広島経済大学研究論集
第33巻第1号　2010年6月

GPGPUによるストカスティックボラティリティモデルのベイズ推定

高　石　哲　弥*

経済時系列の変動の大きさを表すボラティリティを推定するベイズ推定をNVIDIA社のGPU（GT220）を利用して実行した。本研究ではストカスティックボラティリティモデルを用い，ベイズ推定はマルコフ連鎖モンテカルロ（MCMC）法によって実行した。ボラティリティ変数に対するMCMC法には，ハイブリッドモンテカルロ法を用い，この部分をGPUによって並列計算を行った。計算速度をCPU（AMD社3.0 GHz）と比較することによって，GPUの方が最大で26倍程度速く実行できるという結果が得られた。

1. 導　　入

株価や為替レートの収益率の変動の大きさを表す指標として，ボラティリティと呼ばれる量がある。ボラティリティは金融のリスク管理やオプション価格，ポートフォリオマネージメントなどに利用される重要な量である。しかし，ボラティリティは収益率の時系列データから直接測定できないので，何らかの方法によって推定しなければならない。良く用いられる方法は，ボラティリティの時間変動を表すモデルを仮定して，そのモデルが時系列データに当てはまるようにモデルのパラメータを決定する方法である。そして，その決定したパラメータを利用することによってボラティリティを推定することができる。

近年，パラメータ推定にはマルコフ連鎖モンテカルロ（MCMC）法によるベイズ推定法がよく用いられるようになってきている。MCMC法はコンピュータで実行されるが，時々刻々変化する金融データを取り扱うファイナンスの実務においては，リアルタイムに計算結果を求めることができれば，非常に実用的となる。本研究では，パソコンのグラフィックボードを利用したGPGPU（General Purpose Graphics Processing Unit）による計算を行う。最近のグラフィックボードの中のGPU（Graphics Processing Unit）には計算を行うコアを多数持ったものがあり，多数のコアを利用して並列計算ができれば計算を高速に実行することが可能となる。また，パソコンのグラフィックボードは比較的安価に手に入れることが可能であるので，スーパーコンピュータ並みの計算速度をパソコン上で安価に実現することが可能となる。従って，GPGPU計算がファイナンスの実務でも有効であると実証されれば，安価に高速計算ができる手段となると考えられる。

一般的には，MCMC法は変数を逐次にアップデイトしていくので，全てのモデルが並列計算に向いているとは限らない。ボラティリティ推定のモデルとしては，ARCHやGARCHモデル[1,2]が有名であるが，それらは変数の数が少なく，一般には逐次にアップデイトしていくので並列計算向きはない（並列計算をすることは可能である[3]）。本研究では，ストカスティックボラティリティ（SV）モデル[4,5]を利用する。このモデルには，アップデイトするボラティリティ変数が時系列長と同じ数（典型的には数千

* 広島経済大学経済学部教授

個）存在する。これらの多数の変数は，ハイブリッドモンテカルロ（HMC）法を利用することによって同時にアップデイト，つまり並列計算が可能となる[6]。従って，並列計算に向いているので，本研究ではSVモデルを利用する。そして，並列計算をGPUを利用して実行することによって，CPU計算よりも高速化がなされるかどうかを比較する。

2. ストカスティックボラティリティ（SV）モデル

本章ではSVモデル[4,5]について述べる。時刻 t における収益率を r_t とし，r_t が

$$r_t=\sigma_t\varepsilon_t \tag{1}$$

で表されるとする。ここで，ε_t は平均0，分散1の正規分布に従う確率変数である。σ_t が時系列の変動の大きさを表すボラティリティである。このボラティリティを収益率の時系列から決定しなければならない。SVモデルでは σ_t が時間と共に以下のように確率的に変動すると仮定する。（ここでは，$h_t=\log(\sigma_t^2)$ と置く。）

$$h_t=\mu+\phi(h_{t-1}-\mu)+\eta_t \tag{2}$$

ここで，η_t は平均0，分散 σ_η^2 の正規分布に従う確率変数である。SVモデルで決定しなければならないパラメータは μ,ϕ,σ_η^2 の3つである。これらの3つのパラメータを収益率の時系列からベイズ推定によって決定する。

このモデルにおいてボラティティ変数 $h_t=\log(\sigma_t^2)$ の無条件平均と分散はそれぞれ μ と $\frac{\sigma_\eta^2}{1-\phi}$ になる。

現実の株価や為替レートの収益率の時系列に見られる特徴として，変動の激しい時期が続くボラティリティクラスタリングや収益率分布が裾野の厚いファットテイル分布を示す[7]というのがあるが，SVモデルはこの特徴をよく捉えることができるので，ボラティリティ推定のモデルとしてよく利用されている。

3. ベイズ推定

ベイズの定理を利用することによって，時系列 r_t が与えられたもとでパラメータ $\theta=(\mu,\phi,\sigma_\eta^2)$ の従う確率分布 $p(\theta|r_t)$ が以下のように与えられる。

$$p(\theta|r_t)=\frac{L(r_t|\theta)\pi(\theta)}{f(r_t)} \tag{3}$$

ここで，$L(r_t|\theta)$ は尤度関数，$\pi(\theta)$ はパラメータの事前分布，$f(r_t)$ は規格化定数にあたる。

$\pi(\theta)$ は，事前情報があればそれを利用して関数形を仮定するが，ここでは事前情報なしとして定数と仮定する。規格化定数 $f(r_t)$ は

$$\int d\theta p(\theta|r_t)=1 \tag{4}$$

から決定される。しかし，本研究でのMCMC法では定数は寄与しないので，具体的な値は必要ない。

SVモデルの尤度関数 $L(r_t|\theta)$ の特徴は，ボラティリティ変数が式(2)のように確率的に決まるので，以下のようにボラティリティ変数の関数の積分形となることである。

$$L(r_t|\theta)=\int S(\theta,h_1,\cdots,h_T)dh_1\cdots dh_T \tag{5}$$

T は時系列長でボラティリティ変数の個数に対応する。また，$S(\theta,h_1,\cdots,h_T)$ はパラメータとボラティリティ変数からなる関数である[4]。最尤法では，尤度関数を最大化するパラメータ値としてパラメータが決定されるが，尤度関数が式(5)のように多重積分形で与えられる場合は，最尤法を実行することが難しくなる。そのため，SVモデルではMCMC法によるベイズ推定がよく利用されている。

ベイズ推定において，パラメータの推定値は確率分布 $p(\theta|r_t)$ のもとでのパラメータの期待値として与えられる。

$$\langle\theta\rangle=\int\theta S(\theta,h_1,\cdots,h_T)dh_1\cdots dh_Td\theta\,/\,Z \quad (6)$$

ここで，Z は規格化定数である。この積分は，パラメータとボラティリティ変数に関する積分であるが解析的には実行できないので MCMC 法によって積分を実行する。

4. MCMC 法

MCMC 法では，パラメータとボラティリティ変数を $S(\theta,h_1,\cdots,h_T)$ に比例する確率で生成（アップデイト）させる。そして，多数生成したパラメータの値から式(6)の期待値を平均値として見積もる。例えば，N 個のパラメータ値 θ^i $(i=1,\cdots,N)$ を生成したとすると，平均値は以下の式(7)で与えられる。

$$\langle\theta\rangle=\frac{1}{N}\sum_{i=1}^{N}\theta^i \quad (7)$$

この時，パラメータ値 θ^i が独立であれば，式(7)の統計誤差は $\frac{1}{\sqrt{N}}$ に比例するので，N が大きくなれば，真の期待値に近づく。但し，一般には MCMC 法で生成されるデータは相関を持っているので，相関の小さいデータを生成されることのできる MCMC 法を利用したほうが良い。

SV モデルではパラメータとボラティリティ変数をアップデイトする必要があるので，パラメータとボラティリティ変数のアップデイト部分を分けてアップデイトを実行する。

Ⅰ．パラメータのアップデイト

Ⅱ．ボラティリティ変数のアップデイト

ⅠとⅡを繰り返し実行し，生成したパラメータ値から期待値を平均値として求める。Ⅰの部分については有効なアップデイトの方法[4]が知られているのでその方法を用いる。Ⅱの部分では多数のボラティリティ変数のアップデイトを実行しなければならない。ボラティリティ変数を逐次的にアップデイトすることもできるが，本研究では並列計算が容易なハイブリッドモンテカルロ法を用いる。

5. ハイブリッドモンテカルロ (HMC) 法

HMC 法[8]は1987年に Duane らによって考案された。HMC 法の特徴はアップデイトする変数を一度にアップデイトできることである。このようなタイプのアップデイトをグローバルアップデイトと呼ぶ。一方，メトロポリス法[9,10]のように変数を１つずつアップデイトするタイプをローカルアップデイトと呼ぶ。

HMC 法ではハミルトン方程式の積分（分子動力学法）とメトロポリス法を組み合わせることによってグローバルアップデイトを実現している。以下では，HMC 法によってボラティリティ変数をアップデイトすることを考える。HMC 法ではパラメータの期待値を表す式を以下のように変形する。

$$\begin{aligned}\langle\theta\rangle&=\int\theta S(\theta,h_1,\cdots,h_T)dh_1\cdots dh_Td\theta\,/\,Z\\&=\int\theta\exp(\ln S)dh_1\cdots dh_Td\theta\,/\,Z\\&=\int\theta\exp(V)dh_1\cdots dh_Td\theta\,/\,Z\end{aligned} \quad (8)$$

ここで，$V(\theta,h_1,\cdots,h_t)=\ln S(\theta,h_1,\cdots,h_T)$ である。更に，ボラティリティ変数 h_i に共役な運動量 p_i を導入して，ハミルトニアンを定義する。

$$\begin{aligned}\langle\theta\rangle&=\int\theta\exp(-\frac{1}{2}\sum_{i=1}^{T}p_i^2+V)dh_1\cdots dh_T\\&\quad\times dp_1\cdots dp_Td\theta\,/\,\tilde{Z}\\&=\int\theta\exp(-H(h,p,\theta))dh_1\cdots dh_T\\&\quad\times dp_1\cdots dp_Td\theta\,/\,\tilde{Z}\end{aligned} \quad (9)$$

ここで，

$$H(h,p,\theta)=\frac{1}{2}\sum_{i=1}^{T}p_i^2-V(h,\theta) \quad (10)$$

がハミルトニアンである。

HMC 法では，ボラティリティ変数をアップデイトするためにまず，以下のハミルトン方程式を解いて新しい候補を生成する。

$$\begin{cases} \dot{p}_i = -\dfrac{\partial H}{\partial h_i} \\ \dot{h}_i = \dfrac{\partial H}{\partial p_i} \end{cases} \tag{11}$$

そして，新しい候補をメトロポリス法で採択／棄却する。ここでのハミルトン方程式は解析的に解けないため，Leapfrog 法よって積分する（注：本研究で利用した Leapfrog 法以外の積分法[11-13] も利用可能である）。本研究ではこの Leapfrog 法による積分計算の部分が GPU による並列計算の対象となる。

6. 開発環境

GPU には NVIDIA 社の GT220 (1 GB) を利用した。この GPU には 6 個のマルチプロセッサーがあり，1 個のマルチプロセッサには 8 個の計算コアがあるので，合計48個の計算コアを持っている。現在は最大で240個の計算コアを持った GPU が存在し，また240個の計算コアを複数持った GPU も存在する[14]。パソコンの CPU には，AMD 社の PhenomII X2 545 (3.0 GHz) を利用した。プログラミングには CUDA3.0 環境による C++ (マイクロソフト社) を利用した。GT220 のスペックは表 1 に表示してある。

表 1 GT220 のスペック (NVIDIA 社ホームページより[14])

GPU エンジンスペック	
CUDA プロセッサコア	48個
グラフィッククロック	625 MHz
プロセッサークロック	1360 MHz
メモリースペック	
メモリクロック	790 MHz
メモリ	1 GB DDR3
メモリバンド幅	25.3 GB/sec
メモリインターフェース幅	128-bit

7. GPU による Leapfrog 法の計算

Leapfrog 法の最小ステップ（dt 積分）は以下の①から③の 3 つのステップから成り立つ。(以下では，ボラティリティ変数の積分から始めてあるが，運動量の積分から始めることも可能である)

① ボラティリティ変数の積分
$h[j] = h[j] + p[j] * dt / 2$ カーネル 1

② 運動量の積分
$p[j] = p[j] - F[j] * dt$ カーネル 2

③ ボラティリティ変数の積分
$h[j] = h[j] + p[j] * dt / 2$ カーネル 3

j は j 番目のボラティリティ変数 $h[j]$ 及び運動量 $p[j]$ を指し示す。GPU での並列計算では，1 つの j に関する計算を 1 つのスレッドに対応させて計算することにする。①から③の各ステップでは，$j = 1, \cdots, T$ までの計算を行ってから次のステップに移らなければならない。CUDA では同じブロック内のスレッドの同期をとる関数は存在するが，すべてのスレッドで同期を取る関数は存在しないようである。従って，$j = 1, \cdots, T$ の計算が確実に終わってから次のステップに移るように，①から③は別々の GPU カーネル関数（カーネル 1 ～ 3）としてホスト側から呼び出すことにする。

②の $F[j]$ はハミルトニアンをボラティリティ変数で微分したものに対応する。この部分には $h[j]$ 変数以外にも $h[j-1]$ と $h[j+1]$ を含むので，同じブロック内ではボラティリティ変数をシェアードメモリから読み込むことによって高速化がはかれる可能性がある。

8. プログラム概要

今回のプログラムは以下の流れになっている。

1. 初期化（$r[j], h[j]$）
2. メモリー領域の確保（cudaMalloc）
3. GPU へのデータ（r）転送（cudaMemcpy）

4. パラメータのアップデイト
5. 運動量の生成（$p[j]$）
6. ハミルトニアンの計算
7. GPU へのデータ（h,p）転送（cudaMemcpy）
8. カーネル 1（$h[j]$ の計算）
9. カーネル 2（$p[j]$ の計算）Leapfrog 法
10. カーネル 3（$h[j]$ の計算）
11. CPU へのデータ（h,p）転送（cudaMemcpy）
12. ハミルトニアンの計算，メトロポリス法
13. 結果の出力（パラメータの平均値）

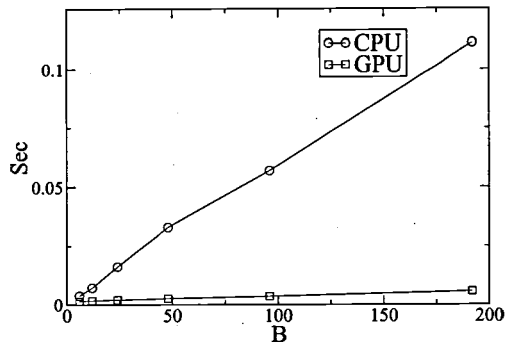
本研究では人工的に作った時系列を利用し MCMC 法を実行した。時系列 $r[j]$ は 1 の初期化の中で発生させる。MCMC 法に当たるのは 4 から12までで，この部分を繰り返し実行する (3000回)。

1 の初期化と 5 の運動量の生成では，正規分布に従う乱数が必要になるが，ここでは乱数を発生させる関数 srand()から一様乱数を生成させ，Box-Muller 法によってそれらの乱数を生成した。

GPU での計算に対応するのは 7 から11までである。8 から10が Leapfrog 法による計算にあたり，この部分を n 回繰り返して $t = n \times dt$ の長さまで積分する。今回は $n = 30$, $dt = 0.005$ とした (注：実際には繰り返しの中で，カーネル 3 とカーネル 1 の部分は 1 つのカーネルとして計算できるが，今回は分けたまま実行した)。時間の計測は CUDA ユーティリティのタイマー関数を利用し，8 から10まで（Leapfrog 法）の経過時間を測定した。

9. GPU と CPU の比較

時系列データ長，即ちボラティリティ変数の個数を $T = 256 \times B$ とする。本研究では，1 ブロック中のスレッド数を256とし，ブロック数 B を変えて経過時間を測定した（ここでは，B は 6 の倍数を設定した)。図 1 は 1 アップデイトあたりの Leapfrog 法に要した平均の時間をプロットしている（注：ここでの GPU の計算で



CPU（○）と GPU（□）

図 1　経過時間

はシェアードメモリは利用していない)。B が非常に小さい所を除き，ほぼ B の 1 次関数となっている。経過時間を $f(B) = A + C * B$ の 1 次関数でフィットして求めた係数 A, C を表 2 にまとめてある。GPU の係数 C 値は CPU の係数 C と比べると非常に小さな値となっている。

表 2　フィッティングによる関数 $f(B)$ の係数 A と C

	A	C
CPU	0.0020	0.00057
GPU	0.0014	2.2e-05

図 2 は CPU での経過時間を GPU での経過時間で割ったもの，即ち CPU に対する GPU の高速化率を表している。(○) は実測値を表し，点線はフィティングによって得られた関数 $f(B)$

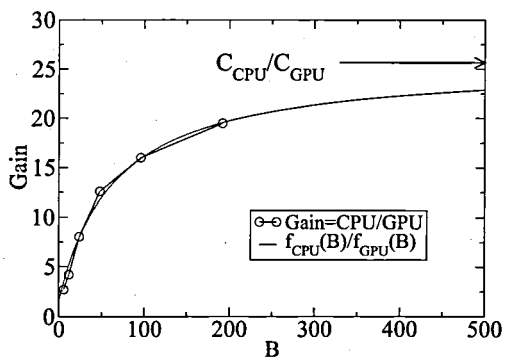


図 2　CPU に対する GPU の高速化率

から求めたものである。B が大きくなるほど高速化率は高く，B が無限大の極限で（メモリ等の制限を無視して）フィティング係数から得られる，$0.00057/2.2e-5 \approx 26$ になると期待される。従って，この結果は時系列データ長が長いほどGPUによる高速化が図れることを示している。但し，実際の実証研究で利用される時系列のデータ長は数千個程度であるので，その場合のGPUの高速化率は5から10倍程度となる。

次に，シェアードメモリを利用した場合と比較する。シェアードメモリはグローバルメモリに比べて100倍ほどアクセススピードが速いので，シェアードメモリの利用によってグローバルメモリとのアクセスを減らすことができれば，その分の高速化が図れると期待できる。

本研究でシェアードメモリが利用できると期待できるのは，②の $F[j]$ の計算部分である。$F[j]$ の計算には $h[j-1]$ と $h[j+1]$ が含まれるが，同じブロック内のシェアードメモリに予め利用するすべての $h[j]$ を置いておくことで，j 番目の計算の時に，この2つの配列をアクセスの遅いグローバルメモリから読み込む必要が無くなる。カーネル1には，グローバルメモリからの配列の読み込みが2回，カーネル2には5回，カーネル3は2回あるが，カーネル2の部分で2回減ることになるので，全体で9回から7回に減少する。

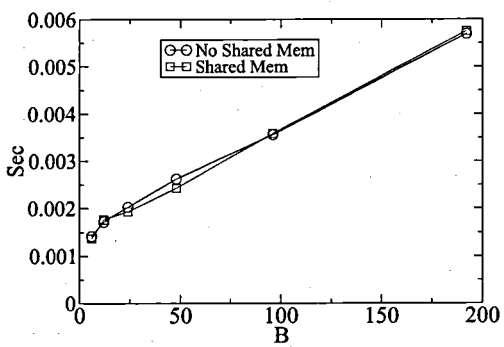


図3 シェアードメモリを利用した場合（□）と利用しなかった場合（○）の経過時間の比較

図3は，シェアードメモリを利用した場合と，利用しなかった場合について，GPUでの経過時間を B に対してプロットしたものである。図からわかるように経過時間には，ほとんど変化が見られず，シェアードメモリによる期待した高速化は見られなかった。理由はよくわからないが，ここで利用したGPUの性質なのかもしれない[15]。

10. まとめと今後の展望

NVIDIA社のGPU（GT220）を用いてSVモデルのベイズ推定を実行した。ベイズ推定には，並列化の容易なHMC法を利用した。HMC法の中のLeapfrog法の計算部分をGPUによって並列計算し，CPU（AMD社3.0 GHz）に対して，高速に計算できることが分かった。高速化率は時系列が長くなれば大きくなり，本研究で利用したGPUとCPUの比較では最大で26倍程度高速化できると期待できることがわかった。だだし，実際の実証研究で利用されている時系列データ長は，数千個程度であるので，その場合は5－10倍程度の高速化率であると考えられる。

シェアードメモリはグローバルメモリよりも100倍程度アクセスが速いので，シェアードメモリの利用による高速化が期待できるが，本研究では期待した高速化は見られなかった。

今回は，Leapfrog法をGPUでの並列化の対象としたが，更にLeapfrog法の前後にあるハミルトニアンの計算部分がGPUで並列化できると考えられる。この部分は，グローバルに和計算をする所なので，うまく和計算が並列化できれば更なる高速化ができるであろう。

付記：本研究は平成20年度広島経済大学特定個人研究費による助成を受けたものである。

注

1) R. F. Engle, Autoregressive Conditional Heteroskedasticity with Estimates of the Variance of the United Kingdom inflation, Econometrica 60 (1982) 987–1007.
2) T. Bollerslev, Generalized Autoregressive Conditional Heteroskedasticity, Journal of Econometrics 31 (1986) 307–327.
3) T. Takaishi, Bayesian Estimation of GARCH model by Hybrid Monte Carlo. Proceedings of the 9th Joint Conference on Information Sciences 2006, CIEF-214 doi:10.2991/jcis.2006.159
4) 渡辺敏明，ボラティティ変動モデル，朝倉書店 (2000).
5) E. Jacquier, N. Polson ad P. E. Rossi, Bayesian Analysis of Stochastic Volatility Models, Journal of Business & Economics Statistics 12 (1994) 371–389.
6) T. Takaishi, Bayesian inference of stochastic volatility model by Hybrid Monte Carlo. Journal of Circuits, and Computers, Vol. 18 (2009) 1381–1296.
7) R. Cont, Empirical Properties of Asset Returns: Stylized Facts and Statistical Issues. Quantitative Finance 1 (2001) 223–236.
8) S. Duane et. al, Hybrid Monte Carlo, Physics Letters B195 (1987) 216–222.
9) Metropolis et. al, Equations of State Calculations by Fast Computing Machines, Journal of Chemical Physics 21 (1953) 1087–1092.
10) Hastings, Monte Carlo Sampling Methods Using Markov Chains and Their Applications, Biometrika 57 (1970) 97–109.
11) T. Takaishi, Choice of Integrator in the Hybrid Monte Carlo Algorithm, Computer Physics Communications 133 (2000) 6–17.
12) T. Takaishi, Higher Order Hybrid Monte Carlo at Finite Temperature, Physics Letters B540 (2002) 159–165.
13) T. Takaishi and Ph. de Forcrand, Testing and tuning symplectic integrators for Hybrid Monte Carlo algorithm in lattice QCD, Physical Review E73 (2006) 036706.
14) NVIDA 社 CUDA ホームページ http://www.nvidia.co.jp/object/cuda_home_new_jp.html
2010年5月時点では480個のコア数を持ったものも存在する。
15) 青木尊之，額田 彰，はじめての CUDA プログラミング，工学社 (2009).