MÄLARDALENS HÖGSKOLA

A Monte-Carlo calculation for Barrier options

Using Python

Mwangota Lutufyo and Omotesho Latifat oyinkansola

2016-10-19

MMA707 – Analytical Finance I:
Lecturer: Jan Roman

Division of Applied Mathematics
School of Education, Culture and Communication
Mälardalen University
Box 883, SE-721 23 Västerås, Sweden

Table of Contents

Introduction

Barrier options are considered more complicated in computation. Therefore, in such mind numerical methods must be applied. One of such methods is Monte Carlo simulation. Monte Carlo simulation is one of the most important algorithms in finance and numerical science in general. It's importance stems from the fact that it is quite powerful when it comes to option pricing. This paper analyses the pricing of barrier options using Monte Carlo Simulation. In the first section this report defines the barrier options and introduce the monte Carlo as a pricing methodology. In the second section we focused on valuating barrier options and plotting by using Python program. The last section is about the results we got from our program.

An Overview of Barrier Option

Barrier options are one of the most widely traded derivatives in the financial markets. They have special characteristics which distinguish them from ordinary options. They are characterised by a strike level and a barrier level, as well as by a cash rebate associated with crossing the barrier. Their payoff depends on whether the underlying asset's price reaches a certain level during a certain period of time (Hull 2000). These options are activated or expired when the underlying asset price either hits or does not hit a specified barrier before expiry. One reason that an investor prefers a barrier option to an ordinary vanilla option is that barrier options are generally cheaper than standard options. This is because the asset price has to cross a certain barrier for the option holder to receive the payoff. The other reason is that barrier options may match risk hedging needs more closely than standard options.

Types of Barriers and Barrier Options

The following definitions can be found in (Chriss 1997, 434 and Ivan 2004).

1. Knock-out options start out as ordinary call or put options, but they become null and void if the spot price ever crosses a certain predetermined knock-out barrier, even before the expiration date.

2. Knock-in options start their lives inactive, in a sense null and void, and only become active on the event that the stock price crosses the knock-in barrier, then it becomes an ordinary call or put option.

The barrier option can be further portrayed by the position of the barrier relative to the initial value of underlying.

- If the barrier is above the initial asset value, one has an up option.
- If the barrier is below the initial asset value, one has a down option.

(Hull 2000, 662) offers the following definitions:

1. **Down-and-out:** An option that terminates when the price of the underlying asset declines to a predetermined level.
2. **Up-and-out:** An option that terminates when the price of the underlying asset increases to a predetermined level.
3. **Down-and-in:** An option that comes into existence when the price of the underlying asset declines to a predetermined level.
4. **Up-and-in:** An option that comes into existence when the price of the underlying asset increases to a predetermined level.

Barrier options can also have cash rebates associated with them. This is a consolation prize paid to the holder of the option when an out barrier is knocked out or when an in barrier is never knocked in (Ivan 2004). The rebate can be nothing or it could be some fraction of the premium. Rebates are usually paid immediately when an option is knocked out, however, payments can be deferred to the maturity of the option, (Kotze 1999).

Pricing barrier options

Barrier options may be priced using binomial models, Monte Carlo simulations or special versions of the standard Black-Scholes-Merton model. The first analytical formula was proposed by Merton in 1973 to value a down-and-out call option with one

flat barrier when an underlying process follows a geometric Brownian motion (GBM). Moreover, more analytical formulae, with this simple case, were provided by Reiner and Rubinstein (1991) and Rich (1994) as follows;

Notations:

H: $Barrier\ \ level$

X: $Strike$

σ: $Volatility$

T: $Time\ \ to\ \ expiration$

b: $Cost\ \ of\ \ carry\ \ rate$ (r − q where q are the dividends)

r: $\ Risk - free\ \ rate$

C: $Price\ \ of\ \ a\ \ barrier\ \ option$

S: $\ Spot\ \ stock\ \ price$

Formulas:

$$\mu = \begin{cases} 1, & if\ \ Down \\ -1, & if\ \ Up \end{cases} \qquad \varphi = \begin{cases} 1, & if\ \ Call \\ -1, & if\ \ Put \end{cases}$$

$$X_1 = \frac{\ln\left({}^{S}/_{X}\right)}{\sigma\sqrt{T}} + (1+\mu)\sigma\sqrt{T} \qquad X_2 = \frac{\ln\left({}^{S}/_{X}\right)}{\sigma\sqrt{T}} + (1+\mu)\sigma\sqrt{T}$$

$$Y_1 = \frac{\ln\left({}^{H^2}/_{(SX)}\right)}{\sigma\sqrt{T}} + (1+\mu)\sigma\sqrt{T} \qquad Y_2 = \frac{\ln\left({}^{H}/_{S}\right)}{\sigma\sqrt{T}} + (1+\mu)\sigma\sqrt{T}$$

$$Z = \frac{\ln\left({}^{H}/_{S}\right)}{\sigma\sqrt{T}} + \lambda\sigma\sqrt{T}$$

$$\mu = \frac{b - {}^{\sigma^2}/_{2}}{\sigma^2} \qquad \lambda = \sqrt{\mu^2 + \frac{2r}{Q^2}}$$

$$A = \phi.S.N(\phi x_1) - \phi.X.e^{-rT}N\left(\phi x_1 - \phi\sigma\sqrt{\mathrm{T}}\right)$$

$$B = \phi.S.N(\phi x_2) - \phi.X.e^{-rT}N\left(\phi x_2 - \phi\sigma\sqrt{\mathrm{T}}\right)$$

$$C = \phi.S.\left(\frac{H}{S}\right)^{2(\mu+1)} N(\eta y_1) - \phi X e^{-rT}\left(\frac{H}{S}\right)^{2\mu} N\left(\eta y_1 - \eta\sigma\sqrt{\mathrm{T}}\right)$$

$$D = \phi.S.\left(\frac{H}{S}\right)^{2(\mu+1)} N(\eta y_2) - \phi X e^{-rT}\left(\frac{H}{S}\right)^{2\mu} N\left(\eta y_2 - \eta\sigma\sqrt{\mathrm{T}}\right)$$

$$E = Ke^{-rT}[N\left(\eta x_2 - \eta\sigma\sqrt{\mathrm{T}}\right) - \left(\frac{H}{S}\right)^{2\mu} - N\left(\eta y_2 - \eta\sigma\sqrt{\mathrm{T}}\right)]$$

$$F = Ke^{-rT}\left(\frac{H}{S}\right)^{\mu+\lambda} N(\eta z) - \left(\frac{H}{S}\right)^{\mu-\lambda} N\left(\eta z - 2\eta\lambda\sigma\sqrt{\mathrm{T}}\right)$$

Then the prices of call barriers are given by;

Type		X < H	X > H
Down-and-In	S > H	A-B+D+E	C+E
Up-and-In	S < H	B-C+D+E	A+E
Down-and-Out	S > H	B-D+F	A-C+F
Up-and-Out	S < H	A-B+C-D+F	F

Tabel 1 Prices of Call barriers

Then the prices of put barriers are given by;

Type		X < H	X > H
Down-and-In	S > H	A+E	B-C+D+E
Up-and-In	S < H	C+E	A-B+D+E
Down-and-Out	S > H	F	A-B+C-D+F
Up-and-Out	S < H	A-C+F	B-D+F

Tabel 2 Prices of Put barriers

Volatility:

In pricing options volatility plays an important role. Barrier options are especially sensitive to volatility. For knock-out options, increased volatility has the effect of decreasing the option value as knock-out becomes more probable. Knock-in options however increase in price with increased volatility as knock-in becomes more probable.

Monte Carlo Simulation:

Barrier options are considered more complicated in computation. Therefore, in such mind numerical methods must be applied. One of such method is Monte Carlo simulation. Monte Carlo simulation is one of the most important algorithms in finance and numerical science in general. It's importance stems from the fact that it is quite powerful when it comes to option pricing. In comparison to other numerical methods, the Monte Carlo method can easily cope with high-dimensional problems where the complexity and computational demand, respectively, generally increase in linear fashion.

When we use the Monte Carlo simulation to price an option, the approach can be summarized into three steps.

- Simulate *n* sample paths of the underlying asset price over the time interval.
- Calculate the payoff of the option for each path.
- Average the discounted payoffs over sample paths.

Barrier options are path-dependent - their payoffs are determined by whether or not the price of the asset hits a certain level during the life of the option. Due to this path-dependency, simulation of the entire price evolution is necessary. Therefore, the first step in pricing barrier options using Monte Carlo methods is to simulate the price evolution. The basic principle here is to simulate as many possible scenarios and to

average those scenarios to get an expectation. To simulate a sample path, we have to choose a stochastic differential equation describing the dynamics of the price. We consider the price of the underlying asset is described by a Geometric Brownian Motion:

$$dS_t = \mu S_t + \sigma S_t dW_t \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \text{(1)}$$

By applying Ito's lemma, we receive the following expression:

$$d\log S_t = \left(\mu - \frac{1}{2}\sigma^2\right)dt + \sigma dW_t \ldots\ldots\ldots\ldots \text{(2)}$$

S_t is log-normally distributed and thereby we have

$$E\left[\log\left(\frac{S(t)}{S(0)}\right)\right] = \nu t$$

$$Var\left[\log\left(\frac{S(t)}{S(0)}\right)\right] = \sigma^2 t$$

$$E\left[\left(\frac{S(t)}{S(0)}\right)\right] = e^{\mu t}$$

$$Var\left[\left(\frac{S(t)}{S(0)}\right)\right] = e^{2\mu t}(e^{\sigma^2 t} - 1)$$

Where $\nu = \mu - \sigma^2/2.$

Integrate equation (2) yielding:

$$S_t = S_0 \exp(\nu t + \sigma \int_0^t dW(\tau))$$

Let us discretize the time interval (0, *T*) with a time step δt. From the equation above and the properties of the standard Wiener process, we obtain

$$S_{t+\delta t} = S_t \exp(\nu\delta t + \sigma\sqrt{\delta t \varepsilon}) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \text{(3)}$$

Where, $\varepsilon \sim N(0,1)$ is a standard normal random variable.

Now based on equation (3), we can therefore generate sample paths for the asset price.

Pricing barrier option in Python

For the valuation of the Barrier options, we wrote a program in python that include the valuation of option price using Black-Schole - Merton model. We wrote the program in

such that it can price both call and put barrier options. For the case of this report we will present a Python program for European call barrier option (up-and-out and up and in parity).

The idea is very simple; If we buy an up-and-out European call and an up-and-in European call, then the following parity should hold good:

$$Call_{up-and-out} + Call_{up-and-in} = Call$$

Moreover, the logic behind this is, if the stock price reaches the barrier, then the first call is worthless and the second call will be activated. If the stock price never touches the barrier, the first call will remain active, while the second one is never activated. Either way, one of them is active.

Using this idea of we use the Monte Carlo simulation to present a parity for values we input in our program. The code is attached in appendix.

Results analysis

We input a set of values to test whether the summation of an up-and-out call and an up-and-in call will be the same as a vanilla call:

S0	**60.**	**Today Stock Price**
X	**60.**	**Exercise Price**
Barrier	61	Barrier level
T	1	**Maturity in Years**
R	0.05	**Risk-Free Rate**
Sigma	0.2	**volatility (annualized)**

Tabel 3 Inputs for the valuation of European Call Opton.

The following output proves the parity mentioned above:

Up-Out-Call = **2.48**,

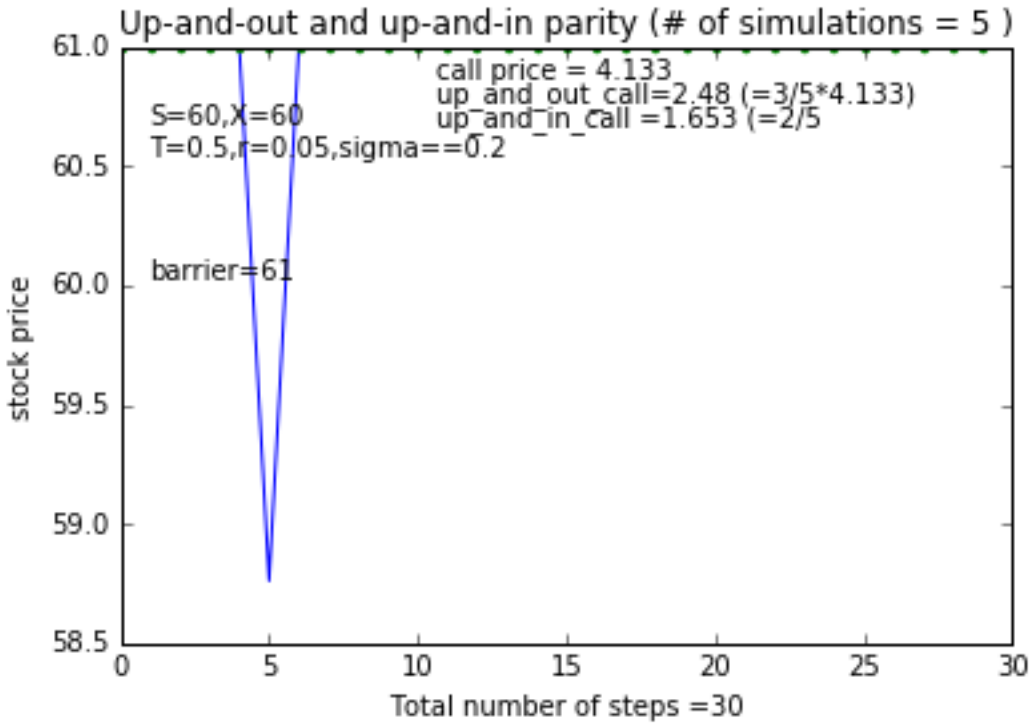
Up-In-Call = **1.653** and

Call (Vanilla) = **4.133**

Such that **2.48** + **1.653** = **4.133**

See the graph hereunder for the same results.

Graphical presentation of an up-and-out and up-and-in parity



Conclusion

This paper gives an overview of how to estimate barrier option prices via Monte Carlo simulation was provided. We started by defining the concept of barrier option and its computation. We discussed the barrier option as the general overview but in the result analysis we narrow down our discussion on barrier option (Up-In and out call option). Moreover, we provided the graph showing the parity of call option.

Appendix: Python Program Code

```
def bs_call(S,X,T,rf,sigma):
    """
        Objective: Black-Schole-Merton option model
        Format     : bs_call(S,X,T,r,sigma)
                    S: current stock price
                    X: exercise price
                    T: maturity date in years
                   rf: risk-free rate (continusouly compounded)
               sigma: volatiity of underlying security

    """
    from scipy import log,exp,sqrt,stats
    d1=(log(S/X)+(rf+sigma*sigma/2.)*T)/(sigma*sqrt(T))
    d2 = d1-sigma*sqrt(T)
    return S*stats.norm.cdf(d1)-X*exp(-rf*T)*stats.norm.cdf(d2)



def bs_put(S,X,T,rf,sigma):
    """
        Objective: Black-Schole-Merton option model
        Format     : bs_call(S,X,T,r,sigma)
                    S: current stock price
                    X: exercise price
                    T: maturity date in years
                   rf: risk-free rate (continusouly compounded)
               sigma: volatiity of underlying security

    """
```

```
    from scipy import log,exp,sqrt,stats
    d1=(log(S/X)+(rf+sigma*sigma/2.)*T)/(sigma*sqrt(T))
    d2 = d1-sigma*sqrt(T)
    return X*exp(-rf*T)*stats.norm.cdf(-d2)-S*stats.norm.cdf(-d1)



#from math import sqrt, log, pi,exp
#import re
#----------------------------------------------------------#
#--- Cumulative normal distribution              ----------#
#----------------------------------------------------------#
def CND(X):
    """ Cumulative standard normal distribution
            CND(x): x is a scale
            e.g.,
            >>> CND(0)
            0.5000000005248086
    """
    (a1,a2,a3,a4,a5)=(0.31938153,-0.356563782,1.781477937,-
1.821255978,1.330274429)
    L = abs(X)
    K = 1.0 / (1.0 + 0.2316419 * L)
    w = 1.0 - 1.0 / sqrt(2*pi)*exp(-L*L/2.) * (a1*K + a2*K*K + a3*pow(K,3) +
    a4*pow(K,4) + a5*pow(K,5))
    if X<0:
        w = 1.0-w
    return w

import scipy as sp
```

```
from math import exp
import matplotlib.pyplot as pl
S0=60
x=60
barrier=61
T=0.5
n_steps=30.
r = 0.05
sigma=0.2
sp.random.seed(125)
n_simulation =5
dt =T/n_steps
S = sp.zeros([n_steps],dtype=float)
time_ = range(0,int(n_steps), 1)
c=bs_call(S0,x,T,r,sigma)
sp.random.seed(124)
outTotal, inTotal=0.,0.
n_out,n_in=0,0
for j in range(0, n_simulation):
    S[0] = S0
    inStatus=False
    outStatus=True
for i in time_[:-1]:
        e=sp.random.normal()
        S[i+1]=S[i]*exp((r-0.5*pow(sigma,2))*dt+sigma*sp.sqrt(dt)*e)
        if S[i+1]>barrier:
            outStatus=False
            inStatus=True
        pl.plot(time_,S)
```

```
        if outStatus==True:
                outTotal+=c;n_out+=1
        else:
                inTotal+=c;n_in+=1
        S=sp.zeros(int(n_steps))+barrier
        pl.plot(time_,S,'.-')
        upOutCall=round(outTotal/n_simulation,3)
        upInCall=round(inTotal/n_simulation,3)
        pl.figtext(0.15,0.8,'S='+str(S0)+',X='+str(x))
        pl.figtext(0.15,0.76,'T='+str(T)+',r='+str(r)+',sigma=='+str(sigma))
        pl.figtext(0.15,0.6,'barrier='+str(barrier))
        pl.figtext(0.40,0.86, 'call price = '+str(round(c,3)))
        pl.figtext(0.40,0.83,'up_and_out_call='+str(upOutCall)+'
(='+str(n_out)+'/'+str(n_simulation)+'*'+str(round(c,3))+')')
        pl.figtext(0.40,0.80,'up_and_in_call                    ='+str(upInCall)+'
(='+str(n_in)+'/'+str(n_simulation))
        pl.title('Up-and-out and up-and-in parity (# of simulations = %d ' %
n_simulation +')')
        pl.xlabel('Total number of steps ='+str(int(n_steps)))
        pl.ylabel('stock price')
        pl.show()
```

References

Wystup, U. (2002). *Ensuring Efficient Hedging of Barrier Options.* Frankfurt: commerzbank Trasury and Financial products.

Stoklosa, J. (2007). *Studies of Barrier Options and their Sensitivities*. The University of Melbourne.

Reiner ,E. and Rubinstein, M. (1991). *Breaking down the barriers.* Risk, vol4.pp. 28–35.

Roman, J.R.M. (2014) Lecture notes in Analytical Finance I

Yuxing Yan (2014). Python for Finance. Birmingham. Packt Publishing Ltd.