PROCEEDINGS OF SCIENCE

An application of the hybrid Monte Carlo algorithm for realized stochastic volatility model

Tetsuya Takaishi*
Hiroshima University of Economics, Hiroshima. Japan
E-mail: tt-taka@hue.ac.jp

Yubin Liu
Nankai University, Tianjin, China
E-mail: liuyb@nankai.edu.cn

Ting Ting Chen
Hiroshima University, Hiroshima, Japan

The hybrid Monte Carlo (HMC) algorithm has been widely used in dynamical lattice quantum chromo-dynamics simulations. An advantage of using the HMC algorithm is that it is a global algorithm that can update all of the link variables simultaneously, thereby greatly reducing the computational cost related to the fermionic part. We exploit this advantage for estimating the parameters of the realized stochastic volatility (RSV) model, which is used for modeling time series data. The RSV model includes a number of volatility variables that need to be updated, and thus we update these variables using the HMC algorithm. We found that the HMC algorithm effectively decorrelates Monte Carlo samples of volatility variables. We also show that the algorithm can be accelerated by GPU computing with OpenACC.

The 33rd International Symposium on Lattice Field Theory
14 -18 July 2015
*Kobe International Conference Center, Kobe, Japan**

*Speaker.

PoS(LATTICE 2015)039

1. Introduction

The hybrid Monte Carlo (HMC) algorithm[1] was developed for dynamical lattice quantum chromo-dynamics (QCD) simulations and it is a global method that can update link variables simultaneously in lattice QCD simulations. This property is advantageous for lattice QCD simulations because it greatly reduces the computational cost related to the fermionic part. In addition to its use for lattice QCD simulations, the HMC algorithm is viewed as a Markov Chain Monte Carlo (MCMC) method. Therefore, the HMC algorithm has potential in many other fields where the MCMC technique is required.

In empirical finance, it is crucial to estimate the volatility (or variance) of asset price returns to manage risk. However, this volatility is not an observable in financial markets, and thus an estimation technique is required such as volatility modeling. A widely used volatility model is the stochastic volatility (SV) model[1][2], which allows the volatility to be treated as a stochastic process. The parameters of the SV model need to be determined so the model matches the observed data. In general, the parameters are estimated for the SV model by Bayesian inference using MCMC methods. The most time-consuming part of the MCMC method for the SV model is volatility updating, which requires the updating of a number of volatility variables. The HMC algorithm has been employed for volatility updating and it was shown that the HMC algorithm can rapidly decorrelate Monte Carlo samples of volatility [3, 4, 5].

Recently, an extended version of the SV model was proposed that utilizes the realized volatility (RV)[9, 10] as additional information, which is called the realized stochastic volatility (RSV) model[11]. In this study, we propose a Bayesian inference method for the RSV model using the HMC algorithm. The HMC algorithm can be readily parallelized, so we performed Bayesian inference by GPU computing and we compared the performance with computation using a CPU machine.

2. SV Model

The standard SV model[2] is written as

$$y_t = \exp(h_t/2)\varepsilon_t, \quad \varepsilon_t \sim N(0,1), \tag{2.1}$$

$$h_{t+1} = \mu + \phi(h_t - \mu) + \eta_t, \quad \eta_t \sim N(0,\sigma_\eta^2), \tag{2.2}$$

where y_t for $t = 1,...,T$ is a daily return at time t and h_t is a latent volatility defined by $\ln\sigma_t^2$. This model includes three parameters, (ϕ,μ,σ_η^2), which we need to estimate from daily returns data. The standard estimation technique is Bayesian inference using the MCMC method. The most time-consuming part of the MCMC approach for SV models is volatility updating[12]. Several MCMC approaches have been developed to improve the efficiency of volatility updating, such as the multi-move sampler[13, 14]. The HMC algorithm has also been employed for this purpose and it was shown that the HMC can accelerate the decorrelation between Monte Carlo samples of volatility[3, 4, 5].

[1] Another popular volatility model is the GARCH model[6, 7], which models a deterministic volatility process. The HMC algorithm also has been applied to GARCH parameter estimation[8].

3. RSV Model

The RSV model introduced by Takahashi *et al.*[11] is written as

$$y_t = \exp(h_t/2)\varepsilon_t, \quad \varepsilon_t \sim N(0,1), \tag{3.1}$$

$$\ln RV_t = \xi + h_t + u_t, \quad u_t \sim N(0,\sigma_u^2), \tag{3.2}$$

$$h_{t+1} = \mu + \phi(h_t - \mu) + \eta_t, \quad \eta_t \sim N(0,\sigma_\eta^2), \tag{3.3}$$

where RV_t for $t = 1,...,T$ is a daily realized volatility at time t. The model parameters that we need to estimate are $\theta = \phi, \mu, \xi, \sigma_\eta^2, \sigma_u^2$. In this study, we propose to estimate the model parameter θ by Bayesian inference and we also use the HMC algorithm for volatility updating in MCMC simulations.

4. HMC algorithm

The HMC algorithm combines molecular dynamics (MD) simulation and the Metropolis accept/reject test. The basic HMC algorithm for the RSV model is as follows. First, the next candidate volatility variables are generated by solving the Hamilton's equations of motion in fictitious time τ,

$$\frac{dh_i}{d\tau} = \frac{\partial H}{\partial p_i}, \tag{4.1}$$

$$\frac{dp_i}{d\tau} = -\frac{\partial H}{\partial h_i}, \tag{4.2}$$

where p_i for $i = 1,...,T$ is a conjugate momentum to h_i. The Hamiltonian H is defined by $H(p,h) = \frac{1}{2}\sum_i^T p_i^2 - \ln f(h,\theta)$, where $f(h,\theta)$ is the conditional posterior density of the RSV model[11]. In general, Eqs.(4.1)–(4.2) cannot be solved analytically, so we integrate them numerically by MD simulation. The standard integrator for the MD simulation in the HMC algorithm is the second order leapfrog integrator[1] given by

$$h_i(\tau + \delta\tau/2) = h_i(\tau) + \tfrac{\delta\tau}{2} p_i(\tau), \tag{4.3}$$

$$p_i(\tau + \delta\tau) = p_i(\tau) - \delta\tau \tfrac{\partial H}{\partial h_i}, \tag{4.4}$$

$$h_i(\tau + \delta\tau) = h_i(\tau + \delta\tau/2) + \tfrac{\delta\tau}{2} p_i(\tau + \delta\tau), \tag{4.5}$$

where $i = 1,...,T$ and $\delta\tau$ denotes the step size. Higher order or other integrators[15]-[18] can also be used in the HMC algorithm if necessary. Actually, for the RSV model, it was found that the MN integrator[17, 18] is superior to the leapfrog integrator[19]. Eqs. (4.3)–(4.5) are repeated k times and we then obtain the total integration length $l = k \times \delta\tau$. After the MD simulations, we obtain new volatility and conjugate momentum variables, which are denoted as $h_i' = h(\tau + l)$ and $p_i' = p(\tau + l)$. The new volatility variables h_i' are accepted at the Metropolis step with a probability of $\sim \min\{1, \exp(-\Delta H)\}$, where $\Delta H = H(p',h') - H(p,h)$.

Table 1: GTX 760 Specifications[20]

GPU Engine Specifications	
CUDA Cores	1152
Base Clock (MHz)	980
Boost Clock (MHz)	1033
Memory Specifications	
Memory Speed	6.0 Gpbs
Memory Config	2048 MB
Memory Interface	GDDR5
Memory Interface Width	265-bit
Memory Bandwidth (GB/s)	192.2

5. Environment for GPU coding

In this study, we used the NVIDIA GeForce GTX760 for GPU computing. Table 1 shows the specifications of the GTX760[20]. The original HMC code for the RSV model on a single CPU was developed by [21] and it was modified for GPU code using OpenACC[22] in PGI Fortran[23]. We also executed the code on a CPU (Intel i7-4770, 3.4 GHz) to compare the performance using a GPU and CPU.

6. HMC algorithm in OpenACC

OpenACC allows directive-based programming for use on a GPU, which can greatly reduce the coding effort required. The following is a schematic representation of the OpenACC coding process.

!$acc data copy(h,p)
do loop, repeat k **times up to** l **trajectory length**
!$acc kernels

$$h_i(\tau+\delta\tau/2) = h_i(\tau) + \tfrac{\delta\tau}{2} p_i(\tau) \qquad i=1,\ldots,T \qquad (6.1)$$

$$p_i(\tau+\delta\tau) = p_i(\tau) - \delta\tau \tfrac{\partial H}{\partial h_i} \qquad i=1,\ldots,T \qquad (6.2)$$

$$h_i(\tau+\delta\tau) = h_i(\tau+\delta\tau/2) + \tfrac{\delta\tau}{2} p_i(\tau+\delta\tau) \qquad i=1,\ldots,T \qquad (6.3)$$

!$acc end kernels
end do loop
!$acc end data

Eqs. (6.1)–(6.3) between "!$acc kernels" and "!$acc end kernels" are translated automatically into GPU code and executed on the GPU. The data directive "!$acc data copy(h,p)" specifies the place where the variables (h and p) are transferred to the GPU. This avoids unnecessary data transfers between the CPU and GPU when the kernels, Eqs. (6.1)–(6.3), are called. It has been shown that the appropriate insertion of a data directive can achieve speedup similar to CUDA Fortran[19].

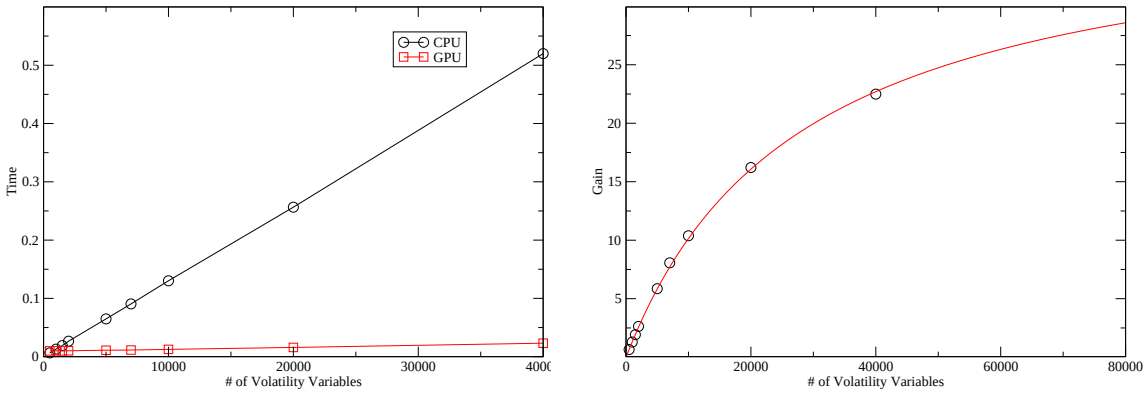

Figure 1: (Left) Average time required to compute 200 leapfrog steps. (Right) Gain obtained using GPU computation compared with a CPU.

Table 2: Fitting parameters

	A	C
GPU (OpenACC)	9.41×10^{-3}	3.36×10^{-7}
CPU (Intel i7-4770, 3.4 GHz)	-1.85×10^{-4}	1.30×10^{-5}

To compare the performance using the GPU and CPU, we measured the time required to compute the leapfrog step. Figure 1 (left) shows the average time required to compute 200 leapfrog steps as a function of the number of volatility variables. The computational time was linear for both the GPU and CPU. We fitted the results with a linear function: $f(n) = A + C \times n$, where n is the number of volatility variables and A, C are the parameters. The fitting results are provided in Table 2. We defined the speedup of the GPU over the CPU by: $Gain = f_{CPU}(n)/f_{GPU}(n)$. Figure 1 (right) shows the Gain as a function of n. The Gain increased with n and it was predicted to reach $C_{CPU}/C_{GPU} \approx 37$ in the limit of $n \to \infty$.

7. Empirical application

We applied Bayesian inference to an RSV model of stock price data for Nissan Motor Co. based on trades on the Tokyo Stock Exchange between July 3, 2006 and 20 December 20, 2009. Figure 2 (left) shows the return time series data for Nissan Motor Co. Using high-frequency data, we calculated the RV at a sampling frequency of 1 min. Figure 2 (right) shows the RV as a function of time (days). The returns and RV data were used as input data for Bayesian inference in the RSV model. The HMC algorithm was performed with the MN integrator[18] with a step size of $\delta\tau = 1/9$ and trajectory length $l = 1$, where the acceptance rate at the Metropolis step was found to be about 0.82. We discarded the first 5000 Monte Carlo samples and we then collected 50000 samples for analysis. The results using the parameters obtained by the MCMC simulations are listed in Table 3.

Figure 3 shows the autocorrelation function of the volatility Monte Carlo samples. The result for h_{100} is shown as a representative example. The integrated autocorrelation time was calculated

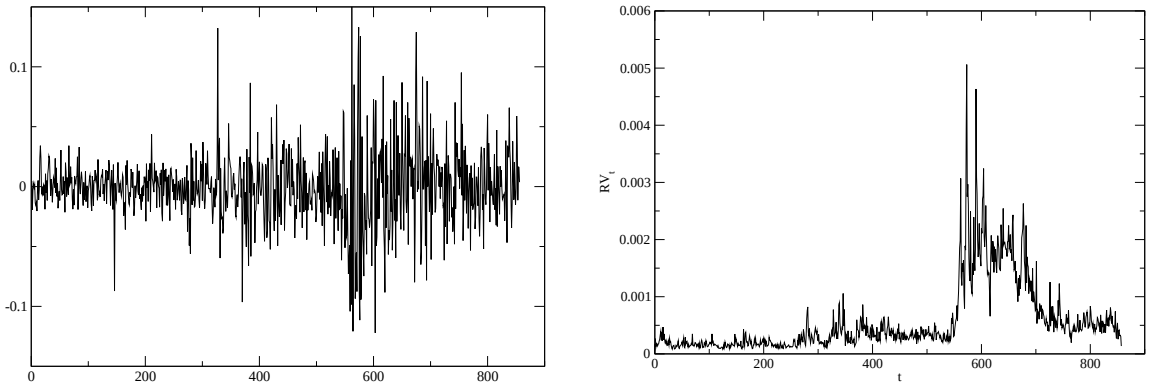

Figure 2: (Left) Return time series of stock price data for Nissan Motor Co. trades on the Tokyo Stock Exchange from July 3, 2006 to December 20, 2009. (Right) RV sampled at a frequency of 1 min from July 3, 2006 to Decembe

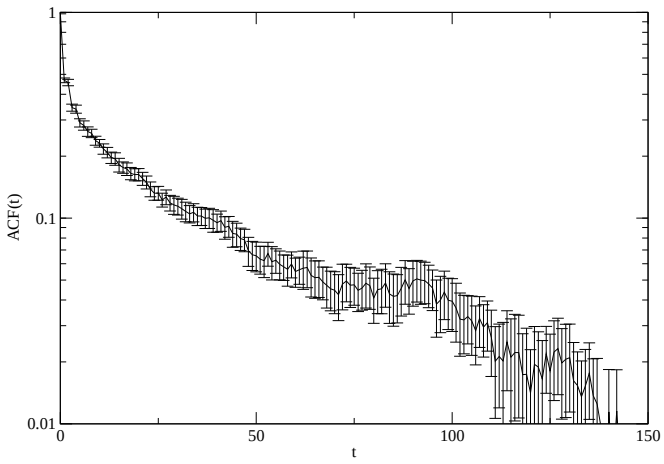

Figure 3: Autocorrelation function for the volatility variable. The result is shown for h_{100} as a representative example.

Table 3: Results of MCMC simulations. The values in parentheses indicate statistical errors.

μ	ϕ	σ_η^2	ξ	σ_u^2
-7.61(5)	0.9849(1)	0.0237(2)	-0.3703(3)	0.0465(1)

as ≈ 13. This autocorrelation time is very short compared with that of $O(100)$ for Metropolis updating[3, 4].

8. Conclusion

In this study, we performed Bayesian inference of the RSV model using the HMC algorithm and we found that the HMC algorithm can rapidly decorrelate Monte Carlo samples of volatility

variables. The HMC algorithm was implemented on a GPU (GeForce GTX 760) with OpenACC coding. A comparison was performed with a CPU (Intel i7-4770, 3.4 GHz), which showed that the GPU was faster than the CPU and the Gain of the GPU compared with the CPU was about 37 with a large number of volatility variables.

Acknowledgments

The numerical calculations in this study were performed at the Yukawa Institute Computer Facility and the facilities of the Institute of Statistical Mathematics. This study was supported by JSPS KAKENHI Grant Number 25330047.

References

[1] S.Duane *et al.* Phys. Lett. B **195** (1987) 216-222.

[2] S.J. Taylor, *Modelling Financial Time Series* (1986) John Wiley & Sons, New Jersey.

[3] T. Takaishi, Lecture Notes in Computer Science **5226** (2008) 929–936.

[4] T. Takaishi, Journal of Circuits, Systems, and Computers **18** (2009) 1381–1396.

[5] T. Takaishi, J. Phys.: Conf. Ser. **423** (2013) 012021.

[6] R.F. Engle, Econometrica **50** (1982) 987–1007.

[7] T. Bollerslev, Journal of Econometrics **31** (1986) 307–327.

[8] T. Takaishi, Proceedings of the 9th Joint Conference on Information Sciences (2006) 214. DOI: 10.2991/jcis.2006.159

[9] T.G. Andersen and T. Bollerslev, International Economic Review **39** (1998) 885–905.

[10] T.G. Andersen, T. Bollerslev, F.X. Diebold and P. Labys, Journal of the American Statistical Association **96** (2001) 42–55.

[11] M. Takahashi, Y. Omori and T. Watanabe, Compt. Stat. & Data Anal. **53** (2009) 2404–2425.

[12] S. Kim, N. Shephard and S. Chib, Review of Economic Studies **65** (1998) 361–393.

[13] N. Shephard and M.K. Pitt, Biometrika **84** (1997) 653–667.

[14] T. Watanabe and Y. Omori, Biometrika **91** (2004) 246–248.

[15] T. Takaishi, Comput. Phys. Commun. **133** (2000) 6–17.

[16] T. Takaishi, Phys. Lett. B **540** (2002) 159–165.

[17] I.P. Omelyan, I.M. Mryglod and R. Folk, Comput. Phys. Commun. **151** (2003) 272–314.

[18] T. Takaishi and P. de Forcrand, Phys. Rev. E **73** (2006) 036706.

[19] T. Takaishi, J. Phys.: Conf. Ser. **574** (2015) 012143.

[20] http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-760

[21] T. Takaishi, J. Phys.: Conf. Ser. **490** (2014) 012092.

[22] http://www.openacc.org/

[23] https://www.pgroup.com/resources/cudafortran.htm