Plain Vanilla Options Pricing using Monte Carlo Simulations in Python

Mukund Raghav Sharma

May 2014

1 Preliminaries

We start off the process of pricing by deriving the value of the underlying, in our case, stock at time 't'.
The dynamics of a stock i.e. geometeric brownian motion are given by the following:

$$dS_t = S_t\mu dt + S_t\sigma dB_t \tag{1}$$

From (1), we can deduce:

$$\frac{dS_t}{S_t} = \mu dt + \sigma dB_t \tag{2}$$

And hence, we clearly know that by integrating both sides, we will be $\ln(S_t)$ on the left hand side of (2).

Therefore, we consider a function $f(S_t)$ that we apply the Ito's Lemma to.
Now, let $f(S_t) = \ln(S_t)$

Therefore:

$f^{'}(S_t) = \frac{1}{S_t}$
$f^{''}(S_t) = -\frac{1}{S_t^2}$

Ito's Lemma in one dimension:

$f(x) = f^{'}(x)dx + \frac{1}{2}f^{''}(x)d[x]$

where [x] is the quadratic variation of x.

Now, we have:

$f(S_t) = \frac{1}{S_t}dS_t - \frac{1}{2}\frac{1}{S_t^2}d[S_t]$

Clearly, $d[S_t] = S_t^2\sigma^2 dt$

Therefore, by putting the value of geometeric brownian motion in the first term, we end up getting:

$$f(S_t) = (\mu - \frac{1}{2}\sigma^2)dt + \sigma dB_t \tag{3}$$

By Integration and by applying the exponential function on both sides, we get:

$$S_t = S_0 exp((\mu - \frac{1}{2}\sigma^2)t + \sigma B_t) \tag{4}$$

2 Risk Neutral Pricing

The price of a vanilla option is the discounted risk neutral expectation of the pay off i.e.:

1. For Call Options:

$$E^Q[e^{-rt}(S_t - K)^+] \tag{5}$$

2. For Put Options:

$$E^Q[e^{-rt}(K - S_t)^+] \tag{6}$$

Also, B_t or Brownian Motion at time t can be written in the following form:
$B_t = \epsilon\sqrt{t}$

Where ϵ is a random drawing from N(0, 1)

Substituing the value of S_t found in (4), we get:
1. For Call Options:

$$E^Q[e^{-rt}(S_0 exp((\mu - \frac{1}{2}\sigma^2)t + \sigma\epsilon\sqrt{t}) - K)^+] \tag{7}$$

2. For Put Options

$$E^Q[e^{-rt}(K - S_0 exp((\mu - \frac{1}{2}\sigma^2)t + \sigma\epsilon\sqrt{t}))^+] \tag{8}$$

Taking the exponential of -rt term out of the Risk Neutral Expectation (since, it is a constant) with respect to the other terms, we get: 1. For Call Options:

$$e^{-rt}E^{Q}[(S_0 exp((\mu - \frac{1}{2}\sigma^2)t + \sigma\epsilon\sqrt{t}) - K)^{+}] \tag{9}$$

2. For Put Options

$$e^{-rt}E^{Q}[(K - S_0 exp((\mu - \frac{1}{2}\sigma^2)t + \sigma\epsilon\sqrt{t}))^{+}] \tag{10}$$

3 Monte Carlo Simulations

We use Monte Carlo Simulations approximate the risk neutral expectation operator; this can be concluded due to the law of large numbers.

For this approximation, we will need to take the mean of a significant number of paths; the more the number of paths, the better the approximation of the risk neutral measure.

$$\sum_{i=0}^{N} \frac{X_i}{N} \tag{11}$$

Where N is the number of paths and X_i is the price of the option.

4 Python Code

The following is the code that runs the Monte Carlo Pricer based on the inputs by the user i.e the expiry, strike, spot, volatility, the interest rate and the number of paths. I used the math library in Python to generate a random drawing from the normal distribution of mean 1 and variance 0 as opposed to the Box Muller method; I did this specifically because the Box Muller method results in low discrepancy numbers as opposed to the library function. Apart from that the code is pretty straight forward.

```
import random
import math

class VanillaOption:
    def __init__(self, option_type, expiry, strike, spot, volatility, interest_rate, numb
        self.option_type = option_type
        self.expiry = expiry
        self.strike = strike
        self.spot = spot
        self.volatility = volatility
        self.interest_rate = interest_rate
        self.number_of_paths = number_of_paths

    def monte_carlo_pricer(self):
        variance = self.volatility * self.volatility * self.expiry
        standard_deviation = math.sqrt(variance)
        ito_correction = -0.5 * variance
        spot_changed = self.spot * math.exp(self.interest_rate * self.expiry + ito_corre
        sum = 0
        for i in range(0, self.number_of_paths):
            normal = random.normalvariate(0,1)
            stock_val = spot_changed * math.exp(standard_deviation * normal)
            if self.option_type.lower() == 'call':
                sum += max(stock_val - self.strike, 0.0)
            elif self.option_type.lower() == 'put':
                sum += max(self.strike - stock_val, 0.0)
        result = sum / self.number_of_paths
        result *= math.exp(-self.interest_rate * self.expiry)
        return result

def main():
    option_type = str(raw_input("Enter 'call' for call options or 'put' for put options:
    expiry = float(raw_input("Enter the expiry of the option: "))
    strike = float(raw_input("Enter the strike price of the option: "))
    spot = float(raw_input("Enter the spot value of the option: "))
    volatility =  float(raw_input("Enter the volatility value of the option as a rate: "
    interest_rate = float(raw_input("Enter the risk free interest rate as a rate: ")) / 
    number_of_paths = int(raw_input("Enter the number of paths you want to run the simul
    call = VanillaOption(option_type, expiry, strike, spot, volatility, interest_rate, n
    print "Result: ", call.monte_carlo_pricer()

if __name__ == '__main__':main()
```

5 Citation

1. C++ Design Patterns and Derivative Pricing by Mark Joshi
2. Stochastic Calculus for Finance II - Continuous Time Models by Steven Shreve