Stochastic Process Simulations, With Application To Asian Options

Xiaojian Jin,[*] Lingyi Kong,[†] and J. Sidrach[‡]
University of California, San Diego

(Dated: June 11, 2016)

The random walk hypothesis states that stock market prices cannot be predicted due to the behavior of random walk-like behavior. However, using stochastic process, like Brownian motion, we can model them fairly well for short term data. This paper describes our attempt at using a modified Black-Scholes Model with Markov Chain Monte Carlo simulations, in order to predict stock prices and Asian options.

I. INTRODUCTION

An option is a type of security which gives the owner the right buy (call option) or sell (put option) the underlying asset at a predefined strike price. The one who issues an option, called the writer, must deliver to the buyer a specified number of shares if the latter decides to exercise the option. The buyer pays the writer a premium in exchange for writing the option. The Asian option is an option whose payoff depends on the average price of the underlying asset over a certain period of time as opposed to at maturity. Two types of Asian options are found in the market: average price options and average strike options. Average price options have a fixed strike value and the average used is the asset price. Average strike options have a strike equal to average value of the underlying asset. In this paper, the payoff at maturity of an average price Asian option is:

$$Payoff_{call} = \max(0, S_{avg} - K)$$
$$Payoff_{put} = \max(0, K - S_{avg})$$

where S_{avg} is the average price of underlying asset and K is the strike price. The average can be arithmetic or geometric.

We will use the Black-Scholes Model for our simulation of the options. The Black-Scholes model assumes that the market consists of at least one risky asset, usually called the stock, and one risk-less asset, usually called the money market, cash, or bond. The Black-Scholes Model includes [1] two assets: stock prices and bond prices. We simulate using the model for the stock price and then using it to calculate Asian options. Given some initial stock price, the stock price in some near future follows the stochastic equation below:

$$S_t = S_0 e^{(\mu - \frac{1}{2}\sigma^2)t + \sigma W_t}, t \in [0, T] \quad (1)$$

where μ is also called the "drift rate" and σ is the "volatility". We'll show shortly that Eq 1 is simply the solution to a specific continuous-time stochastic differential equation (SDE) which is commonly known as "geometric Brownian motion" (GBM).

A. Geometric Brownian Motion (GBM)

Brownian motion was first discovered a botanist by Robert Brown in 1827, but it has gained a lot of attentions from various fields, including mathematical finance. It is more commonly known as the Wienner process in mathematics. GBM is a special case of Brownian motion which satisfies the stochastic differential equation below:

$$dS_t = \mu S_t dt + \sigma S_t dW_t \quad (2)$$

where W_t is the Wienner process given by $W_t \sim N(0, \sigma^2 t)$. As mentioned previously, Wienner process is a continuous-time stachastic process. Although it's continuous, it's not differentiable. In the next section, we'll discuss how to solve such SDE using Ito process and Ito Calculus.

B. Ito Calculus

Ito Calculus can be viewed as extended calculus which handles continuous-time stochastic differential equation. As we have previously mentioned, the nature of Brownian motion makes it continuous in time but not differentiable. The notion of derivatives no longer apply to SDE. Instead, Ito Calculus introduces the notion of differentials, which is essentially finite difference, given as:

$$dW(t + dt) = W(t + dt) - W(t)$$

Therefore, the notion of integral with random variables can also be defined using the same notion as Riemann sum

$$\int_0^t X(\tau) dW(\tau) = \lim_{n\to\infty} \sum_{k=0}^{t-1} X_k (W_{k+1} - W_k) \quad (3)$$

We should also notice that Eq 3 also gives us a way to approximate the integral using discretized summation. Now, with Ito Calculus's notion of differentials and integral, we obtain the integral form of the SDE

$$S(t) = S(0) + \int_0^t \mu(S(\tau), \tau) d\tau + \int_0^t \sigma(S(\tau), \tau) dW_\tau \quad (4)$$

The following properties are results of Ito calculus, which we simply assumed true:

[*] xij049@ucsd.edu
[†] l1kong@ucsd.edu
[‡] jsidrach@ucsd.edu

1. $(dt)^2 = 0$
2. $(dt)(dW(t)) = 0$
3. $(dW(t))^2 = dt$

Ito's formula, which can be viewed as a result of Taylor expansion up to second-degree term, shows that for

$$dX(t) = \mu(t)dt + \sigma(t)dW(t)$$

$$\begin{aligned} df(W(t),t)) = \{ &\frac{\partial f(X(t),t)}{\partial t} + \mu(X(t),t)\frac{\partial f(X(t),t)}{\partial X} \\ &+ \frac{1}{2}\sigma^2(X(t),t)\frac{\partial^2 f(X(t),t)}{\partial X^2}\}dt \end{aligned} \tag{5}$$

Letting $\mu(X(t),t) = \mu S(t,t)$, $\sigma(S(t),t) = \sigma S(W(t),t)$, $S(W(t),t) = \ln(W(t))$ and plugging in Eq 2, it follows from Eq 5 that

$$dS(W(t),t) = (\mu - \frac{1}{2}\sigma^2)dt + \sigma dW(t) \tag{6}$$

This is exactly the differential form for Eq 1.

C. Wiener Process and Random Walk

The stochastic equation we took from Black Scholes model depends both on time and a random variable $\sigma W(t)$, which conforms to the Wiener Process. Because Wiener process is continuous in time, we need to discretize it for numerical simulation. It has been proven that a random walk with infinitestimal step size converges to Wiener process as number of steps goes to infinity. We take the step size of a random walk to be Δx with a probability p in a time interval Δt, then the expected step length and variance are given as

$$E[X(t+\Delta t) - X(t)] = (2p-1)\Delta x$$

$$E[(X(t+\Delta t) - X(t))^2] = (\Delta x)^2$$

Since we also know that in Wiener process, transition process conforms to normal distribution $\sim N(0,\sigma^2 t)$, equating the expected value and variance, we can solve that

$$\Delta x = \sigma\sqrt{\Delta t} \tag{7}$$

Hence, we have a stochastic process with a Gaussian transition probability.

II. METHOD

A. Parameters

The choice of drift and volatility parameter are of the freedom of programmer. In our simulation, we define these two parameters as risk-free rate and standard deviation in stock price. The value of these two parameters are extracted using historical data since we assume the expected value and variance are stable for short term predictions.

B. Metropolis-adjusted Langevin Algorithm

When running Markov chain, one needs to define an update rule between current and proposed next state. Metropolis Hasting Algorithm provides a general way of calculating update rule if the relative probability between two states is known. The acceptance rule is then given as

$$A(x'|x) = \max\left(1, \frac{P(x')}{P(x)}\frac{g(x'|x)}{g(x'|x)}\right) \tag{8}$$

Where $P(x)$ is the probability of state x, $g(x'|x)$ is the true transition probability density, and we accept proposed state with probability $A(x'|x)$.

The SDE black scholes equation satisfies is the same SDE for 1-d Langevin diffusion process, then using Metropolis-adjusted Langevin Algorithm (MALA) [2], the acceptance rule can be calculated as

$$A(x'|x) = \max\left(1, \frac{\pi(x')q(x|x')}{\pi(x)q(x'|x)}\right) \tag{9}$$

With $\pi(x')$ being the probability density of the state and $q(x'|x')$ the transition probability density. For Wiener process

$$\pi(W_t) \sim N(0,\sigma^2 t)$$

$$q(W_t|W_s) = q(W_s|W_t) \propto e^{\frac{W_t - W_s}{2\sigma^2 t}}$$

Therefore the transition probability density term cancels out which gives simply

$$A(x|x') = \frac{\pi(x')}{\pi(x)} \tag{10}$$

Variation

The stock price is log-Gaussian due to the Brownian term in our model, where the evolution of the stock follows some random walk. Using Black-Scholes model as a starting point, we made some variation to the model and hope to see if running MALA will give us something that's a closer approximation to the true evolution of the stock process. Since we discretized time in the model, the random variable can be viewed as an N-dimension vector where N is the number of discretization. Our variation, instead of using one random walk of N steps as the path for calculating stock price, performs random N random walks and took the end step from each walk as the path for the model. Hence instead of running a Markov Chain along the path, we are running Markov Chain between paths.

III. IMPLEMENTATION

We implemented our simulations using an iterative approach. First we built a toy model, and once we made sure its results were convincing, we estimated the parameters for the final model and implemented it.

A. Toy model

Our approach was to build a toy model using an IPython notebook. The advantages of using an IPython notebook instead of MATLAB as we were used to before are significant. It is free, so every member of the team could use it on their personal computers. It is also faster to develop for, and easier to translate to C code later on the project. On the other hand it comes with worse performance and it is harder to automatize since we do not generate a single binary, but these two are not concerns in the context of a toy model.

The first toy model we built was a simple Monte Carlo simulation of the stock prices as random walks, without using Metropolis algorithm. This model was really simple but we were not convinced by the results of it, so we ditched it in favor of a Metropolis algorithm of a Monte Carlo Markov Chain.

This last toy model already successes at simulating the final distribution of the prices (expected to be log normal), as seen in Figure 1. The code is available at `src/toy-model-asian-options.ipynb`, and a HTML version in case IPython is not installed can be found at `docs/noteboks/toy-model-asian-options.html`.

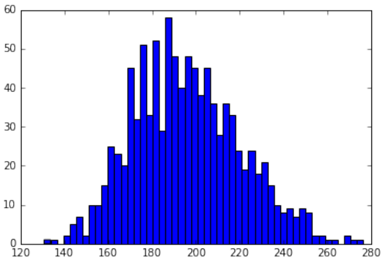


FIG. 1. Histogram of the final prices for the toy model

B. Parameter estimation

In order to test our model and code, we test it with real stock prices data. We selected three companies from NASDAQ-100 index: Baidu, Facebook and Yandex. The data we used can be found online at Yahoo Finance in CSV format, and in the `data/historical` folder in our project. We restricted the data to 300 days, and calculated the initial stock price, μ and σ from the data in a IPython notebok. The code is available at `src/estimate-parameters.ipynb`, and its HTML counterpart can be found at `docs/noteboks/estimate-parameters.html`.

The estimated parameters are:

Company / Parameter	Initial price	σ	μ
Baidu	207.33	-0.0004196	0.02143578
Facebook	83.30	0.00135278	0.00401862
Yandex	15.45	-7.9067e-05	0.03078405

There parameters were hardcoded into the CUDA simulation at compile time, using the `Makefile`.

C. CUDA

The CUDA implementation allowed us to simulate in parallel each path (Markov Chain). Without parallelization, the usual way to generate the paths would be to simulate a single Markov Chain, and take M samples (one every n steps) after the warmup, assuming that n is large enough to make the sampled paths memoryless. Thanks to the parallelization, we can simulate M independent paths from the beginning, and after the warmup iterations just take one sample from every independent path. Not only it is faster but we do not need to make the assumption about the memory we made in the previous implementation.

However, translating the original implementation into CUDA came with some drawbacks due to the nature of the simulation. First, since we are running the Metropolis algorithm we need to store in each thread the previous path, which is innefficient as memory communication in GPUs is slower than actual computation. Second, we could not use the standard random number generator as it was too slow and we had no guarantees that every thread generated a pseudo-random number independently. We use the CUDA library for random number generation to solve this. Finally, the output files were too large to deal with them in the IPython notebook directly. We solved this by automatically compressing the output and decompressing it in IPython once it has been loaded into the notebook.

All in all, because the extensive amounts of memory needed in every thread to compute the path simulation and the communication overhead this introduces, CUDA was not as fast as expected, but still orders of magnitude faster than the Python toy model implementation. The complete simulation for one company (640 threads or final paths with 1000 warmup iterations each one, and 3000 time discretization steps in each path) runs in about one minute.

The CUDA source code can be found in `src/asian-options-main.cu` (main function), `src/asian-options.cu` (simulation functions) and `src/asian-options.h` (definitions).

IV. RESULTS

The following table summarizes the simulation results in CUDA for the final price of the stock price, and the payoff obtained. We measure the error using the pricing error, given by:

$$\frac{e^{-rT}}{\sqrt{N-1}}\sqrt{\frac{1}{N}\sum_{i=1}^{N}H^i(T)^2-\left(\frac{1}{N}\sum_{i=1}^{N}H^i(T)\right)^2}$$

where H^i is the payoff of the i^{th} path, N is the total number of paths, r is the risk-free rate and T is the expired time. The pricing error seems acceptable in the last two

Company	Sim. Final Price	Payoff Sim.	Pricing Err.
Baidu	202.05	53.57	0.16
Facebook	83.15	28.19	0.02
Yandex	14.88	0.20	0.009

companies.

Figures 2 and 3 sum up graphically the results of our simulations for one of the companies (Baidu). As we can see, the histogram of the final prices correspond to a log normal distribution - as expected. The full results, statistics and error measurements can be found in `src/simulations-cuda.ipynb` (and `docs/notebooks/simulations-cuda.html`)

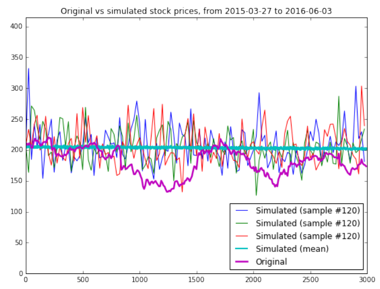


FIG. 2. Simulated paths (Baidu)

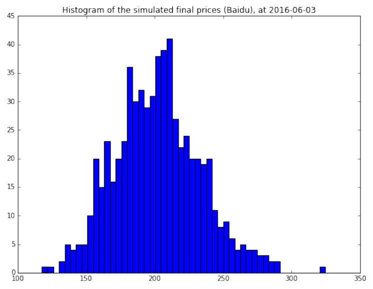
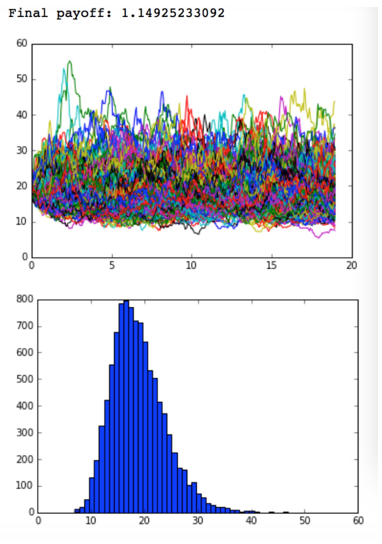


FIG. 3. Histogram of the simulated final prices (Baidu)

Additionally, we made a conceptual change in the Metropolis method in order to compare with our final model. In the original model, the proposed new point S_t^i is compared with the point S_t^{i-1} in the previous path. Instead, now we compare the proposed point S_t^i with the point S_t^{i-1} in the previous time of the same path.

Figure 4 shows the result of this modified method:



```
Final payoff: 1.14925233092
```

FIG. 4. Simulated paths and histogram of the final prices of the modified Metropolis method

This "variation" method looks similar to our model, while conceptually different it ends up converging to almost the same result.

V. CONCLUSIONS

We simulated stock prices and options using Monte Carlo methods, and over the course of the project implemented and compared different models, the last one being the one which yielded the best result and the one we tried to explain in detail in this report.

This model works best for the stock evolution in the near future, fitting historic data accurately in the short term, and can be used to successfully predict the pricing of Asian options. However, long terms predictions have been empirically proved not to be accurate enough, and this model also fails to incorporate external but important factors that play a role into the stock price, such as general market trends or economic crisis.

Overall, we are satisfied with the results obtained given the time limitation, and we think it is a good starting point to further refine and develop stock pricing Monte Carlo based models.

[1] R.J.Williams. American Mathematical Society, 2006.
[2] T. Xifara, C. Sherlock, S. Livingstone, S. Byrne, and M. Girolami. Langevin diffusions and the metropolis-adjusted langevin algorithm. 2014.