On option pricing in local volatility models using parallel computing

Sergey G. Shorokhov

Peoples' Friendship University of Russia (RUDN University), 6, Miklukho-Maklaya St., Moscow, 117198, Russia

Abstract

Pricing of options and other financial instruments is one of the most important problems in finance. To price an option, one needs first to choose the model of underlying asset price dynamics, usually in the form of a stochastic differential equation depending on market parameters such as interest rate, asset volatility, etc., then find a solution of the chosen model in some form, and finally obtain the required option price. The asset volatility can be constant (Black-Scholes model), depend on the value of the underlying asset and time (local volatility model), or satisfy some other stochastic differential equation (stochastic volatility model). Exact analytical formula for option price was obtained for Black-Scholes model and a few other models with local and stochastic volatility, but in general case one has to use approximations or numerical methods to evaluate options. We study the problem of European option pricing when volatility is a function of underlying asset value and time. To solve the problem, we use Monte Carlo method to construct the empirical distribution density of underlying asset and then determine the option value using Feynman-Katz formula. The issues of parallelization of the option pricing algorithm and its implementation on computers with multicore processors are discussed. Possible applications of local volatility models with parallel computing include modeling and management of large equity portfolios, assessing and managing market and credit risks.

Keywords

option, local volatility, Monte Carlo method, parallel computing

1. Introduction

The increasing amount of data being processed and active use of machine learning methods lead to a growing demand for high-performance computing (HPC) solutions in modern financial industry [1]. The traditional applications of HPC in finance include pricing of financial instruments (derivative securities), algorithmic high-frequency trading, market and credit risk management, etc.

The fundamental problem of finance is the valuation (pricing) of securities and other financial instruments traded in financial markets. For nonlinear derivatives (options), the pricing problem was first solved for constant volatility case by F. Black, M. Scholes [2] and R. Merton [3]. Under certain assumptions the formula obtained in [2] gives us exact fair value of European call option.

Workshop on information technology and scientific computing in the framework of the X International Conference Information and Telecommunication Technologies and Mathematical Modeling of High-Tech Systems (ITTMM-2020), Moscow, Russian, April 13-17, 2020

shorokhov-sg@rudn.ru (S. G. Shorokhov)
https://esystem.rudn.ru/users/3295 (S. G. Shorokhov)
0000-0001-6835-4110 (S. G. Shorokhov)

However, the assumption of constant volatility and based on this assumption Black-Scholes model fail to explain some important effects observed in financial markets, such as volatility smiles [4] and fat tails of financial data distributions [5]. Therefore, alternative stochastic models [6] with volatility depending on the value of the underlying asset and time (local volatility models) should be examined to determine the most appropriate model for available financial data. However, for a general local volatility model, exact closed form formula for the option price has not been derived and miscellaneous numerical methods have to be used.

The paper considers the application of high-performance computing in calculating the value of options under assumption that underlying asset price follows stochastic differential equation (SDE) with volatility, being a function of asset price and time.

2. Local volatility models

In risk-neutral framework the price S of underlying asset is supposed to follow SDE

$$dS = rS\,dt + \sigma(S,\, t)\; S\,dW, \tag{1}$$

where $r > 0$ is the risk-free interest rate, $\sigma(S,\, t) > 0$ is a volatility function of underlying asset value S and time t, W is a Wiener random process, with initial condition

$$S(t_0) = S_0 > 0. \tag{2}$$

We intend to determine the fair value of a derivative (option or other derivative instrument) $V(S_0,\, t_0)$, being a function of the current price of the underlying asset S_0 and current time t_0, with payment function $V(S,\, T) = \psi(S)$ at the time $T > t_0$ of execution (expiration) of the derivative.

Known local volatility models of the form (1) with non-constant asset price volatility and exact closed form solutions for transition density and option price include shifted lognormal model [7], normal (Ornstein–Uhlenbeck) model [8], CEV model [6], hyperbolic sine model [9].

To find the value of an option (derivative) in model (1), one can use the risk-neutral pricing (Feynman–Katz) formula:

$$V(S_0,\, t_0) = e^{-r(T-t_0)} \int\limits_{-\infty}^{+\infty} \rho(y,\, T;\, S_0,\, t_0)\; \psi(y)\, dy, \tag{3}$$

where $\rho(y,\, T;\, S_0,\, t_0)$ is the probability density of the transition from state (S_0, t_0) to state (y, T).

In Black–Scholes model [2, 3] the transition probability density function (pdf) is equal to

$$\rho(x,t;S_0,t_0) = \frac{\exp\left\{-\frac{\left(\ln x - \ln(S_0) - \left(r-\frac{\sigma^2}{2}\right)(t-t_0)\right)^2}{2\sigma^2(t-t_0)}\right\}}{\sqrt{2\pi}\sigma\sqrt{t-t_0}\,x} U(x)\,, \tag{4}$$

where

$$U(x) = \mathbf{1}_{\{x>0\}} = \begin{cases} 1, & x > 0, \\ 0, & x \leqslant 0. \end{cases}$$

In shifted lognormal model [7] the volatility function is $\sigma\left(1-\frac{\alpha e^{rt}}{S}\right)$ and the transition pdf is a shifted version of the transition pdf for Black-Scholes model

$$\rho(x,t;S_0,t_0)=\frac{\exp\left\{-\frac{\left(\ln\left(x-\alpha e^{rt}\right)-\ln\left(S_0-\alpha e^{rt_0}\right)-\left(r-\frac{\sigma^2}{2}\right)(t-t_0)\right)^2}{2\sigma^2(t-t_0)}\right\}}{\sqrt{2\pi}\sigma\sqrt{t-t_0}\,(x-\alpha e^{rt})}U\left(x-\alpha e^{rt}\right). \tag{5}$$

In normal (Ornstein–Uhlenbeck) model [8] the volatility function is $\frac{\sigma}{S}$ and the transition pdf is

$$\rho(x,t;S_0,t_0)=\frac{\sqrt{r}}{\sqrt{\pi}\sigma\sqrt{e^{2r(t-t_0)}-1}}\exp\left\{-r\frac{\left(x-e^{r(t-t_0)}S_0\right)^2}{\sigma^2\left(e^{2r(t-t_0)}-1\right)}\right\}, \tag{6}$$

assuming negative underlying asset values.

In hyperbolic sine model [9] the volatility function is $\sqrt{\frac{\sigma^2}{S^2}+2r}$ and the transition pdf is

$$\rho(x,t;S_0,t_0)=\frac{\exp\left\{-\frac{1}{4r(t-t_0)}\left(\operatorname{arsinh}\left(\frac{\sqrt{2r}}{\sigma}x\right)-\operatorname{arsinh}\left(\frac{\sqrt{2r}}{\sigma}S_0\right)\right)^2\right\}}{\sqrt{2\pi}\sqrt{\sigma^2+2rx^2}\sqrt{t-t_0}}, \tag{7}$$

also assuming negative underlying asset values.

3. Monte Carlo method for option pricing in local volatility models

The Feynman–Katz formula (3) for risk-neutral derivative pricing includes the transition density function $\rho(y,\ T;\ S_0,\ t_0)$, which is the distribution density of the underlying asset $S(T)$ at derivative execution (expiration) time T for fixed initial values S_0 and t_0. For a general local volatility model (1) the distribution of $S(T)$, basically, is unknown.

For approximate determination of the distribution density of the random variable $S(T)$, one can use Monte Carlo method [10], which implies generation of a large number of realizations of the random variable $S(T)$, namely, a large bundle of trajectories of SDE (1) with initial condition (2) with time varying from t_0 to T.

The time segment $[t_0,\ T]$ is divided into N equal parts of length $\triangle t=\frac{T-t_0}{N}$.

Let $t_k=t_0+k\triangle t$, $k=\overline{1,\ N}$, then the simplest numerical approximation of the trajectories of SDE (1) is Euler–Maruyama scheme [11]:

$$S_{k+1}\approx S_k+rS_k\triangle t+\sigma(S_k,\ t_k)\ S_k\sqrt{\triangle t}\,\varepsilon, \tag{8}$$

where $S_k=S(t_k)$, $\varepsilon\sim\mathcal{N}(0,\ 1)$ is a random variable with standard normal distribution. Examples of more complex approximations of solutions of SDE (1) include Milstein scheme [12] and the stochastic Runge–Kutta methods [13].

The simulation of trajectories of SDE (1) leads to a set of values of the underlying asset S at time T of the form $S(T) = S_N^{(i)}, i = \overline{1, n}$, where n is the number of simulated trajectories of SDE (1).

Let $m_1 = \min\left\{S_N^{(i)}\right\}_{i=1}^n, m_2 = \max\left\{S_N^{(i)}\right\}_{i=1}^n$. The segment $[m_1, m_2]$ is divided into M equal parts of length $h = \frac{m_2 - m_1}{M}$ and let ρ_j be the number of values from the set $\left\{S_N^{(i)}\right\}_{i=1}^n$ that fall into the last segment $[m_1 + (M-1)h, m_1 + Mh]$ for $j = M$ and half-intervals of the form $[m_1 + (j-1)h, m_1 + jh)$ for $j = \overline{1, M-1}$.

Empirical distribution density of the random variable $S(T)$ can be obtained by the formula

$$\rho(y, T; S_0, t_0) = \begin{cases} 0, & y < m_1 \\ \frac{1}{n}\rho_j, & m_1 + (j-1)h \leqslant y < m_1 + jh, \; j = \overline{1, M-1}, \\ \frac{1}{n}\rho_M, & m_1 + (M-1)h \leqslant y \leqslant m_1 + Mh, \\ 0, & y > m_2. \end{cases} \tag{9}$$

Then, using the empirical density (9) in risk-neutral pricing formula (3), we obtain the following approximation for the derivative value $V(S_0, t_0)$

$$V(S_0, t_0) \approx e^{-r(T-t_0)} \frac{m_2 - m_1}{Mn} \sum_{j=1}^{M} \rho_i \, \psi\left(m_1 + \left(j - \frac{1}{2}\right) h\right). \tag{10}$$

For a European call option with strike price K the payment function $\psi(y)$ is equal to $\max(y - K, 0)$, so the price of call option is

$$V(S_0, t_0) \approx e^{-r(T-t_0)} \frac{m_2 - m_1}{Mn} \sum_{j=\left[\frac{K-m_1}{h} + \frac{1}{2}\right]}^{M} \rho_j \left(m_1 + \left(j - \frac{1}{2}\right) h - K\right), \tag{11}$$

where the square brackets in (11) stand for the integer part of the number.

The values $S_N^{(i)}$, $i = \overline{1, n}$ can be calculated in parallel, and this makes it possible to use parallel computing in option pricing using formula (10) or formula (11).

4. Parallel computing in Python

There are different ways to organize parallel computing in Python [14], including multi-threading (module `threading`) and multi-processor code execution (method `fork` of module `os` or module `multiprocessing`). If parallel processes do not interact with each other, then parallel computations are better done using the `multiprocessing` module API, which is a part of the standard Python library.

To execute multiple calls of the function `lvm_sde`, which returns the value $S_N^{(i)}$, in parallel, module `multiprocessing` is imported, an object `lvm_pool` of class `Pool` is created, the function `lvm_sde` is launched for parallel execution using method `map`, which applies the function `lvm_sde` to the list of input parameters `input_list` :

```
import multiprocessing as mp
lvm_pool = mp.Pool()
result = lvm_pool.map( lvm_sde, input_list )
```

The number of calls of the function `lvm_sde` corresponds to the length of the `input_list` list. The function `lvm_sde` for calculating $S_N^{(i)}$ according to formula (8) is defined as follows:

```
import numpy as np

def lvm_sde(_):
    np.random.seed()
    s_t = s0
    t = t_start
    dt = ( t_stop - t_start ) / N
    noise = np.random.normal( loc=0.0, scale=1.0, size=N ) * np.sqrt( dt )
    for eps_sqrt_dt in noise:
        s_t += drift( s_t, t ) * dt + diffusion( s_t, t ) * eps_sqrt_dt
        t += dt
    return s_t
```

SDE simulation parameters (variables `s0`, `t_start`, `t_stop`, `N` and lambda expressions `drift`, `diffusion`) are passed to the function `lvm_sde` as global variables.

5. Empirical distribution construction in local volatility models

Empirical pdf for random variable $S(T)$ can be obtained using Monte Carlo method for various local volatility models of the form (1), including the models with theoretical transition probability density functions, determined by (5), (6), (7).

Simulation parameters for SDE (1) are the following:

$$S_0 = 1,\ t_0 = 0,\ t = 1,\ r = 7\%,\ \sigma = 30\%,\ \alpha = \pm 8\%,\ N = 100\,000,\ n = 10\,000. \tag{12}$$

Theoretical probability density functions in local volatility models under consideration are shown in Figure 1.

Lambda expressions for simulation of SDE (1) are initialized as given below:

```
drift = lambda y, t: 0.07 * y
# shifted lognormal model with positive alfa
diffusion = lambda y, t: 0.3 * (y - 0.08 * math.exp(0.07 * t))
# shifted lognormal model with negative alfa
diffusion = lambda y, t: 0.3 * (y + 0.08 * math.exp(0.07 * t))
# Ornstein-Uhlenbeck model
diffusion = lambda y, t: 0.3
# hyperbolic sine model
diffusion = lambda y, t: math.sqrt( 0.14 * y * y + 0.09 )
```

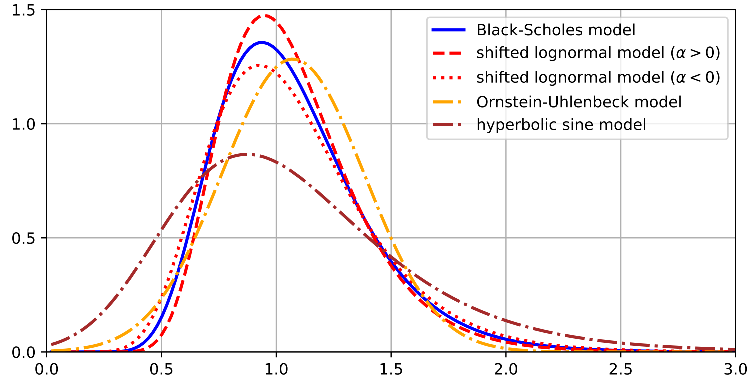

Figure 1: Probability density functions in local volatility models

The simulation of SDE (1) by Monte Carlo method with parameters (12) for different volatility functions confirms (see Figure 2) that empirical probability density functions obtained with the simulation approximate well enough theoretical probability distribution density functions of the underlying asset in local volatility models under consideration.

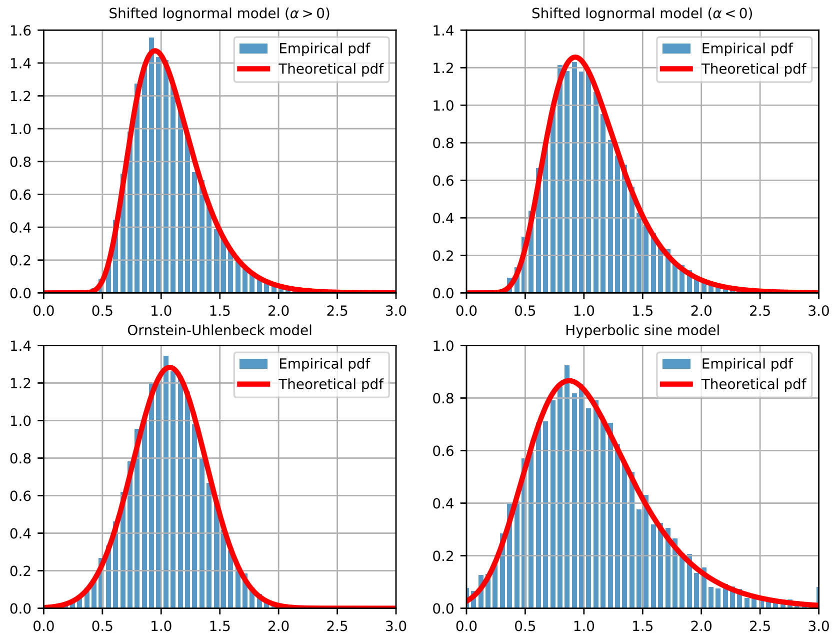

Figure 2: Empirical and theoretical distribution densities in local volatility models

Parallel computing using module `multiprocessing` on MacBook Pro with 8-Core Intel Core i9 processor can significantly accelerate the execution of program code (see Table 1) with almost full load of MacBook Pro i9 processor cores (see Figure 3).

Table 1
Empirical distribution density construction time in sequential and parallel execution

Mode of execution	Runtime (in milliseconds)	Speedup
Sequential execution time	951 512.8	1x
Time in parallel execution	159 308.7	6x

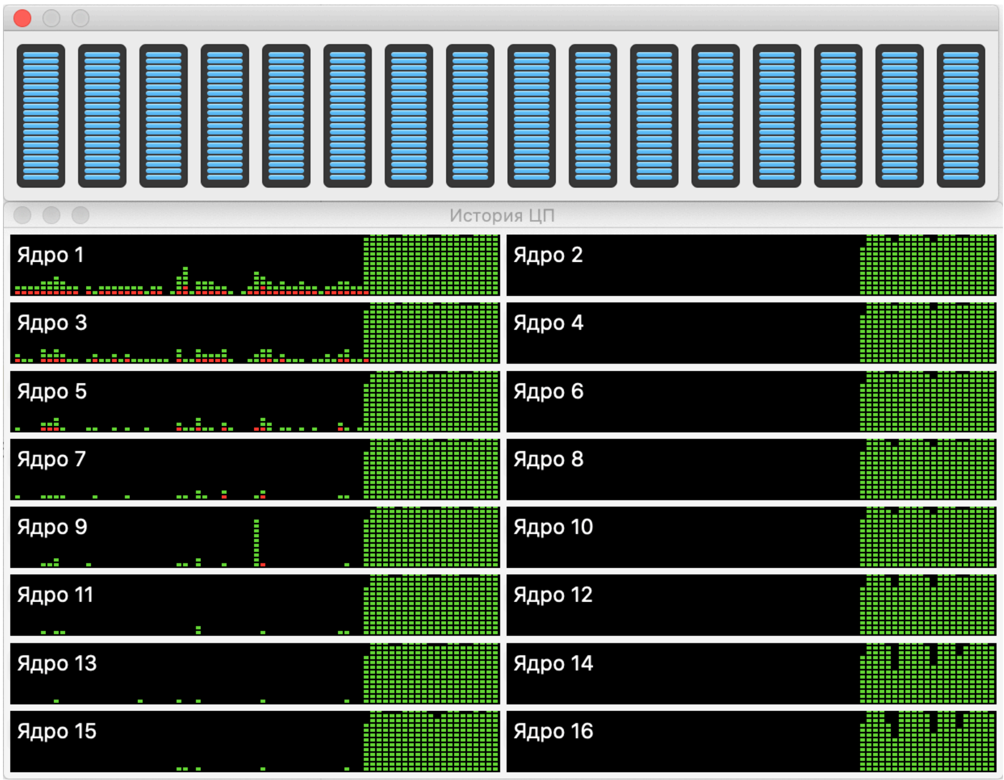

Figure 3: Full load of MacBook Pro with 8-Core Intel Core i9 processor

Further speedup of calculation of empirical distribution density can be gained using GPU resources [15]. When using GPU, program code can be accelerated many times depending on the task being solved. For instance, on MacBook Pro with Intel Core i9 processor and discrete graphics card AMD Radeon Pro 5500M (1536 shader processors) the program code can be accelerated hundreds or even thousands of times compared to sequential execution on CPU.

Parallel GPU computing can be implemented in a Python program using the PyCUDA library (for Nvidia graphics cards) or the PyOpenCL library (for AMD graphics cards) [16]. The idea of using Monte Carlo method on a GPU for financial simulation was studied in a number of papers [17], including applications to financial risk measurement [18]. Calculating multiple trajectories of a multidimensional system of SDE in parallel using OpenCL is a subject of research in [19].

OpenCL framework lacks a standard random number generator, so in order to take advantages of PyOpenCL library to build empirical distributions and evaluate options with a GPU, one has to implement a pseudorandom number generator. The recognized random number generators implemented on GPU are collected in Random123 library [20].

Acknowledgments

The publication has been prepared with the support of the "RUDN University Program 5–100". The research was funded by RFBR, grant No. 19-08-00261.

References

[1] M. A. H. Dempster, J. Kanniainen, J. Keane, E. Vynckier (Eds.), High-Performance Computing in Finance, Chapman and Hall/CRC, 2018. doi:10.1201/9781315372006.

[2] F. Black, M. Scholes, The Pricing of Options and Corporate Liabilities, Journal of Political Economy 81 (1973) 637–654. doi:10.1086/260062.

[3] R. C. Merton, Theory of Rational Option Pricing, The Bell Journal of Economics and Management Science 4 (1973) 141–183. doi:10.2307/3003143.

[4] E. Derman, M. B. Miller, The Volatility Smile, John Wiley & Sons, Inc., 2016. doi:10.1002/9781119289258.

[5] S. T. Rachev, C. Menn, F. J. Fabozzi, Fat-Tailed and Skewed Asset Return Distributions: Implications for Risk Management, Portfolio Selection, and Option Pricing, Wiley, 2005.

[6] J. C. Cox, S. A. Ross, The valuation of options for alternative stochastic processes, Journal of Financial Economics 3 (1976) 145–166. doi:10.1016/0304-405x(76)90023-4.

[7] D. Brigo, F. Mercurio, Fitting volatility skews and smiles with analytical stock-price models, Seminar Paper, Institute of Finance, University of Lugano, 2000.

[8] G. E. Uhlenbeck, L. S. Ornstein, On the theory of the brownian motion, Physical Review 36 (1930) 823–841. doi:10.1103/physrev.36.823.

[9] S. Shorokhov, M. Fomin, Modeling of financial asset prices with hyperbolic-sine stochastic model, in: Convergent Cognitive Information Technologies, Springer International Publishing, 2020, pp. 3–10. doi:10.1007/978-3-030-37436-5_1.

[10] P. Jäckel, Monte Carlo Methods in Finance, The Wiley Finance Series, Wiley Finance, 2002.

[11] G. Maruyama, Continuous markov processes and stochastic equations, Rendiconti del Circolo Matematico di Palermo 4 (1955) 48–90. doi:10.1007/bf02846028.

[12] G. N. Mil'shtejn, Approximate integration of stochastic differential equations, Theory of Probability & Its Applications 19 (1975) 557–562. doi:10.1137/1119062.

[13] P. E. Kloeden, E. Platen, Numerical Solution of Stochastic Differential Equations, Springer Berlin Heidelberg, 1992. doi:10.1007/978-3-662-12616-5.

[14] J. Palach, Parallel Programming with Python, Packt Publishing, 2014.

[15] S. Grauer-Gray, W. Killian, R. Searles, J. Cavazos, Accelerating financial applications on the GPU, in: Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units - GPGPU-6, ACM Press, 2013, pp. 127–136. doi:10.1145/2458523.2458536.

[16] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, A. Fasih, PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation, Parallel Computing 38 (2012) 157–174. doi:10.1016/j.parco.2011.09.001.

[17] L. Xu, G. Ökten, High-performance financial simulation using randomized quasi-monte carlo methods, Quantitative Finance 15 (2015) 1425–1436. doi:10.1080/14697688.2015.1032549.

[18] J. A. Varela, N. Wehn, S. Desmettre, R. Korn, Real-time financial risk measurement of dynamic complex portfolios with python and PyOpenCL, in: Proceedings of the 7th Workshop on Python for High-Performance and Scientific Computing - PyHPC'17, ACM Press, 2017, pp. 1–10. doi:10.1145/3149869.3149872.

[19] E. Avramidis, M. Lalik, O. E. Akman, SODECL: An Open Source Library for Calculating Multiple Orbits of a System of Stochastic Differential Equations in Parallel, arXiv:1908.03869 (2019).

[20] J. K. Salmon, M. A. Moraes, R. O. Dror, D. E. Shaw, Parallel random numbers: as easy as 1, 2, 3, in: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '11, ACM Press, 2011, pp. 1–12. doi:10.1145/2063384.2063405.