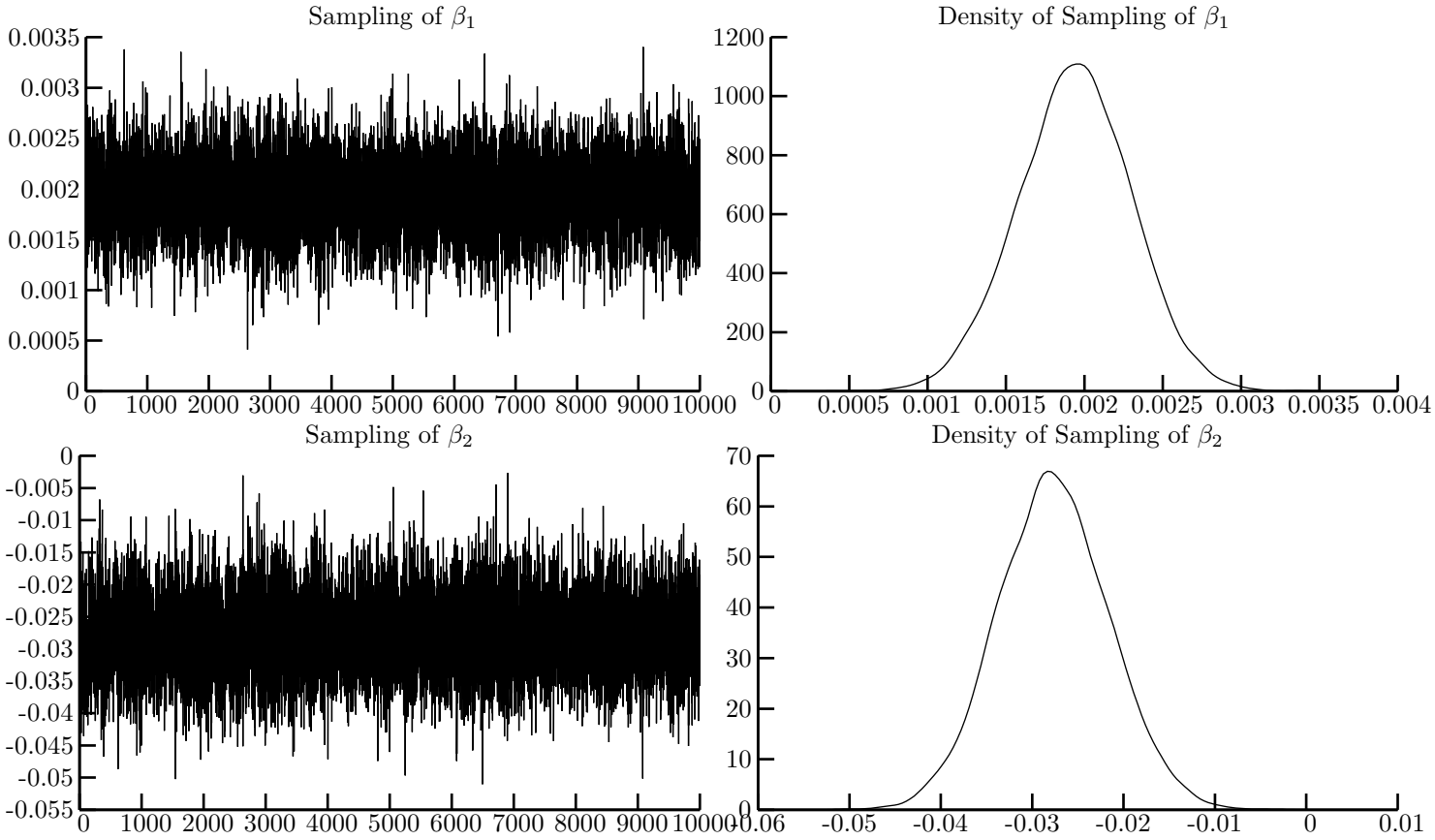

Figure 4.2: Estimators of β_1 and β_2 in Exam 1

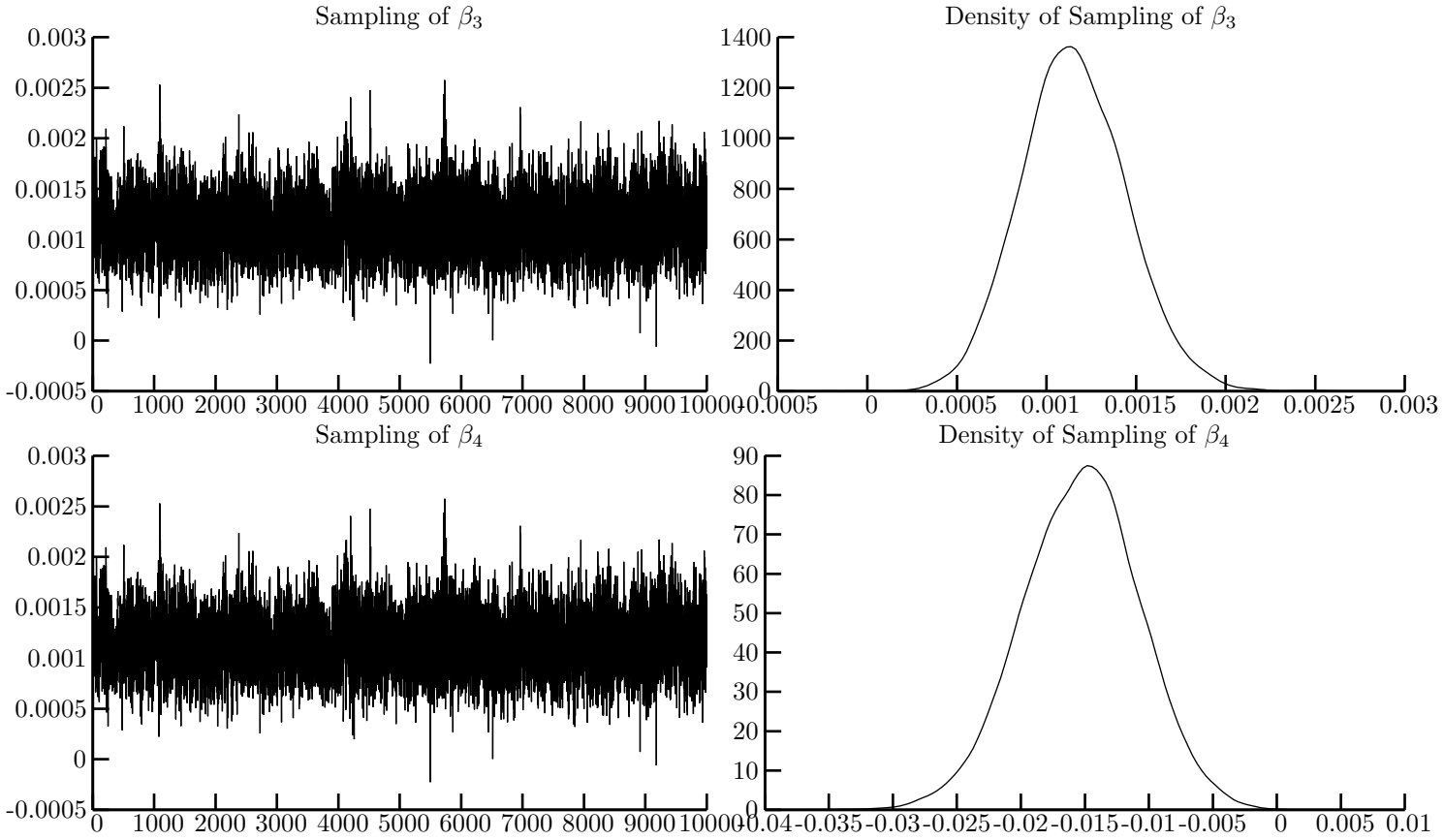

Figure 4.3: Estimators of β_3 and β_4 in Exam 1

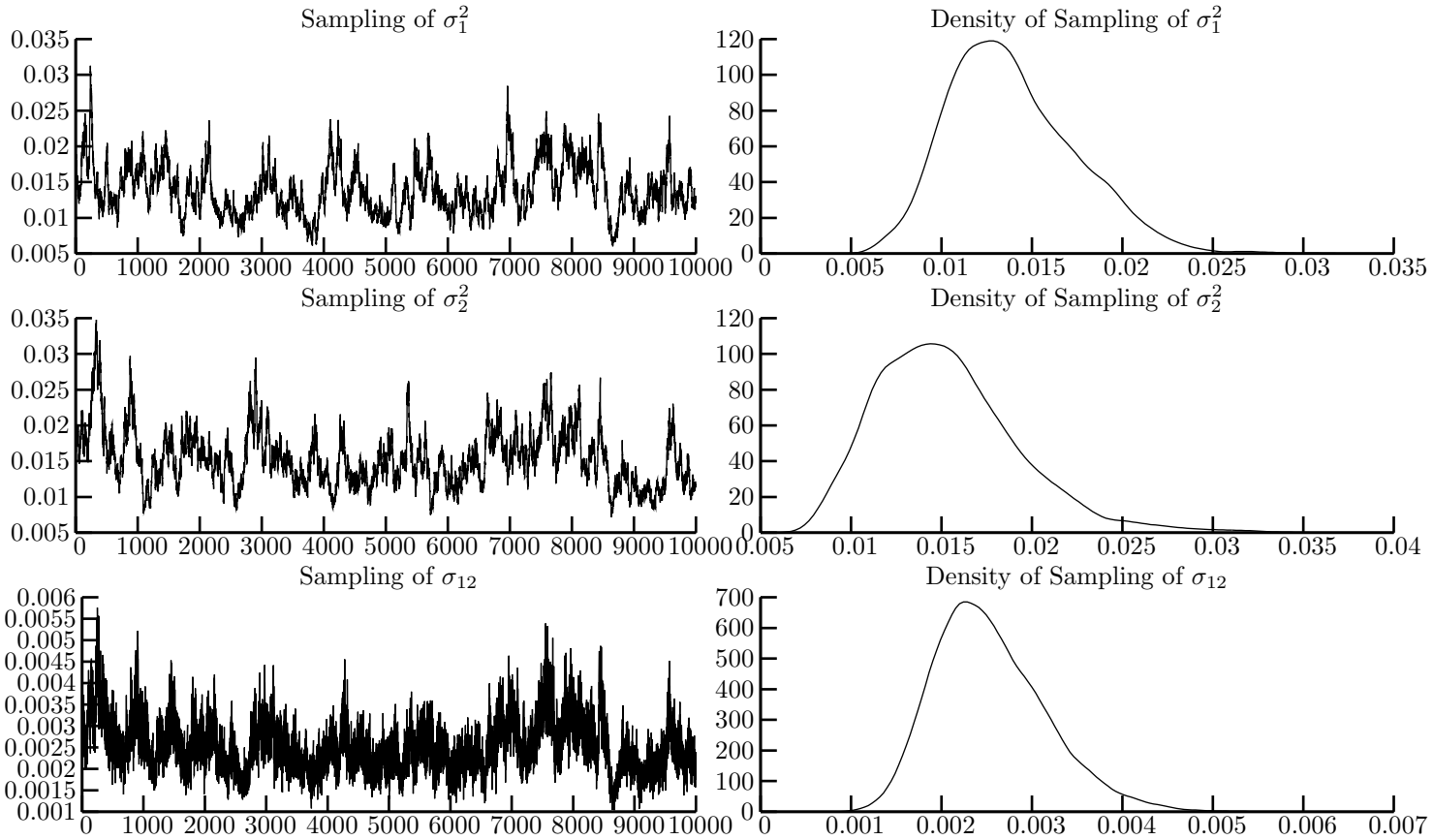

Figure 4.4: Estimators of Σ in Exam 1

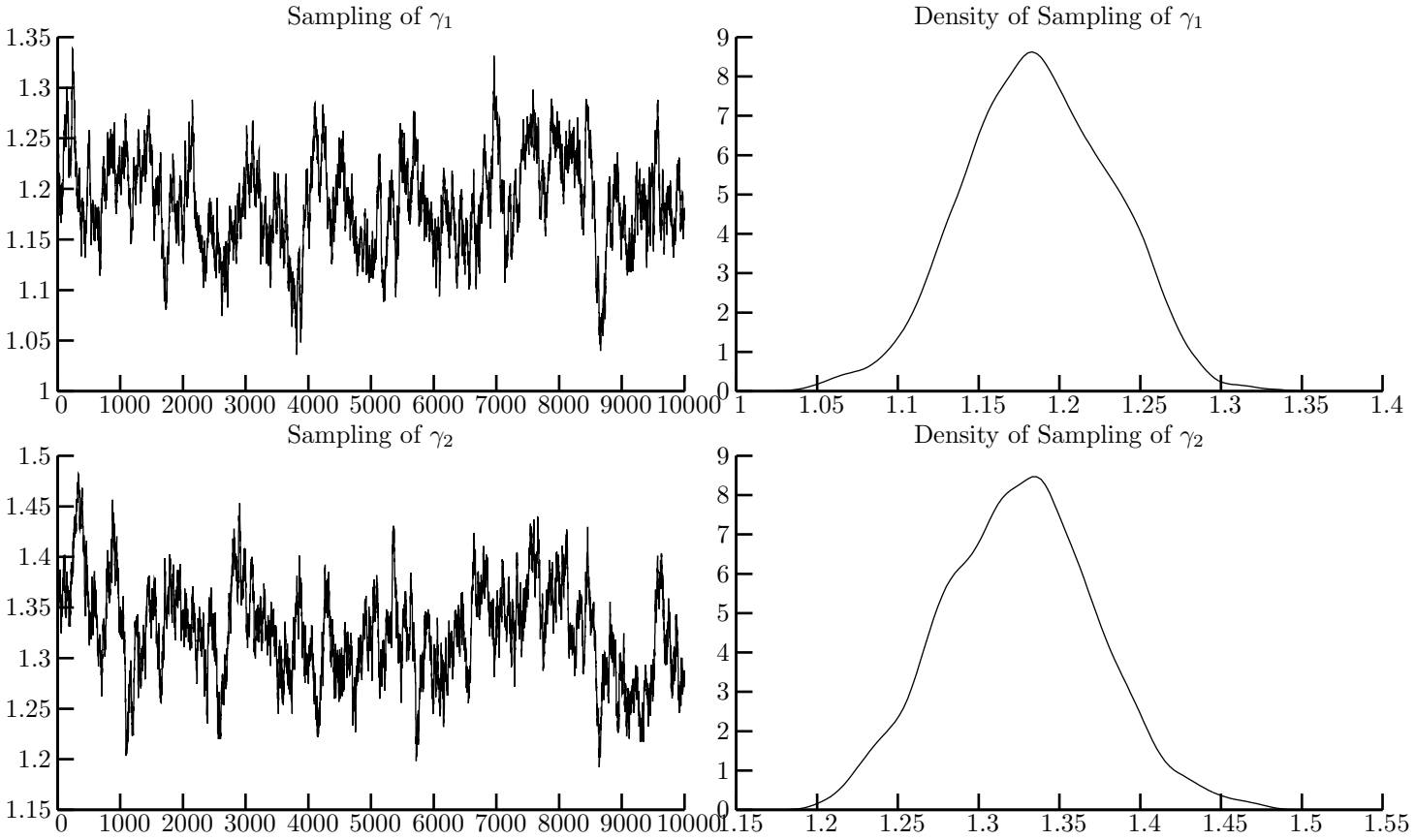

Figure 4.5: Estimators of γ_1 and γ_2 in Exam 1

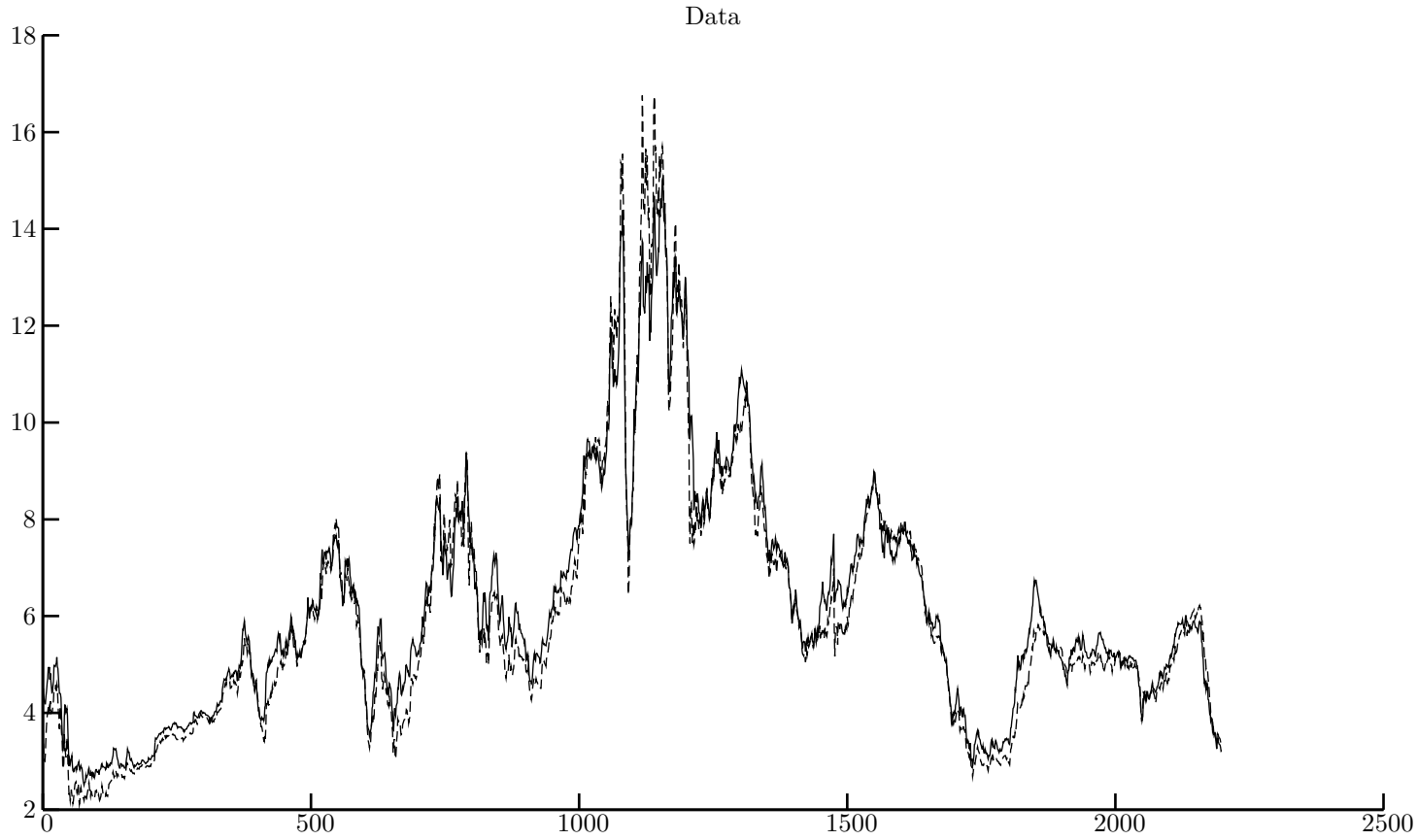

Figure 4.6: Data:market rates for 1 year and 3 month treasury bonds

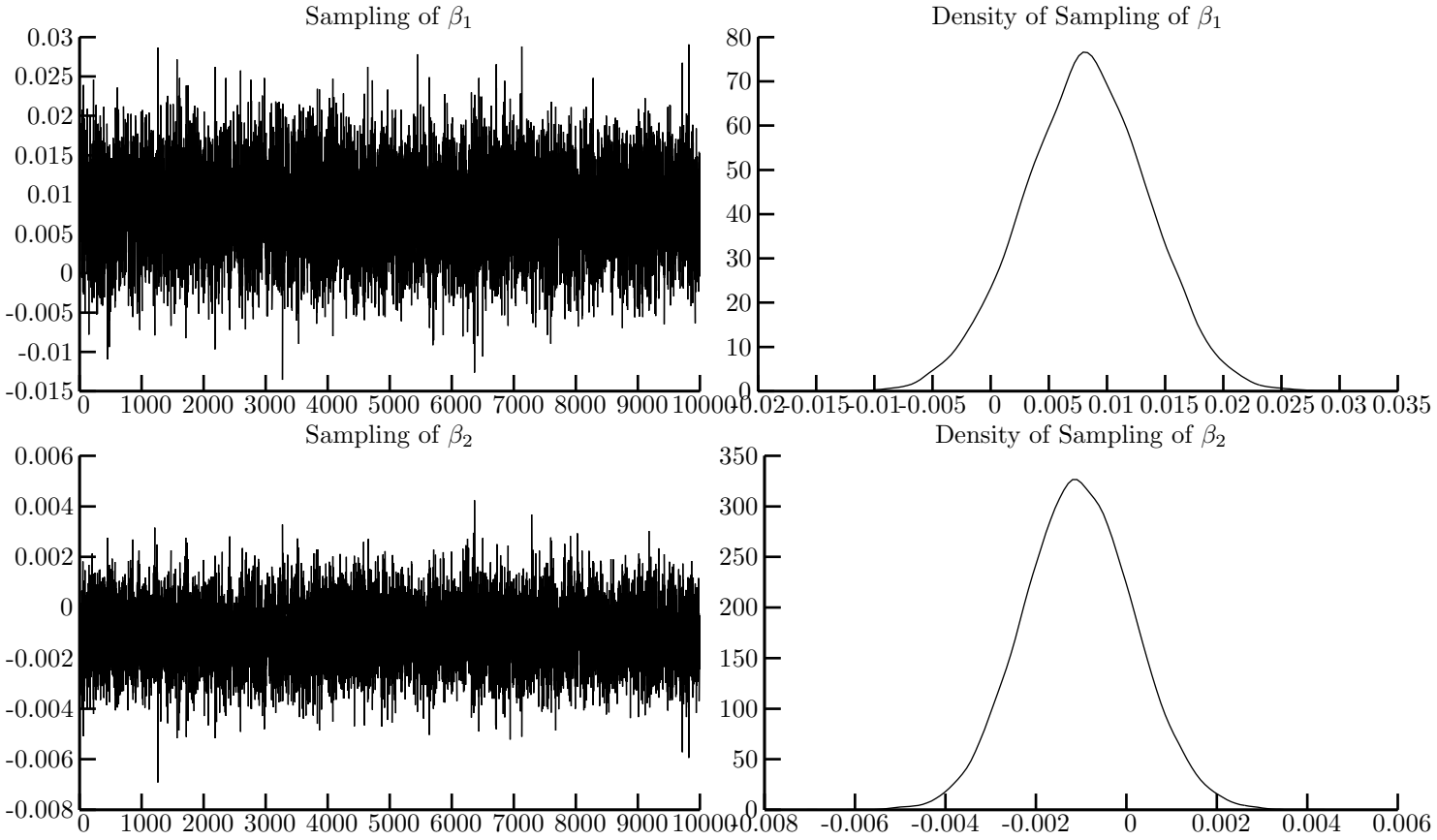

Figure 4.7: estimators of : β_1 and β_2 in Exam 2

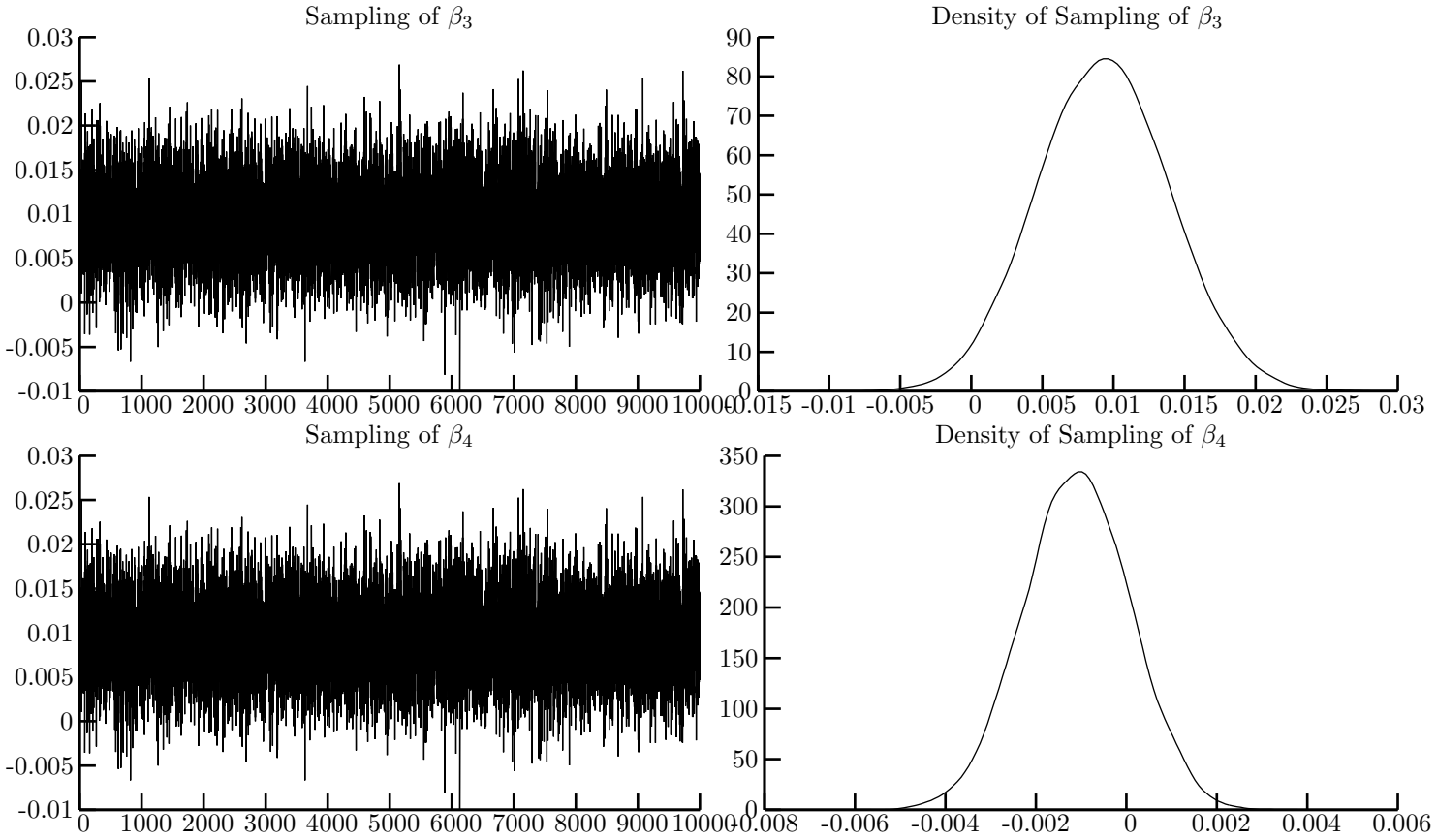

Figure 4.8: estimators of :β_3 and β_4 in Exam 2

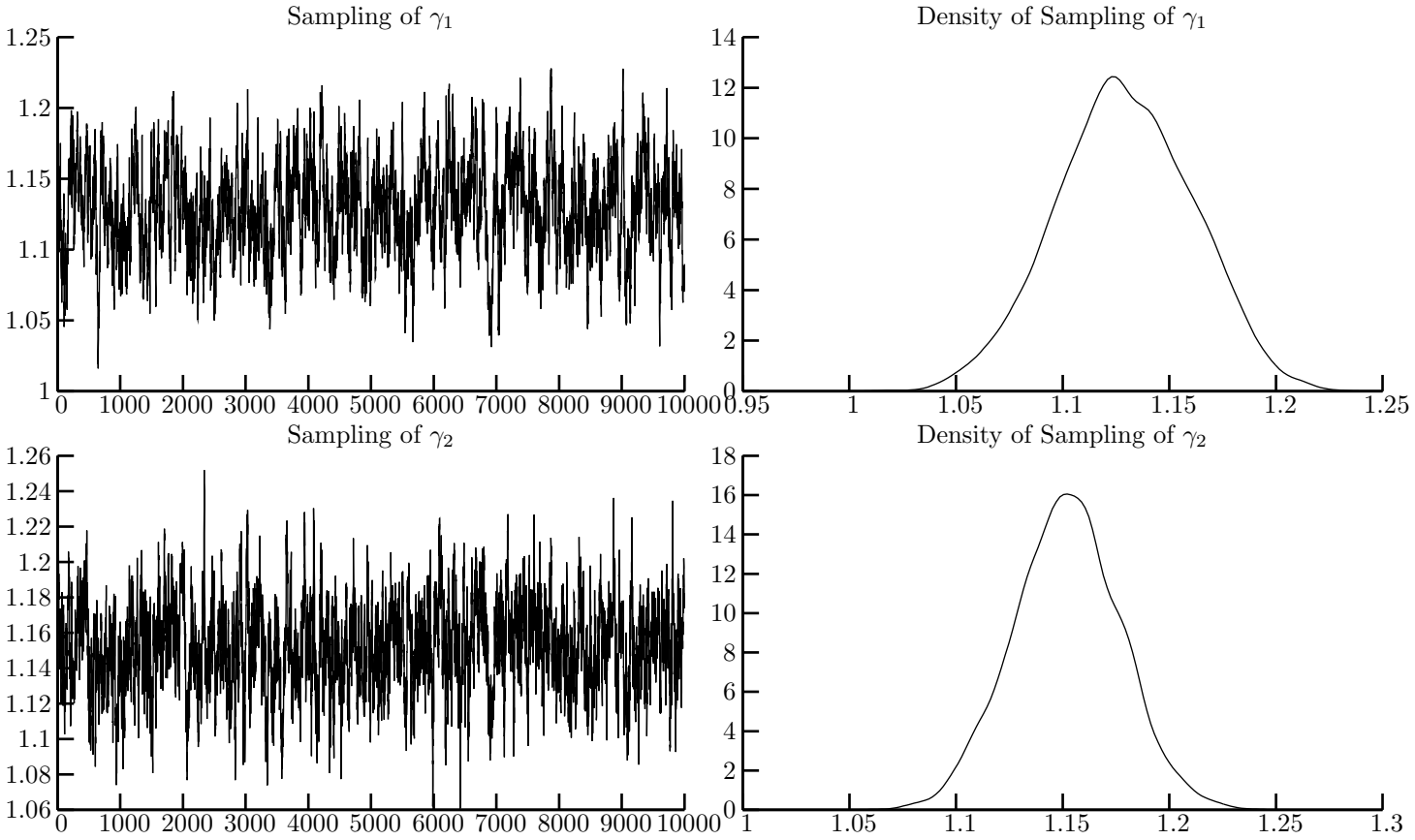

Figure 4.9: estimators of γ_1 and γ_2 in Exam 2

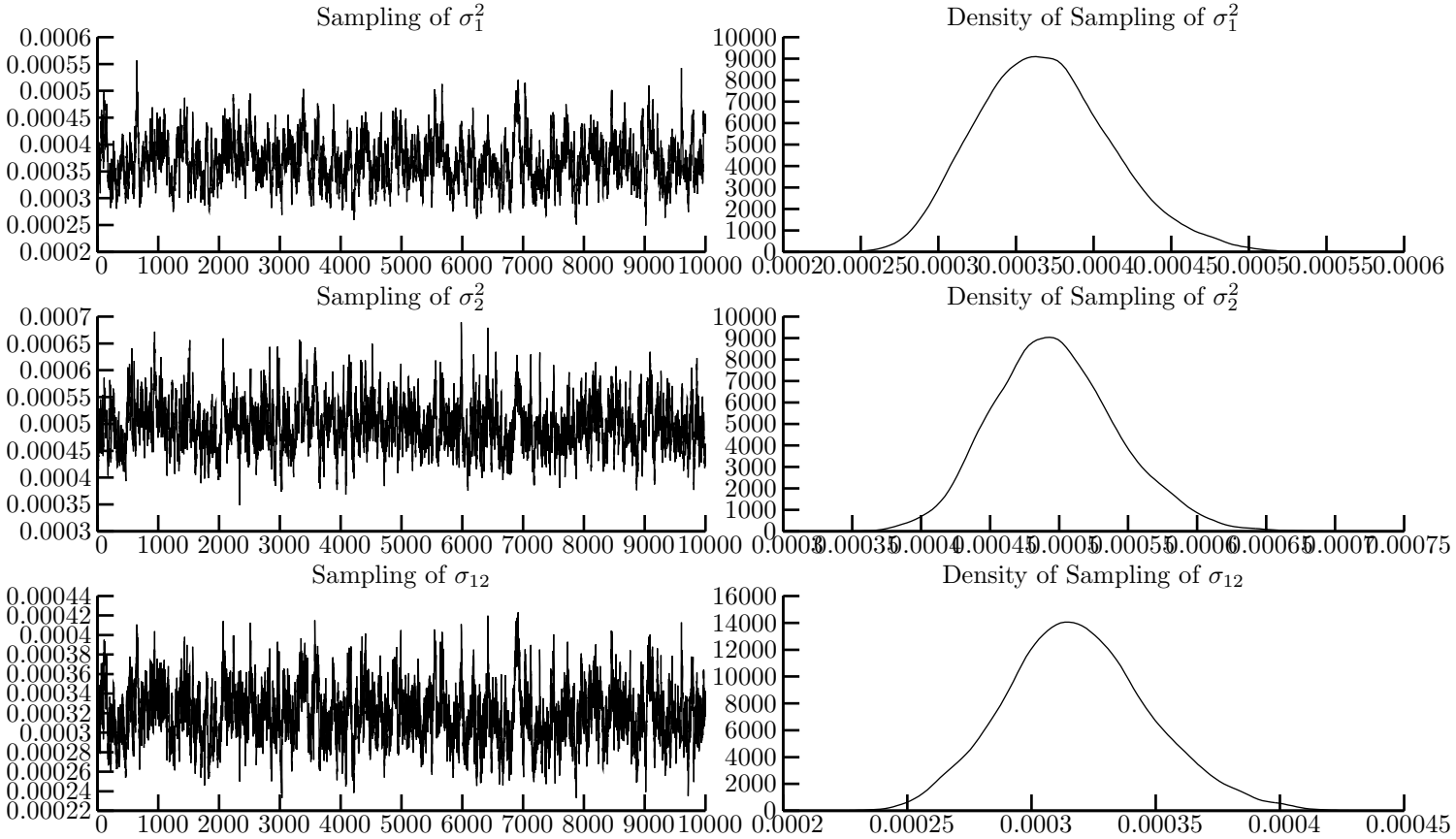

Figure 4.10: Estimators of Σ in Exam 1

References

[1] J.F. Botev, Z.I. Grotowski and D.P. Kroese. Kernel density estimation via diffusion. *The Annals of Statistics*, 38(5), 2010.

[2] Adrian W. Bowman and Adelchi Azzalini. *Applied Smoothing Techniques for Data Analysis: The Kernel Approach with S-Plus Illustrations (Oxford Statistical Science Series)*. Oxford University Press, USA, November 1997.

[3] M. Brennan and E. Schwartz (1979). A continuous time approach to the pricing of bonds. *Jouranl of Banking and Finance*, 3:133–155, 1979.

[4] M. Brennan and E. Schwartz(1980). Analyzing conbertible bonds. *Journal of Financial Quantitative Analysis*, 15:907–929.

[5] Damiano Brigo and Fabio Mercurio. *Interest Rate Models - Theory and Practice*. Springer Varlag, Berlin, 2006.

[6] John Y Campbell. Some lessons from the yield curve. *Journal of Economic Perspectives*, 9(3):129–52, 1995.

[7] John C Cox, Jr Ingersoll, Jonathan E, and Stephen A Ross (1981). A reexamination of traditional hypotheses about the term structure of interest rates. *Journal of Finance*, 36(4):769–99, 1981.

[8] John C Cox, Jr Ingersoll, Jonathan E, and Stephen A Ross (1985a). An intertemporal general equilibrium model of asset prices. *Econometrica*, 53(2):363–84, 1985.

[9] John C Cox, Jr Ingersoll, Jonathan E, and Stephen A Ross (1985b). A theory of the term structure of interest rates. *Econometrica*, 53(2):385–407, 1985.

[10] John C Cox, Jr Ingersoll, Jonathan E, and Stephen A Ross(1980). An analysis of variable rate loan contracts. *Journal of Finance*, 35(2):389–403.

[11] Jurgen A. Doornik, Neil Shephard, and David F. Hendry. Parallel computation in econometrics: A simplified approach. Economics Papers 2004-W16, Economics Group, Nuffield College, University of Oxford, January 2004.

[12] O.A. Dothan(1978). On the term structure of interest rates. *Journal of Financial Economics*, 6:59–69.

[13] R.F. Engle and V.K. Ng. Measuring and testing the impact of news on volatility. *Journal of Finance*, 48:1749–1778, 1993.

[14] Bjorn Eraker (2001). Mcmc analysis of diffusion models with application to finance. *Journal of Business & Economic Statistics*, 19(2):177–91.

[15] E. Goldman, E. Valieva, and H. Tsurumi. Kolmogorov-smirnov, fluctuation and z_g tests for convergence of markov chain monte carlo draws. *Communications in Statistics, Simulation and Computation*, 37(2):368–380, 2008.

[16] Kazushige Goto and Robert A. van de Geijn. Anatomy of high-performance matrix multiplication. *ACM Trans. Math. Softw.*, 34:12:1–12:25, May 2008.

[17] Kazushige Goto and Robert A. Van De Geijn. High-performance implementation of the level-3 blas. *ACM Trans. Math. Softw.*, 35(1):1–14, 2008.

[18] W.E. Griffiths. Bayesian inference in the seemingly unrelated regressions models. (793), 2001.

[19] T.S. Ho, Richard C. Stapleton, and Marti G. Subrahmanyam. A two factor no-arbitrage model of the term structure of interest rates. New York University, Leonard N. Stern School Finance Department Working Paper Seires 96-29, New York University, Leonard N. Stern School of Business-, August 1996.

[20] Jessica James (2000) and Nick Webber. *Interest Rate Modelling*. John Wiley, Chichester, 2000.

[21] J.S. Jones, M. C. Marrron and Sheather. A brief survey of bandwidth selection for density estimation. *Journal of the American Statistical Association*, 91, 1996.

[22] M. C. Jones and D. F Signorini. A comparison of higher-order bias kernel density estimators. *J. Amer. Statist. Assoc*, 92(439):1063–1073, 1997.

[23] Raymond Kan(1992). Shape of the yield curve under cir single-factor model:a note. Technical report, University of Toronto.

[24] Jeremy Kepner. *Parallel MATLAB for Multicore and Multinode Computers*. SIAM, 2009.

[25] David B. Kirk and Wen-mei W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann, 1 edition, February 2010.

[26] H Lütkepohl. *Handbook of Matrices*. John Wiley and Sons, Chichester, 1996.

[27] Liuling Li. Three essays on bayesian analysis of financial econometrics. *Rutgers University PhD dissertation*, 2009.

[28] F. Longstaff and E. Schwartz (1992b). A two-factor interest rate model and contingent claims valuation. *Journal of Fixed Income*, pages 16–23, 1992.

[29] H.B. Madsen, K. Nielsen and O. Tingleff. Methods for non-linear least squares problems. *Lecture notes*, 2004.

[30] Konstantijn Maes (2003). Modeling the term structure of interest rates: Where do we stand. working paper.

[31] J. S. Marron and M. P. Wand. Exact mean integrated squared error. *The Annals of Statistics*, 20(2):712–736, 1992.

[32] Mathwork. *Parallel Computing Toolbox 5.0 User's Guild*. Mathwork, 2010.

[33] Matlab. *Matlab 7 External Interface Guild*. 2009.

[34] A. M. Mineo. The norm-p estimation of location, scale and simple linear regression parameters. *Lecture Notes in Statistics, Statistical Modelling*, 57:222–233, 1989.

[35] A.M. Mineo and M. Ruggieri. A software tool for the exponential power distribution: the normalp package. *Journal of Statistical Softwaare*, 12(4):2–24, 2005.

[36] Teruo Nakatsuma(2000). Bayesian analysis of arma-garch models: A markov chain sampling approach. *Journal of Econometrics*, 95(1):57–69.

[37] K B Nowman. Gaussian estimation of single-factor continuous time models of the term structure of interest rates. *Journal of Finance*, 52(4):1695–1706, September 1997. available at http://ideas.repec.org/a/bla/jfinan/v52y1997i4p1695-1706.html.

[38] Nvidia. Nvidia compute united device architecture (cuda) toolkit, version 3.1. NVidia, 2010.

[39] R. Rebonato. Term structure models: a review. *Royal Bank of Scotland Quantitative Research Centre Working Paper.*, 2003.

[40] Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 1 edition, July 2010.

[41] B.W. Silverman. *Density estimation for statistics and data analysis*. Chapman and Hall, 1986.

[42] P.R. Tadikamalla. Random sampling from the exponential power distribution. *Journal of American Statistical Association*, 75:683–686, 1980.

[43] M.P. Wand and M.C. Jones. *kernel Smoothing*. Chapman and Hall, 1995.

[44] Tobias Wittwer. Choosing the optimal blas and lapack library. 2008.

[45] Hong Yan(2001). Dynamic models of the term struture. *Financial Analysts Journal*, 57(2):425–442.

[46] D. Zhu and V. Zinde-Walsh. Properties and estimation of asymmetric exponential power distribution. *Journal of Econometrics*, 148:86–99, 2009.

Vita

Shiliang Li

2004-2011 Ph. D. in Economics, Rutgers University

2004-2006 M. A. in Economics, Rutgers University

2000-2003 M. A. in Economics, Peking University

1995-1998 B. A. in Taxation, Shandong Economic University

1991-1993 A. A. in Taxation, Shandong University of Science and Technology, Taian, China

2005-2008 Teaching assistant, Department of Economics, Rutgers University