Markov Chain Monte Carlo Simulation on Stochastic Volatility Estimation of S&P500 Returns

Kapil Kanagal Lorenzo Limonta Quan Chen Jiacheng Zou

{kkanagal,llimonta,quanchen,jasonzou}@stanford.edu

May 30, 2017

Contents

Abstract

This paper seeks to explore the applications of Markov Chain Monte Carlo techniques on S&P 500 data in order to estimate latent volatility. The model used is a stochastic volatility model originally proposed by Jacquier and Rossi [8], which uses an AR(1) parametrization for the conditional variance of the return. The model's parameters are fitted on the daily return (Jan 2003 - Dec 2012) by using as explanatory variables the lagged-1 return. The mean squared error obtained is approximately 0.42 with an arbitrarily early stopping condition $R^2 = 80\%$, otherwise within 100 iterations MCMC would reach a $R^2 = 100\%$ thus over-fitting towards the data. We formulate our model in a very generic way so that we can improve robustness by easily changing the priors without changing the updating schemes, and we can add in jump terms or other exogenous covariates without changing the mathematical formulations.

1 Introduction

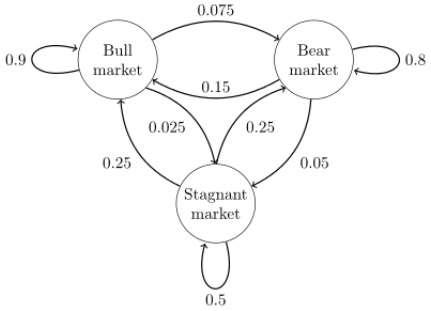
Markov Chain Monte Carlo (MCMC) is a commonly used technique for estimation by simulating the expectation of a statistic in a complex model. Successive random selections form a Markov chain, the stationary distribution of which is the target distribution. It is particularly useful for the evaluation of posterior distributions in complex Bayesian models. In the Metropolis–Hastings algorithm, items are selected from an arbitrary "proposal" distribution and are retained or not according to an acceptance rule. The Gibbs sampler is a special case in which the proposal distributions are conditional distributions of single components of a vector parameter [5].

This has become a commonly used technique to sample from a complicated distribution. Often times, the parameters in a distribution are unknown and resultantly, this makes it extremely difficult to draw from the given distribution. MCMC can be used to estimate these parameters in order to give a clearer picture of the properties of the distribution. This technique combines two other techniques (Markov Chains and Monte Carlo Simulation) to solve this problem.

1.1 Markov Chains

By definition, a Markov Chain is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event. These mathematical systems move from one state to another in the state space – a list of all possible states. With a two state model (states A and B), there are a possible four transitions. A->B, B->A, A->A, B->B. Each move in the Markov Chain is called a step. Instead of using a visual diagram to represent the Markov Chain, a transition matrix is used to tally all the potential transition properties. This allows for all sorts of unique linear algebra usages of these matrices, which can be used to describe even more complex state spaces.[6]

The interesting part of MCMC, is that random walks (i.e. Brownian Motion) can be modeled as Markov Chains. This creates a unique instance, which allows MCMC to model a variety of interesting problems in statistics, biology, physics, and finance. In finance, random walks are often use to model stochastic volatility and pricing models. Below is a simple example of a Markov Chain to give you some intuition for their structure. Recall that the problem we're trying to solve is to draw from a distribution over a finite set X with probability function p(x). The MCMC method is to construct a Markov chain whose stationary distribution is exactly p, even when you just have black-box access to evaluating p.



1.2 Monte Carlo Simulation

The Monte Carlo part of MCMC refers to the simulation portion of the process. Monte Carlo simulation generates a large number of simulated outcomes, from which an average or a probability is estimated. A simple example would be to use Monte Carlo simulation to estimate the probability of a coin toss landing heads ten times in a row. Simulate ten coin tosses, observe if you get ten heads in a row, and repeat this simulation a large number of times (say one million). Out of these 1 million simulations, let's say you observed 967 instances in which you observed ten heads in a row, which means that the probability of getting ten heads in a row is about $967 \,/\, 1000000 \times 100 = 0.0967\%$. Calculating the result mathematically as opposed to using Monte Carlo simulation, we get: $0.5^{10} = 0.00097656 = 0.0977\%$.

Thus, MCMC uses Monte Carlo Simulations on Markov Chains to determine the distribution from which the parameters are drawn and estimate averages for the sample parameters. It does so for each transition in the Markov Chain process that is generated. Now that we have established an understanding of both MC parts (Markov Chains and Monte Carlo Simulation) work, we use the sections below to outline the mathematical background of MCMC, followed by our empirical MCMC simulations run on S&P 500 data. Be sure to remember, that the core problem MCMC tries to solve is a sampling problem.

2 Bayesian Statistics

2.1 Preliminaries

The entire mathematical argument upon which MCMC is predicated on, is the use of Bayesian inference: the ability to combine prior beliefs with data, in order to compute the posterior distribution and make inferences [13]. Bayesian inference is about the quantification and propagation of uncertainty, defined via a probability, in light of observations of the system. Unlike the frequentist approach, Bayesian statistics models both the data and the parameters that govern the model to be probabilistic. Any event that can be witnessed or recorded, occurs under some specific conditions which may (or may not) be observed. Bayesian statistics tries to define the probability of an event E, given some hypothesis H that describes the condition under which E may happen.
The following four axioms make for the building blocks of Bayesian statistics [10]:

$$P(E|H) \geq 0 \qquad \text{for all E,H} \tag{1}$$

$$P(H|H) = 1 \qquad \text{for all H} \tag{2}$$

$$P(E \cup F|H) = P(E|H) + P(F|H) \qquad \text{when E} \cap \text{F} \cap \text{H} = \emptyset \tag{3}$$

$$P(E|F \cap H) = \frac{P(E \cap F|H)}{P(F|H)} \tag{4}$$

Defining | as a conditional probability, we have that eq. 1 tells us the probability of event E happening under condition H. $P(E|H)$ need be equal or greater than zero, since an event may or may not happen. Equation 2 is the tautological condition: under hypothesis H, the probability of H happening must be 1. From a combination of eq. 1,2 we can demonstrate that the probability of P(E|H) is between zero and one. Equation 3 tells us that the probability of a specific set of events happening $(E \cup F)$, given condition H, will be the sum of those events happening independently under condition H, iff the intersection between each set is empty. In lay terms, eq. 3 tells us that the probability of a dicer to roll either a one or a two is equal to the probability of the dicer to roll one plus the probability of the dicer to roll two. Equation 4 relates the probability of E happening given the union between two condition F and H.

Given equations 1-4 we can obtain Bayes' theorem:

$$P(H|E) = \frac{P(E|H) \cdot P(H)}{P(E)} \tag{5}$$

which allows to compute the posterior distribution given information on the prior.

1. P(H) is defined as the prior probability, i.e. the estimate of the probability of hypothesis H before the data E, our evidence, is observed.
2. P(E|H) is the probability of observing E given H. It indicates the compatibility of the evidence with the given hypothesis. This term, as we will explore in sec. 2.4 is termed the likelihood.
3. P(E) is defined as the marginal likelihood or "model evidence".
4. P(H|E) is termed as the posterior probability, i.e. the probability of H given our observations E. This is what we want to know: the probability of a hypothesis given the observed evidence.

Bayes' theorem is usually proposed in the following manner

$$P(H|E) \propto P(H)P(E|H) \tag{6}$$

This happens because the constant of proportionality $\frac{1}{P(E)}$ is generally intractable. Nonetheless, no loss of generality is introduced by doing this simplification.

2.2 Application to modeling

Having introduced the fundamental principle at the basis of Bayesian statistic, we want to extend its role to the solution of real world problems. In almost all the situation we can think of, we are interested in estimating some k unknown quantities that make up our model M, where M is used to explain the data collected.
Our k unwknown parameters will be called the theta vector $\vec{\Theta}$

$$\Theta \ = \{\theta_1 ....\theta_k\}$$

Meanwhile, the set of observed data relevant to estimate these parameters will be the set

$$\mathbf{Y} = \{y_1 ...y_n\}$$

Since we assume that our model predicts the n-observed data, our unknown parameters θ must be related to our observations. Looking at our definition of Bayesian statistic, this dependence can be expressed as follows:

$$p(\Theta|\mathbf{Y}) \tag{7}$$

Equation 7 simply tells us that we can attach a distribution to the values of our parameters under the hypothesis of having observed the $\mathbf{Y}$ data. By applying Bayes' theorem we can re-write the previous expression in familiar terms

$$p(\Theta|\mathbf{Y}) \propto p(\Theta)p(\mathbf{Y}|\Theta) \tag{8}$$

We have now simplified our problem. To infer our posterior we simply need the prior distribution, i.e. our belief on what is the distribution of the parameters that need to be estimated, as well as being able to express the likelihood, i.e. the direct relationship for Y, fixed Θ.

2.3 The prior distribution

A fundamental aspect of Bayesian statistics, and one of its inherent weaknesses, is the influence of the prior distribution on our solution, as it is clearly evidenced by eq. 8 where the prior enters the posterior. The choice of the priors cannot be uniquely determined a priori, but are otherwise determined subjectively according to the beliefs of the practitioner [4]. Under certain conditions — shown further below — this inherent flaw of Bayesian statistics can be voided. Nonetheless, the prior plays an important role within Bayesian structure, it allows us to incorporate non-sample information (e.g. data from the existing literature) in a consistent manner. For example, in the context of econometrics, the prior provides a consistent mechanism to impose important information on certain parameters, such as positivity of volatility. It can impose stationarity, rule out near unit-root behavior, or separate mixture components [9].

When determining a prior distribution two key issues need to be taken into considerations

- what information is going into the prior distribution
- the properties of the resulting posterior distribution

With well-identified parameters and large sample sizes, prior distributions can be reasonably chosen, and will have minor effects on posterior inferences since the likelihood will have a predominance on forming the posterior distribution [1] [3]. If the sample size is small, or available data provide only indirect information about the parameters of interest, the prior distribution becomes more important.

We can broadly distinguish three classes of priors [3]:

1. Non-informative Prior Distributions
2. Moderately Informative Hierarchical Prior Distributions
3. Highly Informative Prior Distributions

Class 1 is used in situation where there is no reasonable or intuitive knowledge about the parameter that need to be evaluated. Usually they corresponds to wide, flat distribution. Class 3 priors are parameters that have a substantial literature with established values and boundaries. Class 2 parameters, are parameters whose values are not well known, but have an established relationship.

Given this classification, one could be enticed into using non-informative prior for all parameters of the model of interest, in order to obviate to Bayesian's flaw. We strongly admonish against this practice. As noted in the opening paragraph of this section, the prior is used to provide constraints on the values that a specific parameter may assume, given existing accepted knowledge. Failing to make use of this information may lead to absurd estimation. Gelman [3] in one such experiment of using only non-informative prior, evaluated the weight of the liver of his sample patients to be five to ten times those expected in literature.

2.4 The likelihood

In order to complete the definition of a Bayesian model, both the prior distributions and the likelihood must be approximated or fully specified. The likelihood, likelihood function, or $p(Y|\Theta)$ contains the available informations provided by the sample and is expressed as

$$p(\mathbf{Y}|\Theta) = \prod_{i=1}^{n} p(y_i|\Theta) \tag{9}$$

It allows us to estimate unknown parameters based on known outcomes. As a matter of fact, the likelihood could be thought of a reversed version of conditional probability.

Rewriting our problem by making use of eq. 9 we can highlight what mentioned in the previous section regarding the relative importance of our prior.

$$p(\Theta|\mathbf{Y}) \propto p(\Theta) \prod_{i=1}^{n} p(y_i|\Theta) \tag{10}$$

As our number of observations increases, the importance on the choice of the prior disappears, as it is vastly outweighed by the products of the likelihood of the observed data.

Note that because of the way we write Bayes' Theorem with a proportionality sign, it does not alter the result if we multiply the likelihood by any constant or indeed more generally by anything which is a function of Y alone.

[1] We refer the reader to http://faculty.washington.edu/kenrice/BayesIntroClassEpi515.pdf for a visualization of this assertion

2.5 The posterior distribution

Obtaining the posterior distribution in closed form is not a simple task, and more often than not is not possible. Nonetheless, there are situation where it is possible to identify closed form posterior, which allows to greatly simplify our problem. We report these cases hereafter as discussed in Tsay [13]

1. Suppose that $y_1...y_n$ form a random sample from a normal distribution with mean μ, which is unknown, and variance σ^2, which is known and positive. Suppose that the prior distribution of μ is a normal distribution with mean μ_0 and variance σ_0^2. Then the posterior distribution of μ given the data and prior is normal with mean and variance given by

$$\mu_* = \frac{\sigma^2\mu_0 + n\sigma_0^2\bar{Y}}{\sigma^2 + n\sigma_0^2} \qquad \sigma_{*^2} = \frac{\sigma^2\sigma_o^2}{\sigma^2 + n\sigma_0^2}$$

   - **Corollary**: For the multivariate situation

$$\mu_* = \Sigma_* \left(\Sigma_0^{-1}\mu_0 + n\Sigma^{-1}\bar{y}\right) \qquad \Sigma_{*^1} = \Sigma_0^{-1} + n\Sigma^{-1} \tag{11}$$

2. Suppose that $y_1...y_n$ form a random sample from a normal distribution with a mean μ and an unknown precision η. If the prior distribution of η is a gamma distribution with parameters $\alpha+\beta$, its posterior distribution is a gamma distribution with parameters $\alpha+n/2$ and $\beta+\sum_{i=1}^n(y_i-\mu)^2/2$

3. Suppose that $y_1...y_n$ form a random sample from a Bernoulli distribution with parameter θ. If the prior distribution of θ is a beta distribution given positive parameters α and β, then the posterior of θ is a beta distribution with parameters $\alpha + \sum_{i=1}^n y_i$ and $\beta + n - \sum_{i=1}^n y_i$

4. Suppose that $y_1...y_n$ form a random sample from a Poisson distribution with parameter λ. Suppose also that the prior distribution of λ is a gamma distribution with given positive parameters α and β. Then the posterior distribution of λ is a gamma distribution with parameters $\alpha + \sum_{i=1}^n y_i$ and $\beta + n$.

5. Suppose that $y_1...y_n$ form a random sample from an exponential distribution with parameter λ. If the prior distribution of λ is a gamma distribution with given positive parameters α and β, then the posterior distribution of λ is a gamma distribution with parameters $\alpha + n$ and $\beta + \sum_{i=1}^n y_i$.

6. Suppose that $y_1...y_n$ form a random sample from a negative binomial distribution with parameters m and λ, where m is positive and fixed. If the prior distribution of λ is a beta distribution with positive parameters α and β, then the posterior distribution of λ is a beta distribution with parameters $\alpha + mn$ and

$$\beta + \sum_{i=1} y_i$$

   .

7. Suppose that $y_1...y_n$ form a random sample from a normal distribution with an unknown mean μ and an unknown precision η. Suppose also that the conditional distribution of μ given $\eta = \eta_0$ is a normal distribution with mean μ_0 and precision $\tau_0\eta_0$ and the marginal distribution of η is a gamma distribution with positive parameters α and β. Then the conditional posterior distribution of μ given $\eta = \eta_0$ is a normal distribution with mean μ_* and precision η_*,

$$\mu_* = \frac{\tau_o\eta_0 + n\bar{y}}{\tau_0 + n} \qquad \eta_* = (\tau_0 + n)\eta_0$$

   where $\bar{y} = \sum_{i=1}^n y_i/n$ is the sample mean, and the marginal posterior distribution of η is a gamma distribution with parameters $\alpha + (n/2)$ and β_*, where

$$\beta_* = \beta + \frac{1}{2}\sum_{i=1}^n (y_i - \bar{y})^2 + \frac{\tau_0 n(\bar{y} - \mu_0)^2}{2(\tau_0 + n)}$$

8. Suppose that $a_1, ..., a_n$ form a random sample from a normal distribution with mean zero and variance σ^2. Suppose also that the prior distribution of σ^2 is an inverted chi-squared distribution with ν degrees of freedom: $(\nu\lambda)/\sigma^2 \sim \chi^2\nu$, where $\lambda > 0$. Then the posterior distribution of σ^2 is also an inverted chi-squared distribution with ν + n degrees of freedom — that is, $(\nu\lambda + \sum_{i=1}^{n} a_i^2)/\sigma^2 \sim \chi^2_{\nu+n}$.

As shown by this set of conjugate prior, computing the closed form posterior is not straightforward and rather cumbersome. Markov Chain Monte Carlo Simulation techniques is what is used to obviate these problem. The application of MCMC allows the practitioner to solve the posterior problem without knowledge of the closed form solution.

2.6 Extension to unobservable state variables

In case of unobservable state variables we can still apply Bayes' theorem without any loss of generality.

Calling

$$\mathbf{X} = \{x_1 ... x_t\}$$

our latent variables, we can rewrite our original problem

$$p(\Theta|\mathbf{Y})$$

as

$$p(\Theta, \mathbf{X}|\mathbf{Y}) \tag{12}$$

Sequentially applying Bayes' theorem to solve the problem expressed by eq. 12 we obtain the following formulation

$$p(\Theta, \mathbf{X}|\mathbf{Y}) \propto p(\mathbf{Y}|\mathbf{X}, \Theta)p(\mathbf{X}|\Theta)p(\Theta) \tag{13}$$

In this situation we have that $p(\mathbf{Y}|\mathbf{X}, \Theta)$ is our likelihood function, $p(\mathbf{X}|\Theta)$ is the distribution of the state variable, and $p(\Theta)$ is once again our prior

3 Monte Carlo Markov Chains

MCMC classifications, theoretical foundations, and convergence are discussed in this section.

3.1 Clifford Hammersley Theorem

In most asset pricing models, $p(\Theta, X|Y)$ is a high- dimensional distribution and it is unlikely to sample from it directly. Nevertheless, the Clifford-Hammersley theorem tackles this problem by characterizing the joint posterior into its complete set of conditional posteriors.

The simplest example is that $p(\Theta, X|Y)$ can be expressed by $p(\Theta|X, Y)$ and $p(X|\Theta, Y)$. Furthermore, if the dimensionality of two conditional posteriors are still high, the Clifford-Hammersley theorem suggests that they can be further partitioned into lower dimension components. Consider a M-dimensional X, the following equations t

$$p(X_1|X_2, X_3 ... X_m\Theta, Y) \tag{14}$$

$$p(X_2|X_1, X_3 ... X_m\Theta, Y) \tag{15}$$

$$\vdots$$

$$p(X_m|X_1, X_2 ... X_{m-1}\Theta, Y) \tag{16}$$

determine $p(X|\Theta, Y)$. In general, the information of M one-dimensional conditionals is equivalent to the knowledge of a M-dimensional posterior.

3.2 Algorithms

Suppose we have a method to generate i.i.d. copies of a random variable X. If we generate many copies this way, we can use the frequency with which different values are observed to approximate the distribution of X. These approximations work because of the law of large numbers, and accuracy of the approximations can be evaluated using the central limit theorem. This is known as Monte carlo estimation of a distribution. It is useful when we have a way to simulate the distribution, but no way to do calculations with the distribution in closed form.

For Bayesian statistics, the distribution we want to understand can be enormously complex — making even simulation of i.i.d. copies from the distribution intractable.MCMC refers to an algorithmic method for generating these samples.
The basic idea is as follows:

1. Suppose we want to sample from a distribution π.
2. We create a Markov chain that is irreducible and aperiodic has π as its invariant distribution.
3. Now we know that if we run the chain long enough, by the convergence theorem, the distribution of the state will be approximately π. Further, by applying the Markov LLN, we can compute statistics from the distribution π using the sample path of states observed.

Thus MCMC uses Markov chains entirely as a way to efficiently generate samples from very complicated distributions. The most common MCMC algorithms are discussed in following sections.

3.2.1 Gibbs sampler

The Gibbs sampler requires the complete conditionals of the target distribution to be sampled iteratively. When drawing from the full-conditional distributions is not straightforward other samplers-within-Gibbs are used. Gibbs sampling is popular as it is the simplest MCMC algorithm. The full procedure of Gibbs sampling is listed below:

1. Pick any initial state $X_0 = \theta_0$.
2. Given state $X_n = \theta_n$ at time step n, pick a coordinate m uniformly at random from 1, . . . , M, and then we randomly pick a coordinate m, and resample θ_m from the full conditional distribution, given θ_{-m}

In most asset pricing models, the implementation of Gibbs sampler requires sampling from standard continuous time distributions.

The Gibbs sampler is popular for sampling from complex, multivariate probability distributions. However, the Gibbs sampler cannot be used for general sampling problems. For many target distributions, it may be difficult to obtain a closed-form expression for all the needed conditional distributions. In other scenarios, analytic expressions may exist for all conditionals but it may be difficult to sample from any or all of the conditional distributions. Gibbs samplers are commonly used for Bayesian methods where models are often devised in such a way that conditional expressions for all model variables are easily obtained and take well-known forms that can be sampled from efficiently.

3.2.2 Griddy Gibbs

When non-linear parameters are present in financial models, the Gibbs Sampler cannot be implemented directly since there are no closed-form expressions for posteriors. The Griddy Gibbs solves this problem by approximating the continuous θ with discrete points. The Griddy Gibbs flows as follows:

1. select N points from θ
2. calculate ω_j=$p(\theta_{ij}|X, \theta-i)$ for j=1,2.....n

3. Get an approximation of the inverse CDF of $p(\theta_i|X, \theta{-}i)$ by using ω_1,...ω_n

4. Draw a uniform on (0,1) and transform the inverse CDF to get a sample for θ_i

The proper choose of set of points significantly increase the accuracy of the approximation of the target distribution. However, this algorithm works relatively poor when the dimensionality is high.

3.2.3 Metropolis-Hastings

This algorithm applies when some parameters cannot be directly or efficiently sampled. The algorithm takes the following steps:

1. pick an initial state θ_i at random

2. find a candidate density distribution q

3. randomly pick a state θ_{i+1} according to the candidate density q(θ_{i+1} | θ_i)

4. update the state with probability min($\frac{\theta_{i+1}q(\theta_i|\theta_{i+1})}{\theta_i q(\theta_{i+1}|\theta_i)}$,1)

It should be noticed that the selection of the candidate density depends on other parameters. In addition, it is recommended to check the acceptance rate in practice.

Although the Gibbs sampler and Metropolis-Hastings algorithms can be thought of as separate instruments for posterior simulation, this distinction is rather artificial. In fact, the Gibbs sampler can be regarded as a special case of the Metropolis-Hastings algorithm, where the proposal density consists of the set of conditional distributions, and jumps along the conditionals are accepted with probability one.

3.3 Convergence of MCMC methods

Since Gibbs Sampler is a special case of Metropolis-Hastings, we can concentrate on checking the convergence of Metropolis-Hastings. This can be easily done by verified by examining the time-reversibility of the Metropolis-Hastings. The detailed balance is satisfied when there exists a time invariant distribution π such that

$$P(x, y)\pi_x = P(y, x)\pi_y \tag{17}$$

where P is the transition function.

The transition function of Metropolis-Hastings is

$$P(x, y) = \alpha(y, x)Q(x, y) + (1 - r(x))\delta_x(y) \tag{18}$$

where r(x)=$\int \alpha(y, x)Q(x, y)dy$ and Q(x,y)=q(y|x) The proof of detailed balance for the first term is given as follow:

$$\alpha(x, y)Q(x, y)\pi(x) = min(\frac{\pi(y)Q(y, x)}{\pi(x)Q(x, y)}, 1)Q(x, y)\pi(x) \tag{19}$$

$$= min(1, \frac{\pi(x)Q(x, y)}{\pi(y)Q(y, x)})Q(y, x)\pi(y) \tag{20}$$

$$= \alpha(y, x)Q(y, x)\pi(y) \tag{21}$$

The proof of detailed balance for the second term is similar. Therefore, the Metropolis-Hastings satisfies the detailed balance and the Markov Chain is reducible. Therefore, the algorithm converges.

4 Stochastic Volatility Models

In the world of asset pricing, especially option pricing, we conduct most of our estimation via parameterizations of volatility. There are currently three mainstream methods to go by: constant volatility (which basically assumes we can observe volatility directly), GARCH and stochastic volatility. The first one is of little interest for further research and mainly has been used as a benchmark, the latter are instead the industry standard. In this section we justify our usage of stochastic volatility models in depicting S&P500's returns, and explain how MCMC plays a centric role in making estimations of stochastic volatility models.

4.1 Limitations of Vanilla GARCH

Prior to the introduction of diffusive volatility models, GARCH(1,1) had been the most common and robust method used to fit a process that has "volatility clustering"[7].
Traditionally, the most basic GARCH model, a GARCH(1,1), allows for an exogenous variable and is formulated as follows:

$$\begin{cases} r_t = \mu_0 + \mu_1 x_{t1} + \epsilon_t \\ \epsilon_t = \sigma_t e_t \\ \sigma_t^2 = \omega + \alpha \epsilon_{t-1}^2 + \beta \sigma_{t-1}^2 \\ e_t \overset{iid}{\sim} \mathcal{N}(0,1) \end{cases} \tag{22}$$

As is supported by multiple empirical researches such as [11], GARCH(1,1) outperforms other models, including volatility diffusion models, in pricing of both fixed income and derivative financial assets.

However, this is essentially because of an implied trade-off, in the sense that GARCH assumes the randomness in variances/volatilities and the randomness in returns come from the same term e_t's. Therefore, in GARCH models, the volatilities are implicitly helping in explaining r_t with said assumptions, and over time such assumptions would lead to over-fitting problems. Empirically, this could be interpreted as GARCH not being able to predict well. Specifically, if we were to project risk management quantities such as VaR or CVaR, in theory we can foresee that GARCH would not be ideal compared with models that allow for idiosyncratic dynamics within volatility movements. This has also been proved in the empirical works we referred to.

4.2 Introduction of Stochastic Volatility Models

Intuitively, it is fully understandable if one were interested in considering the movements of volatility as an independent market measure. In Black-Scholes world, such measurement is implied and can only be backed out by observed derivative prices, interest rates and other set contract-specific parameters. But said descriptions fail to be self-consistent, proved by the well-known "volatility smiles". Given that GARCH has already offered interoperability for conditional heteroskedasticity, we need even stronger models that focus on jointly decomposing dynamics of returns and volatilities.

4.2.1 Diffusive Volatility Models

Using a Bayesian framework, Jacquier and Rossi [8] have proposed a model commonly referred to as stochastic volatility model, and is in fact an AR(1) model for the conditional variance of the return series. Namely,

$$\begin{cases} y_t = \sqrt{h_t} u_t \\ \ln h_t = \alpha + \delta \ln h_{t-1} + \sigma_v v_t \\ \mathbf{Z}_t = [u_t, v_t]^T \overset{iid}{\sim} \mathcal{N}(0, \mathbf{I}_2) \end{cases} \tag{23}$$

where y_t's are daily returns of certain assets (i.e. not limited to equities), h_t's are implied volatilities, but different from those live in Black-Scholes' model, and $\mathbf{Z}_t$'s are two-dimensional jointly Gaussian random vector with an identity matrix as covariance.

Without empirical work, we can notice from (23)'s model formulation several important properties:

- It allows for idiosyncratic movements in y_t's and h_t's respectively, through the $\mathbf{Z}_t$'s. But they can still be fitted to correlation, since the scale of $\sigma_v v_t$ as one entity can be adjusted through the coefficient σ_v;
- This model can be extended rather conveniently even if we do not consider jumps for now. On two major fronts:
  - we can consider adding in exogenous covariates in the mean regression so that it becomes:

$$y_t = \mathbf{X}_t\boldsymbol{\beta} + \sqrt{h_t}u_t \tag{24}$$

    where similar to OLS, we can impose a column vector purely made of 1 in $\mathbf{X}_t$'s so that it allows for intercepts;
  - we can modify the autoregressive lag in the variance regression to be arbitrarily AR(p), or similar to the common practice in extending ARCH to GARCH, we can add in $\sigma_v v_{t-1}$ so that it becomes an ARMA(1,1) regression for the variances:

$$\ln h_t = \alpha + \delta \ln h_{t-1} + \sigma_v^{(1)} v_t + \sigma_v^{(2)} v_{t-1} \tag{25}$$

  More generally, if we were to ignore the jumps part, in [2] the authors have proposed a more compact way of formulating the diffusion volatility models that allow for correlation terms between the returns' regression and the volatilities' regression:

$$\begin{bmatrix} \mathrm{d}Y_t \\ \mathrm{d}V_t \end{bmatrix} = \begin{bmatrix} \mu \\ \kappa(\theta - V_t) \end{bmatrix} \mathrm{d}t + \sqrt{V_t} \begin{bmatrix} 1 & 0 \\ \rho\sigma_v & \sqrt{1-\rho^2}\sigma_v \end{bmatrix} \mathrm{d}W_t \tag{26}$$

4.2.2 SV Models with Jumps

Taking a step back, we must also notice that multiple works, such as the Merton jump diffusion model built by Merton himself[12], have been published to improve the B-S-M's measurement of heteroskedasticity using jumps, namely formed in the following SDE:

$$\frac{\mathrm{d}S}{S} = (\alpha - \lambda k)\mathrm{d}t + \sigma \mathrm{d}Z + \mathrm{d}q \tag{27}$$

where:

- S denotes an asset/portfolio's price;
- α could be interpreted as the expected return of said asset/portfolio over a very small time window, σ^2 is the volatility of said asset/portfolio over a very small time window. These two can be treated as expectation of each own represented statistic, conditional on no q events have occurred in this small time window;
- Z is a standard Brownian motion;
- $q(t)$ is a Poisson process with rate λ satisfying:
  - $\mathbb{P}$[The event does not occur in time interval $(t, t+h)$]$=1-\lambda h + o(h)$;
  - $\mathbb{P}$[The event occurs once in time interval $(t, t+h)$]$=\lambda h + o(h)$;
  - $\mathbb{P}$[The event occurs more than once in time interval $(t, t+h)$]$=o(h)$;

This inclusion of jumps in the dq part allows for abnormal price vibrations, but said effects are transient, because even if the arrival of a Poisson event is taking place, its impact on S would be over a small time horizon. This is similar to the AR vs. ARMA model case in the sense that we want the process to remember a shock, which naturally leads us to inclusion of jump terms in the variance regression in our stochastic volatility model in (26):

$$\begin{bmatrix} \mathrm{d}Y_t \\ \mathrm{d}V_t \end{bmatrix} = \begin{bmatrix} \mu \\ \kappa(\theta - V_t) \end{bmatrix} \mathrm{d}t + \sqrt{V_t} \begin{bmatrix} 1 & 0 \\ \rho\sigma_v & \sqrt{1-\rho^2}\sigma_v \end{bmatrix} \mathrm{d}W_t + \begin{bmatrix} \xi^y \mathrm{d}P_t^y \\ \xi^v \mathrm{d}P_t^v \end{bmatrix} \tag{28}$$

where $(P_t^y : t \geq 0)$ and $(P_t^v : t \geq 0)$ are Poisson processes depicting the arrivals of extreme events to return process Y_t's and volatility process V_t's.

One fascinating facet of this model formulation is that we can arbitrarily impose very complex distributions on the $(V_t : t \geq 0)$, $(P_t^y : t \geq 0)$ and $(P_t^v : t \geq 0)$. For instance, if we wish to argue that the jumps in volatilities partially cause the jumps in returns, or in math's sense: conditional on the jump processes in returns as priors, we can easily confine the statistics of returns jump processes using the correlation term ρ in (28), such as:

$$\text{IF} \quad \xi^v \sim \exp(\mu_v), \quad \text{THEN} \quad \xi^y | \xi^v \sim \mathcal{N}(\mu_y + \rho\xi^v, \sigma_y^2) \tag{29}$$

4.3 Estimations of SV Models Using MCMC

Even though stochastic volatility models present itself with so many stated mathematical advantages, in reality they are hard objects to estimate because the typical methods for estimating complex system coefficients, such as Maximum Likelihood, requires a closed form expression for every parameter in question. However, with or without the jump terms, in stochastic volatility models, even if we can estimate everything in the variance regression by pretending we are estimating an ARMA(1,1), we still do not have an explicit formulation for Y_t's given it is specified with an $\sqrt{V_t}$ term in our model.

4.3.1 General Approach of Estimating SV

A common approach is to treat stochastic volatility model similar with what we did in *Parameter Estimation of SDEs* (Assignment 1, Question 6 of MS&E349, Spring 2017), which is using a discretized approximation to iteratively break down the model, namely the following equation if we omit jumps:

$$\begin{cases} Y_{t+h} = \beta_0 + \beta_1 Y_t + \sqrt{hV_t} \cdot \epsilon_t^y \\ V_{t+h} = \alpha_0 + \alpha_1 V_t + \sigma_v \sqrt{hV_t} \cdot \epsilon_t^v \\ \mathbf{Z}_t = [\epsilon_t^y, \epsilon_t^v]^T \sim \mathcal{N}(0, \mathbf{I}_2) \end{cases} \tag{30}$$

In practice, we can treat h as one day if we are looking at daily return dynamics over a decade, which gives us readily available Y_{t+h} and Y_h and does not require us to simulate the paths between Y_{t+1} and Y_t (in other words, Y_{t+1} is regarded as Y_{t+h} since one day is relatively small time horizon).

4.3.2 Where MCMC Comes Into Play

This paragraph discusses the advantages and necessities of using the MCMC approach in estimating the diffusion volatility model family, with or without the jump terms:

i. We can estimate the latent variables with a very flexible constraint on model setup: Notice that in (30), we have used as many V_t's as Y_t's, but these V_t's are artificially objects that we build to describe volatilities. In other words, one could think of (30) as having not merely 5-dimensional parameter vector $\Theta = [\beta_0, \beta_1, \alpha_0, \alpha_1, \sigma_v]^T$, but rather on top of that, we have an additional T-dimensional parameter vector $\mathbf{V} = [V_1, \cdots, V_T]^T$. This cause a major difference from GARCH, because we have no closed form expressions for the V_t's distributions, even if we condition them on V_{t-1}, because they are not observed but rather implicitly fitted coefficients by backing out from $Y_{t+h} = \beta_0 + \beta_1 Y_t + \sqrt{hV_t} \cdot \epsilon_t^y$;

ii. We can allow very loosed priors in MCMC so that the model specification errors are minimized in our estimation process: As discussed in section 3 of this working paper, we know MCMC will converge towards an asymptotically close shape of the true parameters, as long as we state our priors to be very general. In practice, for instance, there is no trivial solution as for what priors we should impose on σ_v term in (30), but we can let it be a very uninformative χ^2 distribution and run the simulations long enough to avoid the complex mathematical problem;

iii. MCMC are computationally efficient in the sense we have too many latent variables in our model: As pointed out in [i] of this paragraph, we need to treat $\mathbf{V}$ as a T-dimensional parameter vector and in reality T could be over 2500. Thus, it would be unlikely if we were to update these parameters using traditional ML approach assuming we have imposed a strong prior model specification where their likelihood functions can be calculated. Using MCMC's Metropolis scheme, however, updating the entire $\mathbf{V}$ vector becomes acceptable because we can leverage the Markovian property to consider only $\mathbf{Y}, \Theta, V_{t-1}$ and V_{t+1} when updating V_t.

Due to the great flexibility of MCMC, we can in fact use both branches of the MCMC methods in our estimation. More specifically, since our Θ have closed-form priors that fall under Gibbs sampler's solvable distribution classes, we can use Gibbs sampler for those. On the other hand, we can use Metropolis' algorithm for calculating our posteriors and making draws for $\mathbf{V}$. In very general flow diagram, the procedures are:

Algorithm 1: Mixture of Gibbs Sampler & Metropolis-Hastings for Estimating SV (30)

Input : Observations $\mathbf{Y} \in \mathbb{R}^T$
Output: $\hat{\Theta} = [\hat{\beta}_0, \hat{\beta}_1, \hat{\alpha}_0, \hat{\alpha}_1, \hat{\sigma}_v]^T$ and $\hat{\mathbf{V}} \in \mathbb{R}^T$
Initialize $\hat{\Theta}$ and $\hat{\mathbf{V}}$ according to their priors;
while $r < nRoundAllow$ **do**
  Use Gibbs sampler's Gaussian formula to calculate posterior of $[\beta_0, \beta_1]^T$, conditional on $\{\mathbf{Y}, [\hat{\alpha}_0^{(r)}, \hat{\alpha}_1^{(r)}]^T, \hat{\sigma}_v^{(r)}$ and $\hat{\mathbf{V}}^{(r)}\}$. Then, make a draw from it to be $[\hat{\beta}_0^{(r+1)}, \hat{\beta}_1^{(r+1)}]^T$;
  Use Gibbs sampler's Gaussian formula to calculate posterior of $[\alpha_0, \alpha_1]^T$, conditional on $\{\mathbf{Y}, [\hat{\beta}_0^{(r+1)}, \hat{\beta}_1^{(r+1)}]^T, \hat{\sigma}_v^{(r)}$ and $\hat{\mathbf{V}}^{(r)}\}$. Then, make a draw from it to be $[\hat{\alpha}_0^{(r+1)}, \hat{\alpha}_1^{(r+1)}]^T$;
  Use Gibbs sampler's χ^2 formula to calculate posterior of σ_v^2, conditional on $\{\mathbf{Y}, [\hat{\beta}_0^{(r+1)}, \hat{\beta}_1^{(r+1)}, \hat{\alpha}_0^{(r+1)}, \hat{\alpha}_1^{(r+1)}]^T$ and $\hat{\mathbf{V}}^{(r)}\}$. Then, make a draw from it to be $\hat{\sigma}_v^{(r+1)}$;
  Use Metropolis-Hastings to sequentially update $\hat{\mathbf{V}}_t^{(r)}$ based on $[\hat{\beta}_0^{(r+1)}, \hat{\beta}_1^{(r+1)}, \hat{\alpha}_0^{(r+1)}, \hat{\alpha}_1^{(r+1)}]^T$, $\hat{\mathbf{V}}_{t-1}^{(r+1)}$ and $\hat{\mathbf{V}}_{t+1}^{(r)}\}$. Specifically,
    - Step 1: Calculate posterior using $[\hat{\beta}_0^{(r+1)}, \hat{\beta}_1^{(r+1)}, \hat{\alpha}_0^{(r+1)}, \hat{\alpha}_1^{(r+1)}]^T$, $\hat{\mathbf{V}}_{t-1}^{(r+1)}$ and $\hat{\mathbf{V}}_{t+1}^{(r)}\}$. Make a draw such that we have likelihood without normalization $\pi(r+1)$ and cdf $q(r+1)$ within its posterior distribution;
    - Step 2: Accept the draw with probability η, where

$$\eta = \min\{1, \frac{\pi(r+1)/q(r+1)}{\pi(r)/q(r)}\} \tag{31}$$

  if *converges (e.g. updates < 1e-2)* **then**
    Exit the while loop before we reach $nRoundAllow$;
  else
    $r+ = 1$ and continue;
  end
end

where superscript r denotes the round we are at right now and subscript t denotes the corresponding time index into our implied volatility vector $\mathbf{V}$.

5 Empirical MCMC Estimation of Stochastic Volatility Model

5.1 Data Preparations

To prepare our data, we first obtained the Volume-Weighted Daily returns of the S&P 500 index from the Wharton Research Data Services database. This data set ranged from 01/03/2003 (January 3^{rd}) to 12/30/2016 (December 30^{th}). This data set was particularly valuable, since it reflected the market through the S&P 500 index and accounted for adjusted returns.

The second data set we used was the Lag one term 10 year treasury bill rates. Though this dataset ranged from 01/02/1990 (January 2^{nd}, 1990) to 12/30/2016 (December 30^{th}, 2016), we used only the dates that overlapped with our S&P 500 index in our data set. We chose this data set in particular, because it represents a good choice for an exogenous variable – one that is wholly causally independent from the other variables in the system. This allows us to set our arbitrary external conditions, as this is important to setting up the empirical MCMC model.

In line with convention, we also split the data into two sets, a training and a test set. To do so, we split data at the years 2013 (i.e. all data before 2013 is part of the training set and all data after that is part of the test set). This will allow us to calculate residuals and evaluate the efficacy of our MCMC estimated parameters.

5.2 Formulations: Priors of the Model

We now look at the prior distribution for our models, before examining our empirical implementation of them. Since MCMC allows us to model the mean and volatility equations of an asset return r_t as follows:

$$r_t = \beta_0 + \beta_1 x_{1t} + ... + \beta_p x_{pt} + a_t, a_t = \sqrt{h_t}\epsilon_t \tag{32}$$

$$\ln(h_t) = \alpha_0 + \alpha_1 \ln(h_{t-1}) + v_t \tag{33}$$

Specifically,

- $\{x_{it} | i = 1, ..., p\}$ are the external explanatory variables available at time $t-1$. In matrix form, we will in the following descriptions refer to the random processes as $\mathbf{X} \in \mathbb{R}^{T \times p}$;
- β_j's are coefficients for $\mathbf{X}$, whose stacked column vector will be denoted as $\boldsymbol{\beta}$ in the following descriptions. Similarly, we have $\boldsymbol{\alpha} = [\alpha_0, \alpha_1]^T$;
- ϵ_t's are i.i.d. $\mathcal{N}(0, 1)$, where $\mathcal{N}(0, 1)$ is a Gaussian random variable with mean 0 and variance 1. v_t's are also i.i.d. Gaussian, but follow $\mathcal{N}(0, \sigma_v^2)$. We assume v_t's and ϵ_t's to be independent;

Moreover, for the sake of implementation convenience, We log transform h_t to ensure it is positive for all values of t. Note that equation (33) from above is based off of a log transformed AR(1) model, so we assume $|\alpha| < 1$ to ensure that the log volatility process, $ln(h_t)$, is stationary. This gives us our overall model for the MCMC implementation that follows [13].

5.2.1 Beta Priors

For our $\boldsymbol{\beta}$ priors, we note that the coefficient vector of the mean equation (32) can be denoted by:

$$\boldsymbol{\beta} = (\beta_0, \beta_1, \beta_2, \beta_3, ..., \beta_p)^T$$

We now assume that the prior distribution of $\boldsymbol{\beta}$ is multivariate normal with mean β_0 and covariance matrix A_0. For our purposes, we generated a 1x2 matrix with two floating point numbers between [0,1) drawn from a uniform distribution. This was the mean for the priors of $\boldsymbol{\beta}$. This mean was a 1x2 matrix with the first index being the mean for β_0 and the second index being the mean for β_1 We generated a 2x2 covariance matrix with floating point numbers drawn from the same distribution as the mean. This yielded our covariance for $\boldsymbol{\beta}$. Thus, our value for the prior of $\boldsymbol{\beta}$ was drawn from a multivariate normal distribution with this mean and covariance [13].

5.2.2 Alpha Priors

For our $\boldsymbol{\alpha}$ priors, we note the coefficient vector of the variance equation (33) is given above. We follow a similar process as we did with $\boldsymbol{\beta}$ to generate our $\boldsymbol{\alpha}$ priors. Thus, they follow a multivariate normal distribution with the mean equal to a 1x2 matrix with two floating point numbers between [0,1) drawn from a uniform distribution. The covariance matrix of $\boldsymbol{\alpha}$ was a 2x2 covariance matrix with floating point numbers drawn from the same distribution as the mean. From the multivariate normal distribution we drew our values for α_0 and α_1 [13]

5.2.3 Sigma Priors

To generate σ_v^2 priors, we used a slightly different strategy. As given in Tsay 2002, the prior distribution of σ_v^2 is given as follows:

$$(m\lambda)/\sigma_v^2 \sim \chi_m^2$$
$$(m\lambda)/\chi_m^2 \sim \sigma_v^2$$

We generated m by choosing an random integer between [1,10]. We generated λ by drawing from a standard normal distribution. The number of degrees of freedom for our χ_m^2 distribution was given by the length of our Training Set + m - 1. This yielded a single floating point number for the value our prior on σ_v^2 [13].

5.2.4 Volatility Priors

We generated a prior for our ϵ_t by drawing |len(TrainingSet)| samples from $\mathcal{N}(0,1)$.
From Tsay 2002, we let:

$$\boldsymbol{H} = (h_1, h_2, ..., h_n)^T$$

be the vector of unobservable volatilities as given by equation (33). Thus, our prior on $\boldsymbol{H}$ equals the following:

$$h_t = [(r_t - \beta_0 - \beta_1 \times x_{1t})/\epsilon_t]^2$$

We then set any $h_t = 0$ values equal to 0.00001, as we need to calculate the logs of h_t, so we must avoid zeros, which concludes our discussion of the necessary priors for the variables of interest in our MCMC [13].

5.3 Estimations: Posteriors using Gibbs and Metropolis

5.3.1 Alpha Posteriors

Since $\boldsymbol{H}$ follows AR(1), we can borrow from established conclusions that if the priors for $\boldsymbol{\alpha}$ is jointly Gaussian $\mathcal{N}(\boldsymbol{\alpha}_0, \boldsymbol{C}_0)$, then the posteriors are:

$$\begin{cases} \boldsymbol{C}_*^{-1} = \frac{1}{\sigma_v^2}\mathbf{Z}\mathbf{Z}^T + \boldsymbol{C}_0^{-1} \\ \boldsymbol{\alpha}_* = \frac{1}{\sigma_v^2}\boldsymbol{C}_*\mathbf{Z}(\ln \mathbf{H})^T \end{cases} \tag{34}$$

5.3.2 Beta Posteriors

Since the returns' regression is specified by:

$$\frac{r_t}{\sqrt{h_t}} = \frac{1}{\sqrt{h_t}}\mathbf{X}_t\boldsymbol{\beta} + \epsilon_t \tag{35}$$

Which is in a similar form of OLS, but with linear transformations via $\boldsymbol{H}$, we can take advantage of the Gaussian posterior results to arrive at:

$$\begin{cases} \boldsymbol{A}_*^{-1} = \frac{1}{h_t}\mathbf{X}_t\mathbf{X}_t^T + \boldsymbol{A}_0^{-1} \\ \boldsymbol{\beta}_* = \frac{1}{h_t}\boldsymbol{A}_*(\mathbf{X}_t r_t + A_0^{-1}\boldsymbol{\beta}_0) \end{cases} \tag{36}$$

where prior is $\mathcal{N}(\boldsymbol{\beta}_0, \boldsymbol{A}_0)$ and posterior is $\mathcal{N}(\boldsymbol{\beta}_*, \boldsymbol{A}_*)$;

5.3.3 Sigma Posteriors

Given $\boldsymbol{H}$ and $\boldsymbol{\alpha}$, we can back out an intermediate random variable v_t such that:

$$v_t = \ln h_t - \alpha_0 - \alpha_1 \ln h_{t-1} \tag{37}$$

Where σ_v^2 is the variation for v_t. So if the prior for σ_v^2 is an inversed χ^2 such that:

$$\frac{k\lambda}{\sigma_v^2} \sim \chi_m^2 \tag{38}$$

Then its posterior, given the new v_t's should be:

$$\frac{k\lambda + \boldsymbol{v}^T\boldsymbol{v}}{\sigma_v^2} \sim \chi_{T+m-1}^2 \tag{39}$$

5.3.4 Volatility Posteriors

As stated in 4.3.2, we update $\boldsymbol{H}$ element-wise, and since they naturally form a sequence, we can use the newly updated posteriors with the smaller indices as priors for those with larger indices.

Specifically, using Bayes' rule, we have:

$$\begin{aligned} f(h_t|\boldsymbol{R}, \boldsymbol{X}, \boldsymbol{\beta}, \boldsymbol{H}_{-t}, \boldsymbol{\omega}) &\propto f(\boldsymbol{\alpha}|h_t, r_t, x_t, \boldsymbol{\beta}) \cdot f(h_t|h_{t-1}, \boldsymbol{\omega}) \cdot f(h_{t+1|h_t, \boldsymbol{\omega}}) \\ &\propto h_t^{-1.5} \exp\{-\frac{(r_t - \mathbf{X}_t\boldsymbol{\beta}^T)^2}{2h_t} - \frac{(\ln h_t - \mu_t)^2}{2\sigma^2}\} \end{aligned} \tag{40}$$

with

$$\begin{cases} \mu_t = \frac{1}{1+\alpha_1^2}\alpha_0(1-\alpha_1) + \alpha_1(\ln h_{t+1} + \ln h_{t-1}) \\ \sigma^2 = \frac{1}{1+\alpha_1^2}\sigma_v^2 \end{cases} \tag{41}$$

By leveraging Lemma 1 on page 418 of [1]. Notice the proportional likelihoods are not likelihood because numerically it would be very hard to calculate the normalization term. So, as proposed in 4.3.2, we plug in (40) as the $\pi(t)$ terms to perform a Metropolis-Hastings accept/reject draw, rather than straight updating the h_t's.

5.4 Predictions and Evaluations

To evaluate the efficacy of our model, we decided to explore various test statistics on our in-sample and out-of-sample prediction. Since we split our data set into a training set (January 2003 - December 2012) and a test set (January 2013 - December 2016), we decided to evaluate the in-sample and out-of-sample predictions separately.

5.4.1 In-Sample Metrics

To evaluate our in-sample predictions, we decided to use Mean-Squared Error and R^2 to evaluate the efficacy of our model.

- Mean-Squared Error is defined as follows: $MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$.
  Our Mean-Squared Error was calculated in this way and yielded a value of MSE = 0.425589945137
- R^2 was defined as follows: $R^2 = 1 - \frac{SSE}{SST}$, where:
  $SSE = \sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$ and $SST = \sum_{i=1}^{n}(Y_i - \bar{Y})^2$ Unsurprisingly, our model yields a perfect fit if we let it run long enough, so R^2 is an indicator for us to meddle the simulation so that it does not fit.

5.4.2 Parameter Estimates

Ultimately, the goal of our MCMC on the S&P 500 data was to come up with parameter estimates for $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, σ_v^2, and $\boldsymbol{H}$, each of which was defined with priors in Section 5.2 and with Posteriors in Section 5.3. From our empirical MCMC estimation we observed the following results:

- $\boldsymbol{\alpha} : \alpha_0 = 0.53604001, \alpha_1 = 0.55985555$
- $\boldsymbol{\beta} : \beta_0 = 0.02734105, \beta_1 = -0.01074269$
- $\sigma_v^2 = 3.3414786126692713$

These parameter estimations were novel and allowed us to make predictions on otherwise unobservable parameters on the S&P 500 index, using MCMC. We see the model in equation (32) and equation (33) seek to describe the mean and volatility of asset return r_t, but the four parameters $\boldsymbol{\alpha}, \boldsymbol{\beta}, \sigma_v^2, and \boldsymbol{H}$ are unobservable in the data, while MCMC allows us to make a somewhat accurate prediction of them. This is where the real power of MCMC is visible – making predictions for unknown parameters and variables, while using simulation on a data set. This concludes our empirical discussion of our MCMC implementation and the results it yielded.

5.5 Extendability of the Model

It is worth pointing out that our model covers a matrix version of (32) such that even though we have only accounted for one covariate x_t's term on the right hand side of the returns' regression, it would be straightforward to expand such part into arbitrarily high-dimensional $\mathbf{X}$ matrix of exogenous covariates.

Also, one would and should always be concerned over the priors a Bayesian method choose. Whether a method is robust or not should not depend on our priors choice, especially not on our priors' distribution parameters choice. Even though as argues in Tsay's book, one can potentially save time by plugging the fitted coefficients from GARCH(1,1) as prior to our diffusion volatility models, we think it would be more reasonable if we use randomly generated priors with the distribution family specified – any more detailed version of a prior would lead to the result being incomparable to that of other models. On the implementation level, such practice would not be hard because we have isolated and exposed our prior specifications in the code and we can conduct robustness test by running the entire MCMC simulations with different random seeds.

Due to a limited time frame, we have not covered the implementation of estimating a stochastic volatility model with jumps, but since we have established a framework for a rather complex version of MCMC, i.e. jointly usage of Gibbs sampler and Metropolis-Hastings algorithm, we believe such extension of the model would not be hard.

A Python Implementation of Our Empirical Estimation

```
# -*- coding: utf-8 -*-
"""
Created on Fri May 26 02:55:27 2017
@author: Jiacheng Z
"""

from __future__ import print_function

import re
import argparse
import pandas as pd
import numpy as np
import numpy.random as rand
import matplotlib.pyplot as plt
import statsmodels.api as sm
from scipy.stats import gamma
from numpy.linalg import inv as invert
from datetime import datetime


def NOW():
    return str(datetime.now())[:-7]


def NOWDIGIT():
    return re.sub(pattern='[-: ]*', repl="", string=NOW())


def Standardize(dfSerie):
    STD = dfSerie.std()
    MEAN = dfSerie.mean()
    return (dfSerie - MEAN) / STD


class ReadData:
    def __init__(self, SplitYear=2013):
        file_loc = r"https://raw.githubusercontent.com/jiacheng0409/mcmc_sv/
            master/sp_daily.csv"
        rwData = pd.read_csv(file_loc)
        rwData['vwretd'] = Standardize(rwData['vwretd'])
        rwData['tbill'] = Standardize(rwData['tbill'])

        # the following three lines make sure we use lag_1 period t-bill yields
             as exogenous variable
        temp = rwData['tbill'].shift(periods=1)
        temp.iloc[0] = rwData['tbill'].iloc[0].copy()
        rwData['tbill'] = temp

        train_IDX = rwData['caldt'] > SplitYear * (10 ** 4)
        self.train = rwData[train_IDX]
        self.test = rwData[~train_IDX]
        print('{0}\n[INFO] Finished data importing.'.format('=' * 20 + NOW() +
            '=' * 20))


class PriorParameters:
```

```
    def __init__(self, TrainData, Seed=rand.randint(1)):
        rand.seed(Seed)
        TrainLen = TrainData.shape[0]

        def BetaPrior():
            Y = TrainData['vwretd']
            X = TrainData['tbill']
            X = sm.add_constant(X)
            Model = sm.OLS(Y, X)
            Results = Model.fit()

            MeanVec = Results.params
            CovMat = [[9, 0], [0, 9]]
            Beta = dict()
            Beta['Value'] = rand.multivariate_normal(mean=MeanVec, cov=CovMat)
            Beta['Mean'] = MeanVec
            Beta['Cov'] = CovMat
            return Beta

        Beta = BetaPrior()

        def AlphaPrior():
            MeanVec = [0.4, 0.8]
            CovMat = [[0.09, 0], [0, 0.04]]
            Alpha = dict()
            Alpha['Value'] = rand.multivariate_normal(mean=MeanVec, cov=CovMat)
            Alpha['Mean'] = MeanVec
            Alpha['Cov'] = CovMat
            return Alpha

        Alpha = AlphaPrior()
        # this abs(Alpha_2) <= 1 constraint makes sure that our AR(1) for
            volatility is stationary
        while np.abs(Alpha['Value'][1] >= 1):  Alpha = AlphaPrior()

        def SigmaPrior():
            Lambda = 0.2
            m = 5
            # DegreeOfFreedom = TrainLen + m - 1
            # sigma_sq_inv = rand.chisquare(DegreeOfFreedom)
            sigma_sq = dict()
            # sigma_sq['Value'] = float(m * Lambda) / sigma_sq_inv
            sigma_sq['Value'] = 0.5
            sigma_sq['Lambda'] = Lambda
            sigma_sq['m'] = m

            return sigma_sq

        Sigma_Sq = SigmaPrior()

        Epsilon_vec = rand.randn(TrainLen)
        # this following initialization of H comes from Eq. (10.20) in [Tsay;
            2002]
        H = np.square((TrainData['vwretd'] - Beta['Value'][0] - Beta['Value'
            ][1] * TrainData['tbill']) / Epsilon_vec)
        H[H == 0] = 1e-5  # because we wish to calculate log of H_i's, we need
            to avoid zeros
        H = H.tolist()

```

```
109         self.Beta = Beta
110         self.Alpha = Alpha
111         self.Sigma_Sq = Sigma_Sq
112         self.H = H
113         print('{0}\n[INFO] Finished initialization of parameters.'.format('=' *
                20 + NOW() + '=' * 20))
114
115
116 def UpdateParameters(Parameters, TrainDF):
117     R_Vec = TrainDF['vwretd']
118     temp = R_Vec.shift(periods=1)
119     temp.iloc[0] = R_Vec.iloc[0].copy()
120     X_Vec = temp
121     # X_Vec = TrainDF['tbill']
122     R_Vec = TrainDF['vwretd']
123     X_Mat = np.array([np.ones_like(X_Vec), X_Vec])
124     Log_PriorH = np.log(Parameters.H)
125     Lag1_IDX = [0] + range(len(Log_PriorH) - 1)
126     Log_Lag1_PrioH = Log_PriorH[Lag1_IDX]
127
128     def UpdateBeta():
129         # this following updating algorithm comes from Page 419 in [Tsay; 2002]
130         OldMean = Parameters.Beta['Mean']
131         OldCov = Parameters.Beta['Cov']
132         NewCov = invert(np.dot(X_Mat, np.transpose(X_Mat)) + invert(OldCov))
133         NewMean = np.dot(NewCov, np.dot(X_Mat, R_Vec) + np.dot(invert(OldCov),
                OldMean))
134         NewValue = rand.multivariate_normal(mean=NewMean, cov=NewCov)
135
136         NewBeta = {
137             'Value': NewValue,
138             'Mean': NewMean,
139             'Cov': NewCov
140         }
141         return NewBeta
142
143     Parameters.Beta = UpdateBeta()
144
145     def UpdateAlpha():
146         # this following updating algorithm comes from Page 420 in [Tsay; 2002]
147         OldMean = Parameters.Alpha['Mean']
148         OldCov = Parameters.Alpha['Cov']
149         Sigma_Sq = Parameters.Sigma_Sq['Value']
150         Z_Mat = np.array([[1] * len(Log_Lag1_PrioH), Log_Lag1_PrioH.tolist()])
151         NewCov = invert(np.dot(Z_Mat, np.transpose(Z_Mat)) / Sigma_Sq + invert(
                OldCov))
152         NewMean = np.dot(NewCov, np.dot(Z_Mat, np.transpose(Log_PriorH)) /
                Sigma_Sq + np.dot(invert(OldCov), OldMean))
153         NewValue = rand.multivariate_normal(mean=NewMean, cov=NewCov)
154
155         NewAlpha = {
156             'Value': NewValue,
157             'Mean': NewMean,
158             'Cov': NewCov
159         }
160         return NewAlpha
161
162     Parameters.Alpha = UpdateAlpha()
163
```

```
    def UpdateSigma():
        # this following updating algorithm comes from Page 420 in [Tsay; 2002]
        Alpha = Parameters.Alpha['Value']
        Lambda = Parameters.Sigma_Sq['Lambda']
        m = Parameters.Sigma_Sq['m']
        v = Log_PriorH - Alpha[0] - Alpha[1] * Log_Lag1_PrioH
        Numerator = m * Lambda + np.sum(np.square(v))
        Chi2Draw = rand.chisquare(df=m + len(Log_PriorH) - 1)
        NewValue = Numerator / Chi2Draw
        NewSigma_Sq = Parameters.Sigma_Sq.copy()
        NewSigma_Sq['Value'] = NewValue
        return NewSigma_Sq

    Parameters.Sigma_Sq = UpdateSigma()

    def UpdateH():
        Alpha = Parameters.Alpha['Value']
        HVec = Parameters.H[:]

        def CalcPI(H_This, H_Minus, H_Plus):
            PART1 = (R_Vec.iloc[idx] - X_Vec.iloc[idx] * Parameters.Beta['Value
                ']][1]) ** 2 / (2 * H_This)

            mu = Alpha[0] * (1 - Alpha[1]) + Alpha[1] * (np.log(H_Minus) + np.
                log(H_Plus)) / (1 + Alpha[1] ** 2)
            sigma_sq = Parameters.Sigma_Sq['Value'] / (1 + Alpha[1] ** 2)
            PART2 = (np.log(H_This) - mu) ** 2 / (2 * sigma_sq)

            PI = H_This ** (-1.5) * np.exp(-PART1 - PART2)
            return PI

        for idx, H_This in enumerate(HVec):
            if idx == len(HVec) - 1 or idx == 0: continue  # edge case for H_0
                and H_n
            H_Minus = HVec[idx - 1]
            H_Plus = HVec[idx + 1]

            # the following acception/rejection scheme is called 'Metropolis
                Algorithm'
            # see updating scheme on pg. 419 of Tsay
            Pi_Old = CalcPI(H_This, H_Minus, H_Plus)
            Q_Old = gamma.pdf(H_This, 1)

            H_Draw = rand.gamma(1)
            Pi_New = CalcPI(H_Draw, H_Minus, H_Plus)
            Q_New = gamma.pdf(H_Draw, 1)

            if Q_New != 0 and Pi_Old != 0:
                AcceptProbability = 1
            else:
                AcceptProbability = min([Pi_New * Q_Old / (Pi_Old * Q_New), 1])

            if rand.uniform(low=0, high=1) <= AcceptProbability:  HVec[idx] =
                H_Draw

        return HVec

    Parameters.H = UpdateH()
```

```
218
219 def main(NRound, NTrial):
220     # -------------Data Preparations-----------
221     rwData = ReadData(SplitYear=2013)
222     TrainDF = rwData.train[['vwretd', 'tbill']]
223     TestDF = rwData.test[['vwretd', 'tbill']]
224
225     # -------------Initializing Priors----------
226     Priors = PriorParameters(TrainDF)
227
228     # TODO: Explore possibilities of parallel running
229     # -----------------Training----------------
230     AverageContainer = {'Alpha': np.empty(shape=(NTrial, 2)),
231                         'Beta': np.empty(shape=(NTrial, 2)),
232                         'Sigma_Sq': np.empty(shape=NTrial),
233                         'H': np.empty(shape=NTrial)
234                         }
235     Y_Mean = np.mean(TrainDF['vwretd'])
236     TrainSqSum_TOTAL = np.sum(np.square(TrainDF['vwretd'] - Y_Mean))
237
238     R_Vec = TrainDF['vwretd']
239     temp = R_Vec.shift(periods=1)
240     temp.iloc[0] = R_Vec.iloc[0].copy()
241     X_Vec = temp
242
243     for trial in range(NTrial):
244
245         RoundCount = 0;
246         Sum_RESID = 1.0;
247         MSE = 1000.0;
248         MSE_Update = 1.0
249
250         while RoundCount <= NRound and np.abs(Sum_RESID) > 1e-2 and MSE_Update
                > 1e-2:
251             UpdateParameters(Priors, TrainDF)
252             # -------------Calculate the current RE, Sum_RE, MMSE and R_Sq
253
254             B_Vec = Priors.Beta['Value'][0] + Priors.Beta['Value'][1] * X_Vec
255             Fitted_Vec = B_Vec
256
257             RESID = TrainDF['vwretd'] - Fitted_Vec
258             Sum_RESID = np.sum(RESID)
259             Sq_RESID = np.square(RESID)
260
261             Old_MSE = MSE
262             MSE = np.mean(Sq_RESID)
263             MSE_Update = np.abs(MSE - Old_MSE)
264
265             R_Sq = np.sum(Sq_RESID) / float(TrainSqSum_TOTAL)
266             RoundCount += 1
267
268             print(
269                 '{0}\n[INFO] Finished {1}th round of updating parameters using
                        MCMC with:\n * Mean Squared Error={2};\n * R2={3}%;\n'.
270                 format('=' * 20 + NOW() + '=' * 20, RoundCount, MSE, 100 * R_Sq
                        ))
271
272         if RoundCount > NRound:
273             print(
```

```
                '{0}\n[INFO] Successfully finished {1}th trial with convergence
                    . Final in-sample statistics are:\n* Mean Squared Error
                    ={2}\n * R2={3}%\n'.
                format('=' * 20 + NOW() + '=' * 20, trial, MSE, 100 * R_Sq))
        else:
            print(
                '{0}\n[WARNING] Exit {1}th trial without convergence. Final in-
                    sample statistics are:\n * Mean Squared Error={2}\n * R2
                    ={3}%\n'.
                format('=' * 20 + NOW() + '=' * 20, trial, MSE, 100 * R_Sq))

        AverageContainer['Alpha'][trial] = Priors.Alpha['Value']
        AverageContainer['Beta'][trial] = Priors.Beta['Value']
        AverageContainer['Sigma_Sq'][trial] = Priors.Sigma_Sq['Value']
        AverageContainer['H'][trial] = Priors.H[-1]
    OptimalParameters = {
        'Alpha': np.mean(AverageContainer['Alpha'], axis=0),
        'Beta': np.mean(AverageContainer['Beta'], axis=0),
        'Sigma_Sq': np.mean(AverageContainer['Sigma_Sq']),
        'H': np.mean(AverageContainer['H'])
    }
    print('{0}\n[INFO] Training results:{1}'.format('=' * 20 + NOW() + '=' *
        20, OptimalParameters))


    #---------------Prediction----------------
    TestLen = TestDF.shape[0]
    Epsilon_vec = rand.randn(TestLen)

    # this following initialization of H comes from Eq. (10.20) in [Tsay; 2002]
    Alpha_0, Alpha_1 = OptimalParameters['Alpha']
    Beta_0, Beta_1 = OptimalParameters['Beta']

    R_Vec = [1] * TestLen
    temp = TestDF['vwretd'].shift(periods=1)
    temp.iloc[0] = TestDF['vwretd'].iloc[0].copy()
    X_test_Vec = temp

    for idTrial in range(100):
        H_Last = OptimalParameters['H']
        for idx in range(TestLen):
            V_t = (OptimalParameters['Sigma_Sq']**0.5)*rand.randn()
            H = np.exp(Alpha_0 + np.log(H_Last) * Alpha_1 + V_t)
            A = np.sqrt(H) * Epsilon_vec[idx]
            R_Vec[idx] += Beta_0 + Beta_1 * X_test_Vec[idx] + A
            H_Last = H

    #--------Evaluating the model: Calculation of MMSE-----------------
    R_Vec = [element/100.0 for element in R_Vec]
    RESID = TestDF['vwretd']-R_Vec
    Sq_RESID = np.square(RESID)
    MMSE = np.mean(Sq_RESID)
    # --------Evaluating the model: Calculation of R_Sq-----------------
    SqSum_RESID = np.sum(Sq_RESID)
    SqSum_TOTAL = np.sum(np.square( TestDF['vwretd'] - np.mean( TestDF['vwretd'
        ])))
    R_Sq = 1-(SqSum_RESID/SqSum_TOTAL)
    print('{0}\n[INFO] Successfully exit the program with the following
        prediction results:\n * MMSE={1};\n * R_Sq={2}%'.
```

```
            format('='*20+NOW()+'='*20, MMSE, R_Sq*100))


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Conducting MCMC.')

    parser.add_argument('-r', action='store', dest='NRound', default='6000',
                        help='This argument helps specifies how many iterations
                             we run within MCMC.\n' +
                              'If you have input a decimal number, the code will
                                  take the floor int.')

    parser.add_argument('-t', action='store', dest='NTrial', default='1',
                        help='This argument helps specifies how many iterations
                             we run the entire MCMC.\n' +
                              'If you have input a decimal number, the code will
                                  take the floor int.')

    args = parser.parse_args()
    NRound = int(args.NRound)
    NTrial = int(args.NTrial)
    assert (args.NRound > 0) and (
    args.NTrial > 0), '[ERROR] Please give a valid simulation iterations
        command (i.e. must be positive)!'
    main(NRound, NTrial)
```

Please refer to our GitHub repository page {https://github.com/jiacheng0409/mcmc_sv} for the ready-for-download version of the full Python implementation, as well as the data we used.

References

[1] Box and Tiao. *Bayesian Inference in Statistical Analysis*. John Wiley & Sons, 1973.

[2] Johannes Eraker and Polson. The impacts of jumps in volatility and returns. *The Journal of Finance*, 2003.

[3] Andrew Gelman. Prior distribution. *Encyclopedia of environmetrics*.

[4] Andrew Gelman et al. Objections to bayesian statistics. *Bayesian Analysis*, 3(3):445–449, 2008.

[5] Walter R Gilks. Markov chain monte carlo. *Encyclopedia of Biostatistics*, 2005.

[6] Charles M Grinstead and J Laurie Snell. Markov chains. *Introduction to probability*, pages 405–470, 1997.

[7] Hansen and Lunde. A forecast comparison of volatility models: Does anything beat a garch(1,1)? *Journal of Applied Econometrics*, 2005.

[8] Polson Jacquier and Rossi. Bayesian analysis of stochastic volatility models. *Journal of Business and Economic Statistics*, 1994.

[9] Michael S Johannes and Nick Polson. Mcmc methods for continuous-time financial econometrics. 2003.

[10] Peter M Lee. *Bayesian statistics: an introduction*. John Wiley & Sons, 2012.

[11] Scheicher Lehar and Schittenkopf. Garch vs stochastic volatility: Option pricing and risk management. *Journal of Banking and Finance*, 2001.

[12] Merton. Option pricing when underlying stock returns are discontinuous. *Journal of Financial Economics*, 1976.

[13] Ruey S Tsay. Markov chain monte carlo methods with applications. *Analysis of financial time series*, pages 395–443, 2002.