Monte-Carlo Simulation and Application to Financial Option Pricing

Author: Will Noone

Abstract

This paper describes the valuing financial derivatives through Monte Carlo method. Part I will outline basic concepts of Monte Carlo Simulation and its application to financial option pricing. Part II includes a discussion about Monte Carlo enhancements such as: antithetic variates, control variates, quasi-Monte Carlo methods, generating correlated random variables for multiple underlying or stochastic factors via Cholesky and Eigenvalue decomposition, and calculating path-wise Greeks. Part III provides an example implementing the Monte Carlo simulation method to value European Call and Put options and demonstrate pricing and sensitivity convergence upon Black-Scholes estimates. Finally, we will gauge the improvement to the simulation when the antithetic variate is included.

Part I
Monte Carlo Simulation: Basic Concepts and Application to Financial Options

A. Monte Carlo Simulation (MCS)

Monte Carlo simulation samples the probability distribution for a set of variables to produce hundreds or thousands of likely outcomes (e.g. path dependent options) which is an intuitive way to value options. The results are analyzed to infer the probabilities of each outcome. For example, a comparison model run using traditional "what if" scenarios, and then run again with Monte Carlo simulation and triangular probability distributions will show that the Monte Carlo analysis has a narrower range than the "what if" analysis because that analysis gives equal weight to all scenarios, while Monte Carlo method hardly samples in the very low probability regions. The samples in such regions are uncommon events[1].

There are five main aspects for the Monte Carlo Simulation methodology[2] :

1. Simulating the risk neutral dynamics to the relevant stochastic differential equation.
2. Calculate an option payoff taking into account any path dependency for each realization.
3. Perform a large number of simulations over the specified time horizon.
4. Calculate an expected value.
5. Discount the expected value back to time zero to get the option value.

Monte Carlo Simulation can be applied to derivative pricing, notably to more complex varieties of path dependent options and those on multiple assets. A critical assumption of the Black Scholes model (BSM) is that the fair value for an option is the present value of the option payoff at expirations under a risk-neutral random walk for the underlying asset price.[3] Generally, the procedure to achieve this involves simulating sample paths of the asset price of the option time horizon, then calculate and average the discounted cash flows over the sample paths.

General application to financial option valuation can be outlined as such:

1. Discretize time of the life of the option into M time steps (Δt = T/M).
2. For each time step generate a standard normal random variable (ε).
3. One can then map a unique randomized asset path as such:

$$\Delta S \equiv S_{t+\Delta t} - S_t$$

$$S_{t+\Delta t} = S_t + [r - \delta]S_t\,\Delta t + \sigma S_t dW_t$$

where:

$$dW_t = \sqrt{\Delta t}\,\varepsilon_s$$

[1] Wikipedia: http://en.wikipedia.org/wiki/Monte_Carlo_method
[2] Young, Stephen D., Financial Engineering Notes, p. 136
[3] Charnes, John M., "Using Simulation for Option Pricing", p 152

4. Apply M draws of the standard normal random variable (ε), one future asset path has been generated to result in a potential S_T.
5. From this one can then find a corresponding Call max(S_T – X,0) or Put max(X – S_T,0) for that particular asset path and find the average discounted option value.

The Monte Carlo approach can best be demonstrated with a European call option.[4] First, assume continuous time and that the underlying asset follows a Geometric Brownian Motion (GBM) process:

$$dS = rSdt + \sigma SdZ$$

Next, risk-neutrality is assumed. One can discretize the GBM process by:

$$S_{t+1} = S_t * (1 + r\delta t + \sigma\sqrt{\delta t}\varepsilon, \varepsilon \sim N(0,1)$$

Based on the above equation, one can scheme the following:

$$S_1 = S_0 e^{\left(r - \frac{1}{2}\sigma^2\right)\delta t + \sigma\sqrt{\delta t}\varepsilon}$$

$$S_2 = S_1 e^{\left(r - \frac{1}{2}\sigma^2\right)\delta t + \sigma\sqrt{\delta t}\varepsilon}$$

$$S_n = S_{n-1} e^{\left(r - \frac{1}{2}\sigma^2\right)\delta t + \sigma\sqrt{\delta t}\varepsilon}$$

The equations above provide what is needed to value a European call option. If one can assume that the value of the option is a function of the spot price and time (V(S,t)), then the value for the option is given by[5]:

$$V(S_{0,0}) = e^{-rT}\frac{1}{n}\sum_{i=1}^{n} V(S,t)$$

The aforementioned Monte Carlo (MCS) method is attractive relative to other pricing techniques (i.e. BSM, binomial lattices) due to its flexibility, ease of implementation/modification, and the error convergence rate is independent of the dimension of the problem.[6] MCS is not without disadvantages as it is difficult to value American options as one would need to know the value at each date if not exercised, to compare the intrinsic value. Moreover, MCS may be inefficient with respect to the amount of time it takes to implement.[7]

[4] Young, Stephen D., Financial Engineering Notes, p. 137
[5] Ibid p 138
[6] Ibid p 153
[7] Back, Kerry. "A Course in Derivatives Securities", p. 88

Part II

Enhancements to Monte Carlo Simulation

A. Antithetic Variates

Antithetic variates method is a variance decreasing technique used in Monte Carlo methods. For example, consider an error reduction in the simulated signal has a square root convergence (standard deviation of the solution), a very large number of sample paths is required to obtain an accurate result[8]. An antithetic variate is a random variable *y* with the same mean as *x* and a negative correlation with *x*. It follows that the random variable z=(x+y)/2 and will have the same mean as *x* and a lower variance. The standard approach to context pricing is to generate two negatively underlying asset prices which can be done by using the GBM process:[9]

$$S_{t+1} = S_t e^{\left(r-\frac{1}{2}\sigma^2\right)\delta t+\sigma\sqrt{\delta t}\varepsilon_1}$$

$$S'_{t+1} = S_t e^{\left(r-\frac{1}{2}\sigma^2\right)\delta t+\sigma\sqrt{\delta t}\varepsilon_2}$$

In the above equation, $\varepsilon_2 = -\varepsilon_1$. Given the values, the current value equation would be:

$$V(S_0, 0) = e^{-rT}\frac{1}{n}\sum_{i=1}^{n}\frac{V(S,t)+V(S',t)}{2}$$

B. Control Variates

Similar to antithetic variates, control variates is also a variance reduction method applied to Monte Carlo Simulation which exploits information about the errors in estimates of known quantities to reduce the error of an estimate of an unknown quantity.[10] This method is useful when trying to simulate the expected value of a random variable, X. A second random variable, Y, for which the expected value is known, is introduced. The correlation between the two random variables must then be maximized such that the variance of the estimate of the X is reduced, significantly turning it into an improved simulation.[11]

For the control variate technique, one must know the expectation of the correlated problem analytically or numerically. Based on a path dependent contract that has no closed-form solution, one can simulate a value for a European call by: [12]

$$P^* = P_{simulated} + (BS_{analytical} - BS_{simulated})$$

[8] Wikipedia: http://en.wikipedia.org/wiki/Antithetic_variates

[9] Young, Stephen D., Financial Engineering Notes, p. 138

[10] http://en.wikipedia.org/wiki/Control_variates

[11] Young, Stephen D., Financial Engineering Notes, p. 138

[12] Young, Stephen D., Financial Engineering Notes, p. 138

C. Quasi-Monte Carlo Methods

In numerical analysis, quasi-Monte Carlo method is a method for numerical integration and solving some other problems using low-discrepancy sequences (also called quasi-random sequences). Low discrepancy sequence is not random, but deterministic, quasi-Monte Carlo method can be seen as a deterministic algorithm or de-randomized algorithm. In this specific case, the bound (e.g., $\varepsilon \leq V(f)\, D_N$) for error which the error is hard to estimate. In order to improve and analyze estimate the variance, one can randomize the method. The resulting method is called the randomized quasi-Monte Carlo method and can be also viewed as a variance reduction technique for the standard Monte Carlo method.[13] This method circumvents the problem of clustering which occurs only with the regular Monte Carlo method. This clustering is simply random points that do not spread optimally for a specific number of dimensions.[14]

D. Cholesky and Eigenvalue Decomposition

The Cholesky decomposition is commonly used in the Monte Carlo method for simulating systems with multiple correlated variables: the correlation matrix is decomposed, to give the lower-triangular L. Applying this to a vector of uncorrelated samples, u, produces a sample vector Lu with the covariance properties of the system being modeled.[15] When Eigenvalue decomposition is used on a matrix of measured, real data, the inverse may be less valid when all eigenvalues are used unmodified in the form above. This is because as eigenvalues become relatively small and their contribution to the inversion is large. Those near zero or at the "noise" of the measurement system will have undue influence and could hamper solutions using the inverse. “Two mitigations have been proposed: 1) truncating small/zero eigenvalues, 2) extending the lowest reliable eigenvalue to those below it”. The reliable eigenvalue can be found by assuming that eigenvalues of extremely similar and low value are a good representation of measurement noise. [16]

If standard deviations are equal to one, generating standard normal, then a covariance matrix can be decomposed as:[17]

$$\Sigma = \begin{pmatrix}\sigma & 0\\ 0 & \sigma\end{pmatrix}\begin{pmatrix}1 & \rho\\ \rho & 1\end{pmatrix}\begin{pmatrix}\sigma & 0\\ 0 & \sigma\end{pmatrix} = \begin{pmatrix}1 & \rho\\ \rho & 1\end{pmatrix} = \begin{pmatrix}1 & 0\\ \rho & \sqrt{1-\rho^2}\end{pmatrix}\begin{pmatrix}1 & \rho\\ 0 & \sqrt{1-\rho^2}\end{pmatrix}$$

The upper equation provides a lower and upper triangular matrix which satisfies Σ = MM’ which can set X=MY and have the below equation:[18]

$$\begin{pmatrix}X_1\\ X_2\end{pmatrix} = \begin{pmatrix}1 & 0\\ \rho & \sqrt{1-\rho^2}\end{pmatrix}\begin{pmatrix}Y_1\\ Y_2\end{pmatrix};\ X_1 = Y_1\ and\ X_2 = \rho Y_1 + \sqrt{1-\rho^2}Y_2$$

---

[13] http://en.wikipedia.org/wiki/Quasi-Monte_Carlo_method
[14] Young, Stephen D., Financial Engineering Notes, p. 139
[15] http://en.wikipedia.org/wiki/Cholesky_decomposition
[16] http://en.wikipedia.org/wiki/Eigendecomposition_of_a_matrix
[17] Young, Stephen D., Financial Engineering Notes, p. 141
[18] Ibid p 141

E. Calculating Path-wise Greeks

Path-wise derivative estimate derivative directly, without simulating multiple times. It takes advantage of additional information about the dynamics and parameter dependence of a simulated process which in turn converges to their true values much quicker. [19] The use of path-wise Delta is the best method to estimate Delta in this case as well as Gamma and Vega. Specifically, path-wise Delta considers the sample paths of the underlying asset price rather than considering the up and down disruptions. The path-wise Delta can be found by calculating the discounted sample average of [S(T)/S]x.[20]

F. Discretizing a Stochastic Differential Equation (including Euler & Milstein)

In mathematical finance, financial markets are often modeled by the solution X_t of a stochastic differential equation (SDE). The price (at time 0) of a European option with maturity T and payoff f (XT) is given by an expression like[21]:

$$C(X) = E(e - rTf(Xt))$$

The simplest way to discretize the process in most equations is to use Euler discretization. This is equivalent to approximating the integrals using the left point rule. Hence the first integral is approximated as the product of the integrand at time t, and the integration range dt. Milstein is a method that adds a term to the Euler scheme and expands drift and diffusion terms to O(h). The key to the Milstein scheme is that the accuracy of the discretization is increased by considering expansions of the coefficients[22].

---

[19] http://www.mathfinance.cn/pathwise-derivative-vs-finite-difference-for-greeks-computation/

[20] Young, Stephen D., Financial Engineering Notes, p. 100

[21] http://page.math.tu-berlin.de/~bayer/files/euler_talk_handout.pdf

[22] http://www.frouah.com/finance%20notes/Euler%20and%20Milstein%20Discretization.pdf

Part III

Monte Carlo Implementation: European Call and Put Valuation

A. Black Scholes Valuation: A Benchmark.

Finally, we consider European options that are Out-the-Money, At-the-Money, and In-the Money for down, natural, and up states of an initial asset price S_0 (80, 100, 120). For the purpose of this example, we first calculate estimates for European call and put options as well as Delta using the provide inputs across the three states of the initial asset price S_0.

INPUTS	
Strike (X)	100.00
Risk Free (rf)	0.05
Annual Volatility (sigma)	0.20
Time to Maturity (T)	1

	BSM
Initial Stock Price (Sd = 80)	
European Call	1.8594
European Put	16.9824
Call Delta	0.2219
Put Delta	(0.7781)
Initial Stock Price (S0 = 100)	
European Call	10.4506
European Put	5.5735
Call Delta	0.6368
Put Delta	(0.3632)
Initial Stock Price (Su = 120)	
European Call	26.1690
European Put	1.2920
Call Delta	0.8965
Put Delta	(0.1035)

B. Simple Monte Carlo and Antithetic Variate Enhancement: A Comparison

Using MATLAB, we next simulated the underlying asset price paths in an increasing number of iterations (50, 500, 5000) in order to calculate a "base case" set of MCS option value and Delta estimates. As expected, as the number of simulations increase, the option value and Delta estimates converge upon those of the Black Scholes model.

	MSC			
(n)	50	500	5,000	
Initial Stock Price (Sd = 80)				BSM
European Call	2.7573	2.1212	1.8878	1.8594
European Put	15.9880	16.3634	17.0960	16.9824
Call Delta	1.0626	0.4383	0.2028	0.2219
Put Delta	(1.1326)	(1.0706)	(0.6176)	(0.7781)
Initial Stock Price (S0 = 100)				
European Call	12.0700	11.0969	10.4500	10.4506
European Put	4.8277	5.1189	5.6795	5.5735
Call Delta	1.7171	0.9516	0.5825	0.6368
Put Delta	(0.4781)	(0.5573)	(0.2378)	(0.3632)
Initial Stock Price (Su = 120)				
European Call	28.8358	27.3736	26.0975	26.1690
European Put	1.1204	1.1755	1.3483	1.2920
Call Delta	2.0896	1.3652	0.7539	0.8965
Put Delta	(0.1056)	(0.1437)	(0.0664)	(0.1035)

In a similar fashion, we again simulated the underlying asset price paths for a set of MCS option value and Delta estimates, in this instance, incorporating the antithetic variate enhancement. As with the base case Monte Carlo method outlined above, the option value and Delta estimates converge upon those of the Black Scholes model as the number of simulations increase.

	MSC w/Antithetic Variate			
(n)	50	500	5,000	
Initial Stock Price (Sd = 80)				BSM
European Call	2.1682	1.9102	1.9147	1.8594
European Put	17.1584	17.0094	17.0080	16.9824
Call Delta	0.6031	0.2807	0.2559	0.2219
Put Delta	(0.6163)	(0.7471)	(0.7663)	(0.7781)
Initial Stock Price (S0 = 100)				
European Call	10.6949	10.4572	10.5393	10.4506
European Put	5.6519	5.5505	5.6252	5.5735
Call Delta	1.1059	0.6562	0.7046	0.6368
Put Delta	(0.1135)	(0.3716)	(0.3175)	(0.3632)
Initial Stock Price (Su = 120)				
European Call	26.5994	26.2385	26.2515	26.1690
European Put	1.5232	1.3258	1.3299	1.2920
Call Delta	1.2985	0.9533	0.9310	0.8965
Put Delta	0.0791	(0.0745)	(0.0911)	(0.1035)

Note that the estimates for both option values and Delta are more exact given the antithetic variate which produces smaller standard errors when compared to the base case. This enhancement clearly improves the Monte Carlo Simulation.

	Standard Error					
	MSC			MSC w/ Antithetic Variate		
(n)	50	500	5,000	50	500	5,000
Initial Stock Price (Sd = 80)						
European Call	7.2794	6.3053	6.0195	4.3853	4.0817	4.0176
European Put	12.6852	12.7098	12.8632	1.9905	1.8370	1.8055
Initial Stock Price (S0 = 100)						
European Call	16.9420	15.4553	14.8401	8.0448	7.5550	7.4464
European Put	8.2972	8.3969	8.7792	5.0868	4.7944	4.7275
Initial Stock Price (Su = 120)						
European Call	24.5041	22.9919	22.6046	6.8226	6.3438	6.2465
European Put	4.1616	4.0389	4.2099	3.1440	2.8773	2.8332

References

http://page.math.tu-berlin.de/~bayer/files/euler_talk_handout.pdf

http://www.frouah.com/finance%20notes/Euler%20and%20Milstein%20Discretization.pdf

http://www.mathfinance.cn/pathwise-derivative-vs-finite-difference-for-greeks-computation/

Wikipedia: http://en.wikipedia.org/wiki/Antithetic_variates

Wikipedia: http://en.wikipedia.org/wiki/Monte_Carlo_method

Wikipedia: http://en.wikipedia.org/wiki/Antithetic_variates

Wikipedia: http://en.wikipedia.org/wiki/Control_variates

Wikipedia: http://en.wikipedia.org/wiki/Cholesky_decomposition

Wikipedia: http://en.wikipedia.org/wiki/Eigendecomposition_of_a_matrix

Young, D. Stephen, *Financial Engineering Notes*

APPENDIX
MATLAB SYNTAX

1. BLACK SCHOLES EUROPEAN OPTION VALUATION AND DELTA

```
function [C,P,Delta,Gamma,Theta] = BlackScholes(S0,X,r,sigma,T)

%INPUTS
S0=100;
X=100;
r=0.05;
sigma=0.2;
T=1;

d1=(log(S0/X)+(r+sigma^2/2)*T)/(sigma*sqrt(T));
d2=d1-(sigma*sqrt(T));
C=S0*normcdf(d1)-X*(exp(-r*T)*normcdf(d2)) %EUROPEAN CALL
P=X*exp(-r*T)*normcdf(-d2)-S0*normcdf(-d1) %EUROPEAN PUT
[CallDelta,PutDelta]=blsDelta(S0,X,r,T,sigma) %DELTA
end
```

2. MONTE CARLO SIMULATION EUROPEAN OPTION VALUATION

```
function Output = MCS(S0,X,r,sigma,T,n)

%INPUTS
S0=80;
X=100;
r=0.05;
sigma=0.2;
T=1;
n=50;
nsteps=50;  %NUMBER OF REPLICATIONS(n)
dt=T/nsteps;  %DISCRETIZE TIME STEPS
randn('state',0)
mat=randn(nsteps,n);

mat=exp((r-0.5*sigma^2)*dt+sigma*sqrt(dt).*mat);

mat=cumprod(mat,1);

mat = mat.*S0;

C=1;  %CALL
P=-1;  %PUT

valuecall=exp(-r*T) * max(C*(mat(end,:)-X) , 0);
valueput=exp(-r*T) * max(P*(mat(end,:)-X) , 0);

Output.call = 1/n*sum(valuecall);
Output.std_call =std(valuecall);

Output.put =1/n*sum(valueput);
```

```
Output.std_put =std(valueput);
end
```

3. MONTE CARLO SIMULATION EUROPEAN OPTION VALUATION w/ ANTITHETIC VARIATE

```
function Output = MCS_Anti(S0,X,r,sigma,T,n)

%INPUTS
S0=80;
X=100;
r=0.05;
sigma=0.2;
T=1;
n=50;
nsteps=50;  %NUMBER OF REPLICATIONS(n)
dt=T/nsteps;  %DISCRETIZE TIME STEPS
randn('state',0)
mat=randn(nsteps,n);

mat1=exp((r-0.5*sigma^2)*dt+sigma*sqrt(dt).*mat);
mat2=exp((r-0.5*sigma^2)*dt+sigma*sqrt(dt).*(-mat));

mat1=cumprod(mat1,1);
mat2=cumprod(mat2,1);

mat1 = mat1.*S0;
mat2 = mat2.*S0;

C=1;  %CALL
P=-1;  %PUT

valuecall=exp(-r*T) * (max(C*(mat1(end,:)-X) , 0)+max(C*(mat2(end,:)-
X),0))/2;
valueput=exp(-r*T) * (max(P*(mat1(end,:)-X) , 0)+max(P*(mat2(end,:)-X),0))/2;

Output.call= 1/n*sum(valuecall);
Output.std_call=std(valuecall);

Output.put=1/n*sum(valueput);
Output.std_put=std(valueput);
end
```

4. MONTE CARLO SIMULATION EUROPEAN DELTA

```
function [CallDelta,PutDelta] = MCSDelta(S0,X,r,sigma,T,n)

%INPUTS
S0=80;
X=100;
r=0.05;
sigma=0.2;
T=1;
n=50;
nsteps=50;  %NUMBER OF REPLICATIONS(n)
```

```
dt=T/nsteps;  %DISCRETIZE TIME STEPS
randn('state',0)
mat=randn(nsteps,n);

mat=exp((r-0.5*sigma^2)*dt+sigma*sqrt(dt).*mat);
mat=cumprod(mat,1);
mat = mat.*S0*1.01;

C=1;  %CALL
P=-1;  %PUT

valuecall=exp(-r*T) * max(C*(mat(end,:)-X) , 0);
valueput=exp(-r*T) * max(P*(mat(end,:)-X) , 0);

call_value= 1/n*sum(valuecall);
put_value=1/n*sum(valueput);

mat1=randn(nsteps,n);

mat1=exp((r-0.5*sigma^2)*dt+sigma*sqrt(dt).*mat1);
mat1=cumprod(mat1,1);
mat1 = mat1.*S0*0.99;

valuecall1=exp(-r*T) * max(C*(mat1(end,:)-X) , 0);
valueput1=exp(-r*T) * max(P*(mat1(end,:)-X) , 0);

call_value1= 1/n*sum(valuecall1);
put_value1=1/n*sum(valueput1);

CallDelta = (call_value-call_value1)/(S0*(1.01-0.99));
PutDelta = (put_value-put_value1)/(S0*(1.01-0.99));

CallDelta
PutDelta
end
```

5. MONTE CARLO SIMULATION EUROPEAN DELTA w/ ANTITHETIC VARIATE

```
function [CallDelta_Anti, PutDelta_Anti] = MCS_Anti_Delta(S0,X,r,sigma,T,n)

%INPUTS
S0=80;
X=100;
r=0.05;
sigma=0.2;
T=1;
n=50;
nsteps=50;  %NUMBER OF REPLICATIONS(n)
dt=T/nsteps;  %DISCRETIZE TIME STEPS
randn('state',0)
mat=randn(nsteps,n);

mat1=exp((r-0.5*sigma^2)*dt+sigma*sqrt(dt).*mat);
```

```
mat2=exp((r-0.5*sigma^2)*dt+sigma*sqrt(dt).*(-mat));

mat1=cumprod(mat1,1);
mat2=cumprod(mat2,1);

mat1 = mat1.*S0*1.01;
mat2 = mat2.*S0*1.01;

C=1;  %CALL
P=-1;  %PUT

valuecall=exp(-r*T) * (max(C*(mat1(end,:)-X) , 0)+max(C*(mat2(end,:)-
X),0))/2;
valueput=exp(-r*T) * (max(P*(mat1(end,:)-X) , 0)+max(P*(mat2(end,:)-X),0))/2;

call_value= 1/n*sum(valuecall);
put_value=1/n*sum(valueput);

mati=randn(nsteps,n);

mati1=exp((r-0.5*sigma^2)*dt+sigma*sqrt(dt).*mati);
mati2=exp((r-0.5*sigma^2)*dt+sigma*sqrt(dt).*(-mati));

mati1=cumprod(mati1,1);
mati2=cumprod(mati2,1);

mati1 = mati1.*S0*0.99;
mati2 = mati2.*S0*0.99;

valuecall1 = exp(-r*T) * (max(C*(mati1(end,:)-X) , 0)+max(C*(mati2(end,:)-
X),0))/2;
valueput1 = exp(-r*T) * (max(P*(mati1(end,:)-X) , 0)+max(P*(mati2(end,:)-
X),0))/2;

call_value1 = 1/n*sum(valuecall1);
put_value1 =1/n*sum(valueput1);

CallDelta_Anti = (call_value-call_value1)/(S0*(1.01-0.99));
PutDelta_Anti = (put_value-put_value1)/(S0*(1.01-0.99));

CallDelta_Anti
PutDelta_Anti
end
```