Utah State University

DigitalCommons@USU

All Graduate Plan B and other Reports | Graduate Studies

5-2020

Implementing Option Pricing Model

Zhao Ming
Utah State University

Follow this and additional works at: https://digitalcommons.usu.edu/gradreports

Part of the Finance and Financial Management Commons

Recommended Citation

Ming, Zhao, "Implementing Option Pricing Model" (2020). *All Graduate Plan B and other Reports*. 1430.
https://digitalcommons.usu.edu/gradreports/1430

This Report is brought to you for free and open access by the Graduate Studies at DigitalCommons@USU. It has been accepted for inclusion in All Graduate Plan B and other Reports by an authorized administrator of DigitalCommons@USU. For more information, please contact digitalcommons@usu.edu.

UtahStateUniversity
MERRILL-CAZIER LIBRARY

Implementing Option Pricing Model

Zhao Ming *Utah State University*

In this paper I replicate Clewlow and Strickland's control variates methods based on Greek letters method to test if it can improve the simulation efficiency. First, I use Black Scholes Merton formula for option pricing as a benchmark, to compare with the Euorpean call option price from Monte Carlo methods. Then I use Greek letters as control variates to reduce sample standard deviation and improve the efficiency of the Monte Carlo simulation. The whole process is programming in C++. C++ is a compiled language which can generate machine code from source code and provide a shorter running time. This paper is based on ideas from *Implementing Derivatives Models* by Clewlow and Strickland, *On The Simulation of Contingent Claims* by Clewlow and Carverhill, Chapter 4, and *Derivatives Markets* by McDonald, Chapter 12, 13, 19.

Keywords: Monte Carlo, control variates, option pricing

Introduction

Black-Scholes Merton formula is used to compute the theoretical price of the European call option for a stock under certain assumptions. Monte Carlo method is based on random sampling to generate the numerical results. We chose to use Black-Scholes Merton formula as a benchmark to find the option price, then simulate the option price using Monte Carlo method, comparing with the two results to evaluate if Monte Carlo method could provide a correct option price. Next, we combine Greek letters with Monte Carlo method, calculate the standard errors from both Delta-hedging Monte Carlo, Delta-Gamma hedging Monte Carlo and naive Monte Carlo to test how control variates reduce the standard deviation and improve the efficiency of Monte Carlo simulation. At the end, we switch from Black-Scholes Merton's world to Jump world which imitates the real world and test control variates method.

Section 1 Control Variates

The control variates method is a variance reduction technique that is used in the Monte Carlo simulation algorithms. It can help to reduce the standard error of an estimate of an unknown quantity, and improve the efficiency of the simulation.

Clewlow and Carverhill (1995) mention in their paper, “the control variate is a known value that is closely related to the unknown quantity to be simulated. A good control variate for an option problem might be the price of a similar type of option for which an analytic valuation equation exists. The simulation will estimate the difference between the unknown value and the control variate, and this gives a greater accuracy than the unknown value itself. The control variates are intimately related to the hedging of the option. The control Variates are not only improving efficiency but can also providing insights into the hedging of the option” (ClewlowStrickland).

McDonald (2013) presents an example of pricing an arithmetic Asian call option. He suggests using a geometric Asian option as the control variate. There is a convenient closed-form solution that is a special case of the Black-Scholes Merton formula for options with a geometric mean payoff function. He combines this with a Monte Carlo estimate of the known geometric Asian option, and thereby can compute an estimated sampling error along each path. McDonald reports significant variance reduction in the Monte Carlo sampling routine for the arithmetic Asian option.

The underlying method is to estimate the error of each trial by using the price of a related option that does have a pricing formula. The error estimate obtain from this control price can be used to improve the accuracy of the Monte Carlo price on each trial.

Because we are using the same random numbers to estimate the option, the errors in the estimated arithmetic and geometric prices are correlated. When the estimated price for the geometric option is high, it should have a high arithmetic price as well. We can use information on the error in the geometric price to adjust our estimate of the arithmetic price.

$$A^* = \bar{A} + \beta(G - \bar{G})$$

In formula above, we estimate both arithmetic price, $\bar{A}$ and geometric price, $\bar{G}$. And “A” and “G” represent the true arithmetic and geometric prices. The error term from the Monte Carlo estimates the geometric price is $(G - \bar{G})$. We want to use this error to improve our estimate of the arithmetic price.

We could find an optimized β to help better estimate A^*

As Clewlow and Strickland (1998) point out, looking to hedging instruments that reduce the variance of overall portfolio cash-flow variance turns out to be an excellent strategy for identifying

Results Summary

Characteristics	Price	Standard Error	Replication
Black-Scholes Merton	6.96		
Navie Monte Carlo	6.13	0.8965	100
	7.02	0.3127	1000
	6.93	0.0982	10000
	6.92	0.0309	100000
Delta Control Variate MC	6.97	0.0497	100
	6.94	0.0151	1000
	6.94	0.0048	10000
	6.94	0.0014	100000
Delta-Gamma Control Variate MC	6.85	0.0419	100
	6.81	0.0123	1000
	6.93	0.0040	10000
	6.93	0.0012	100000

Figure 1: This is the result by comparing Blach-Schoels Merton option price with Monte Carlo methods.

valid control variates. As such, they employ the Delta and Gamma from the well-known Black-Scholes Merton model. Considering when they sold a call option, when the stock price increases, they would lose lots of money, because the customer would execute the call option. The Market-maker will have a large standard deviation in their whole portfolio. Instead, Market-makers could buy Delta shares when they sold a call option, if the stock price went up, they could sell those shares to customers when they bought at a lower price. If the stock price decreases, the customers would not execute the option, so Market-makers could keep the premium. When the Market-makers applied Delta and Gamma for control variates, they will reduce their portfolio's sample variance.

Section 2 Black-Scholes Merton model

This is Black-Scholes Merton formula for a European call option.

$$C(S,K,\sigma,r,T,\delta) = Se^{-\delta T}N(d_1) - Ke^{-rT}N(d_2)$$

$$d_1 = \frac{\ln{(S/K)} + (r - \delta + \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}}$$

$$d_2 = d_1 - \sigma\sqrt{T}$$

In Black-Scholes Merton formula, we can interpret Delta as a share-equivalent. Black-Scholes Merton formula tells us both option price and what position in the stock is equivalent to the option.

McDonald (2013) points out "Black-Scholes Merton formula is constrained by certain economic environment, the risk-free rate is known and constant, there are no transaction costs, and it is possible to short-sell costlessly and to borrow at the risk-free rate" (McDonald).

Monte Carlo simulation is to obtain the option price for models that do not have analytical formulas; it can also simulate asset path with stochastic volatility.

```cpp
double blackScholesCall(const double& spot, const double& strike, const double& vol
                        , const double& rate,const double& expiry, const double& div)
{
    auto d1 = (std::log(spot / strike) + (rate - div + 0.5 * vol * vol) * expiry)
            / (vol * std::sqrt(expiry));
    auto d2 = d1 - (vol * std::sqrt(expiry));
    auto callPrc = (spot * std::exp(-div * expiry) * normalCDF(d1))
                  - (strike * std::exp(-rate * expiry) * normalCDF(d2));

    return callPrc;
}
```

In the code above, we define a function named "blackScholesCall", and we pass information as parameters. Parameters are specified inside the parentheses, and you can add as many as you want, like here, we include all the variables from the Black-Scholes Merton formula, and specify variables' types before using them.

$N(d_1)$ and $N(d_2)$ are matching with normalCDF(d1) and normalCDF(d2), which stands the cumulative normal distribution function, showing the probability that a number randomly drawn from a standard normal distribution will be less than d_1 and d_2.

From the "blackScholesCall" function, we could change the values of parameters that are passing to the function and generate different prices of call option.

Section 3 Greeks with purposes of heding Versus Greeks for control variates

Before talking about the Greeks, we want to mention what are market-makers. Market-makers are the people or agents that are trading with the customers. Market-makers maintain inventories to satisfy the customers immediately. Market-makers make profits from bid-ask spread. Also, market-makers could short-sell by borrowing other people's shares and then selling those shares to generate inventory as needed.

Under unhedged situation, Market-makers will face the risk from the adverse price movement. At the worst situation, unhedged position will lead to bankruptcy.

Market-makers could use Delta-hedging to control risk. Market-makers could compute the Delta value and buy Delta number of shares to offset risk, market-makers need to invest capital to maintain a Delta-hedging position because the cost of option is different from the cost of stock.

Delta measures the option price changes when stock price changes, also Delta represents number of shares in the portfolio that replicates the option.

$$CallDelta = \frac{\partial C(S, K, \sigma, r, T-t, \delta)}{\partial S} = e^{-\delta(T-t)} N(d_1)$$

```
double BlackScholesDelta(const double& spot, const double& t, const double& strike,
    const double& expiry, const double& vol, const double& rate, const double& div)
{
    auto tau = expiry - t;
    auto d1 = (std::log(spot / strike) + (rate - div + 0.5 * vol * vol) * tau)
              / (vol * std::sqrt(tau));
    auto delta = std::exp(-div * tau) * normalCDF(d1);

    return delta;
}
```

For a call option, Delta is positive. When the stock price increases, the call option price will increase. The Delta will approach 1 when the stock price is deep in the money. It shows the share-equivalent of the option is 1.

Gamma shows the rate change of Delta when stock price changes by 1 dolloar.

$$CallGamma = \frac{\partial^2 C(S, K, \sigma, r, T-t, \delta)}{\partial S^2} = \frac{e^{-\delta(T-t)} N'(d_1)}{S\sigma\sqrt{T-t}}$$

```cpp
double BlackScholesGamma(const double& spot, const double& t, const double& strike,
    const double& expiry, const double& vol, const double& rate, const double& div)
{
    auto tau = expiry - t;
    auto d1 = (std::log(spot / strike) + (rate - div + 0.5 * vol * vol) * tau)
              / (vol * std::sqrt(tau));
    auto gamma = (std::exp(-div * tau) * normpdf(d1) / (spot * vol * std::sqrt(tau)));

    return gamma;
}
```

Delta alone is not predating the option price accurately because Delta changes with the stock price. Delta will underestimate the actual change when underlying asset price increases and overestimate the decline when the stock price decreases. We add Gamma to improve it.

“The changes in the Delta and Gamma as the stock price evolves randomly exactly offset the changes in the option value due to the random changes in the stock price” (ClewlowStrickland), delta and gamma provide good results for hedging purposes, this is the reason why authors choose Delta and Gamma for control variates in a numerical way.

$$C(S_{t+h}) = C(S_t) + \epsilon\Delta(S_t) + \frac{1}{2}\epsilon^2\Gamma(S_t))$$

The Delta approximation is a straight-line tangent to the option price curve and is always below the option price curve. The Delta-Gamma approximation uses the squared stock price

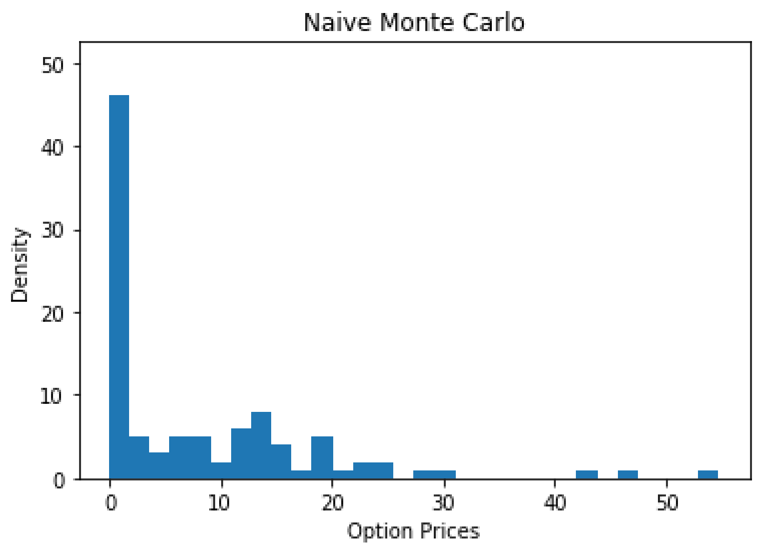

Figure 2: This is the call payoff from Monte Carlo simulation, used 252 time steps, and 100 simulations.

change, which generates a curve more closely approximating the option price curve. Using hedges as control variates was first described by Clewlow and Carverhill. When pricing a European call option, the distribution of the pay-off has a large standard deviation. If we want to estimate the call value as the mean of a number of Monte Carlo simulations then standard error of the mean will be large. When using Delta control variate, the pay-off of the hedged portfolio has a much smaller standard deviation.

$$C_{t_0}e^{r(T-t_0)} = C_T - [\sum_{i=0}^{N-1} \frac{\partial C_{t_i}}{\partial S}(S_{t_i+1} - E[S_{t_i}])e^{r(T-t_{i+1})}] + \eta$$

Section 4 Variance Reduction

The main purpose for the Delta-hedging is to help market-makers to maintain a Delta-neutral position, market-makers sell one call option and hedges the position with shares.

And now from Clewlow and Carverhill, we are going to see how they using Greeks as control variates to reduce the standard deviation and improve the efficiency.

From the graph above, we could easily find after using Delta as control variate, the result of call payoff has an obvious improvement. The sample has a smaller standard deviation, and after using Delta-Gamma as control variates, the standard deviation even gets smaller. If you only

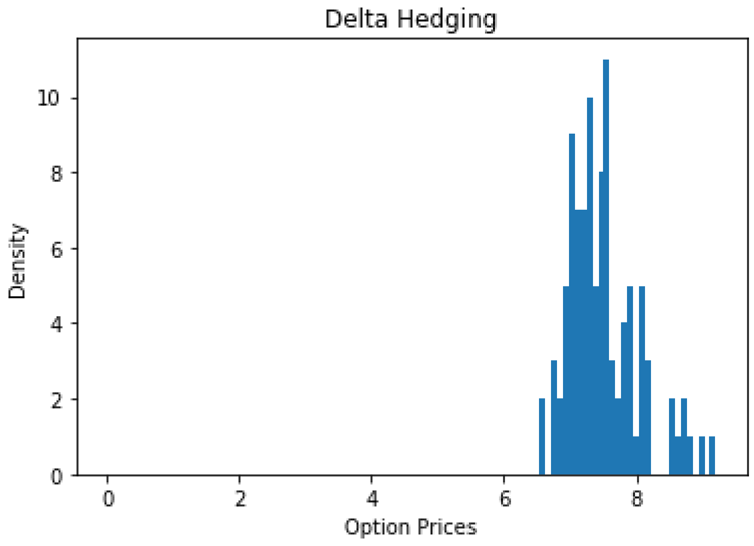

Figure 3: This is the call payoff from Delta-Hedged Monte Carlo simulation, used 252 time steps, and 100 simulations.

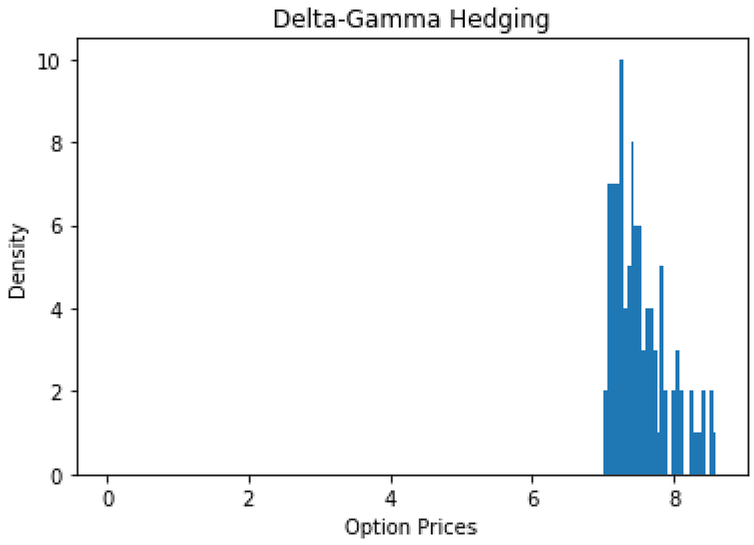

Figure 4: This is the call payoff from Delta-Hedged Monte Carlo simulation, used 252 time steps, and 100 simulations.

using Monte Carlo simulation without control variates, you will need a 100 times larger sample size to get the standard deviation as same as the Delta-hedging Monte Carlo.

We use the Monte Carlo method to simulate the discrete time asset paths for the underlying asset.

$$S_{t+h} = S_t e^{\{(r-\delta-\frac{1}{2}\sigma^2)h+\sigma\sqrt{h}z\}}$$

```
std::vector<double> simulateNvPath(const double& spot, const double& rate, const double& vol,
                                   const double& div, const double& expiry, const size_t& numSte
{
    auto dt = expiry / numSteps;
    auto nudt = (rate - div - 0.5 * vol * vol) * dt;
    auto sigdt = vol * std::sqrt(dt);
    auto z = std::vector<double>(numSteps, 0.0);
    auto path = std::vector<double>(numSteps, spot);
    RandomNormals(z);

    for (auto i = 1; i < numSteps; ++i)
    {
        path[i] = path[i - 1] * std::exp(nudt + sigdt * z[i]);
    }

    return path;
}
```

In our code, we used “dt” to represent each time steps instead of “h” in the formula to represent the discrete intervals; “z” is represented the generated random numbers which are normally distributed, and help with simulating the random stock price for each time step. And we used “path” to indicate the spot price. After that, we called the call payoff function to get the payoff for each simulation.

```cpp
auto callPayoff(const double& spot, const double& strike)
{
    return std::max(spot - strike, 0.0);
}
```

$$\hat{C}_0 = exp(-rT)\frac{1}{M}\sum_{j=1}^{M} max(0, S_{T,j} - K)$$

We sum the pay-offs of each simulation, average them, then discounted the average back to the present value. This is similar to the Law of large numbers theory; the expected option price of some random payoff can be approximated by taking the average of the random variables.

Subsection A Delta control variate

$$cv_1 = \sum_{i=0}^{N-1} \frac{\partial C_{t_i}}{\partial S}(S_{t_{i+1}} - E[S_{t_i}])e^{r(T-t_{i+1})}$$

```cpp
for (auto i = 1; i < numSteps; ++i)
    {
        path[i] = path[i - 1] * std::exp(nudt + sigdt * z[i]);
        auto t = i * dt;
        delta[i] = BlackScholesDelta(path[i - 1], t, strike, expiry, vol, rate, div);
        cv1 += delta[i] * (path[i] - path[i - 1] * erddt);
    }


for (auto i = 0; i < numReps; ++i)
    {
        auto result = simulateDeltaHedgedPath(spot, rate, vol, div, expiry, numSteps, strike);
        auto spotT = std::get<0>(result);
        auto cv1 = std::get<1>(result);
        callPrice += callPayoff(spotT, strike) + beta1 * cv1;
    }
```

From the first "for loop" in the code above, we calculate the Delta for each time step from the Black-Scholes Delta formula. Delta will rebalance discretely for each step for every simulation. We will get a Delta for each time step. And then using the Delta to get the value of control variates. The authors choose the beta1 = -1 which is the appropriate value for this example where they have the exact Delta. The variable "erddt" allow us to compute $E[S_{t_{i+1}}]$ efficiently.

Subsection B Delta-Gamma control variates

$$cv_2 = \sum_{i=0}^{N-1} \frac{\partial^2 C_{t_i}}{\partial S^2}((\Delta S_{t_i})^2 - E[(\Delta S_{t_i})^2])e^{r(T-t_{i+1})}$$

```
for (auto i = 1; i < numSteps; ++i)
    {
        path[i] = path[i - 1] * std::exp(nudt + sigdt * z[i]);
        auto t = i * dt;
        delta[i] = BlackScholesDelta(path[i - 1], t, strike, expiry, vol, rate, div);
        gamma[i] = BlackScholesGamma(path[i - 1], t, strike, expiry, vol, rate, div);
        cv1 += delta[i] * (path[i] - path[i - 1] * errdt);
        cv2 += gamma[i] * ((path[i] - path[i - 1]) * (path[i] - path[i - 1])
                      - path[i - 1] * path[i - 1] * egamma);
    }


for (auto i = 0; i < numReps; ++i)
    {
        auto result = simulateDeltaGammaHedgedPath(spot, rate, vol, div, expiry, numSteps, strik
        auto spotT = std::get<0>(result);
        auto cv1 = std::get<1>(result);
        auto cv2 = std::get<2>(result);
        callPrice += callPayoff(spotT, strike) + beta1 * cv1 + beta2 * cv2;
```

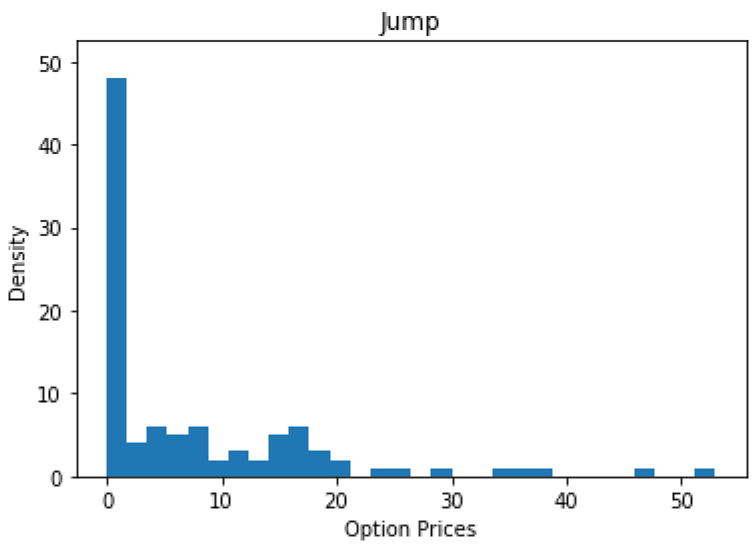

Figure 5: This is the call payoff from the Jump world, used 252 time steps, and 100 simulations.

```
    }
```

The code above is for Delta-Gamma control variates, and we add cv_2 for the control variate of Gamma.

Section 5 Jump World

Because the unpredictable future, sometimes the stock markets will move more than we expected. Therefore, we add the Poisson distribution to simulate the Jump world which is closer to realistic. The Poisson distribution is a discrete probability distribution that counts the number of events - such as large stock price moves - that occur over a period of time. The Poisson distribution is summarized by the parameter λ, where λh is the probability that one event occurs over the short interval "h".

$$\hat{S}_{t+h} = \hat{S}_t e^{(\alpha-\delta-\lambda k-0.5\sigma^2)h+\sigma\sqrt{h}Z} e^{m(\alpha_j-0.5\sigma_j^2)+\sigma_j \sum_{i=0}^{m} W_i}$$

Those pictures are the results from the Jump world, and we could see control variates did a great job to reduce the variance of the sample.

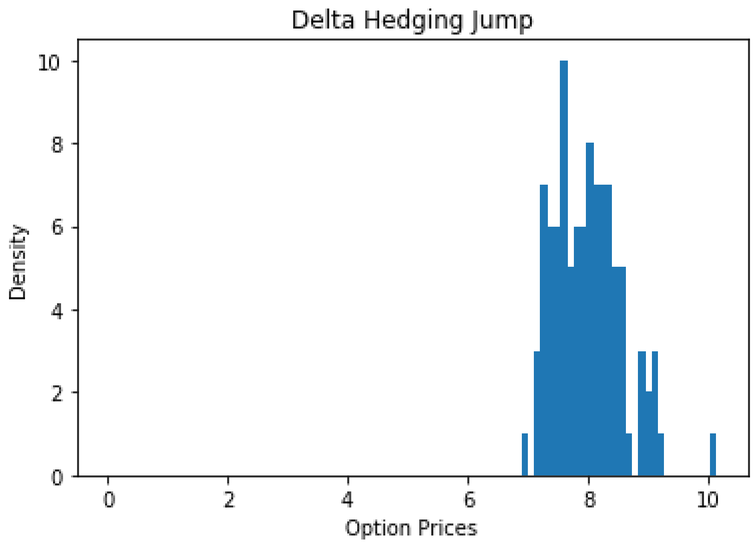

Figure 6: This is the call payoff from Jump world with Delta-hedging, used 252 time steps, and 100 simulations.

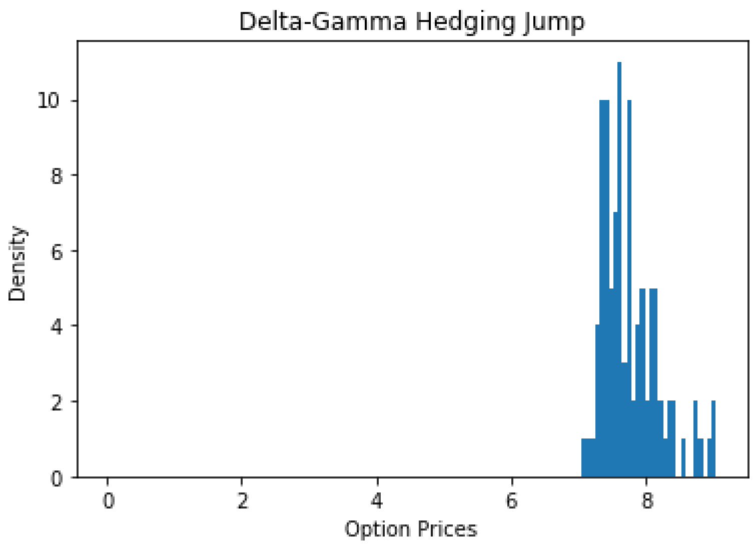

Figure 7: This is the call payoff from Jump world with Delta-Gamma-hedging, used 252 time steps, and 100 simulations.

Conclusion

Clewlow and Carverhill bring out the idea that using control variates to reduce the standard error, also improved the efficiency of Monte Carlo simulation, combined both perspectives from Finance and Computer Science. My project is only replicated the process for European call option, but control variates also good for more complicated path dependent options, like Asian option, look-back option. In the future, I am going to combine stochastic volatility to test the real market data.

References

Clewlow and Carverhill. 1995. "On the Simulation of Contingent Claims." *The Journal of Derivatives* 2(2):66–74.

Clewlow and Strickland. 1998. *Implementing Derivatives Models*. Wiley.

McDonald, Robert L. 2013. *Derivatives Markets*. Pearson.