MÄLARDALENS HÖGSKOLA
ESKILSTUNA VÄSTERÅS

Pricing Barrier Options Using Monte Carlo Simulation

Pricing Options with Python

Submitted by:
Augustine Y. D. Farley
Ahmad Ahmad
Programme: Financial Engineering

Submitted to:
Jan Roman
Lecturer (Analytical Finance 1)
Mälardalen University

Contents

1.0 Introduction

This report is an assignment in the course Analytical Finance 1. The object of the assignment is to implement in Python a Monte-Carlo model to calculate the price for barrier options. In this report, we make an attempt to price both 'up and out' and 'up and in' barrier options. The Black-Scholes formula was used based on lecture notes from Analytical Finance 1.

2.0 Variables used in our model

We used seven variables in our model to price a European call barrier option. They are as follows:

Current Stock price: This is the price today for a share of a company's assets. It is represented in our model as 'S0'.

Strike price: The stated price per share for which underlying stock may be purchased (in the case of a call) or sold (in the case of a put) by the option holder upon exercise of the option contract. It is represented in our model as 'x'.

Barrier: The trigger point of an option that, which when crossed, the option gain or lose it's value. This is a characteristic of exotic options. This is represented as 'barrier'.

Time to maturity: The time for which a financial contract expires. It is represented in our model as 'T'.

Number of steps: The number of available changes or steps could be taken per a period of time equals to 1 unit of the time units t. And it will be represented as 'n_steps'.

Interest rate: This is the price paid for borrowing money. It is expressed as a percentage rate over a period of time. It is represented in our model as ‘r’.

Sigma: This is the measure of risk based on the standard deviation of the asset return. It is represented in our model as ‘sigma’.

3.0 The Black Scholes Formula

We used the Black Scholes formula in our model to calculate the price of the European call option. The value of a call option on a stock that pays no dividends is given by the following parameters:

$$C(S_t, t) = N(d_1)S_t - N(d_2)Ke^{-r(T-t)}$$
$$d_1 = \frac{1}{\sigma\sqrt{T-t}}\left[\ln\left(\frac{S_t}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)(T-t)\right]$$
$$d_2 = d_1 - \sigma\sqrt{T-t}$$

Where,

N (.) represents the cumulative distribution function of the standard normal distribution.

(T - t) represents the time to maturity.

(St) represents the spot price of the underlying asset

(K) represents the strike price

(r) represents the risk free rate or interest rate, and

(б) represents the volatility of returns of the underlying asset.

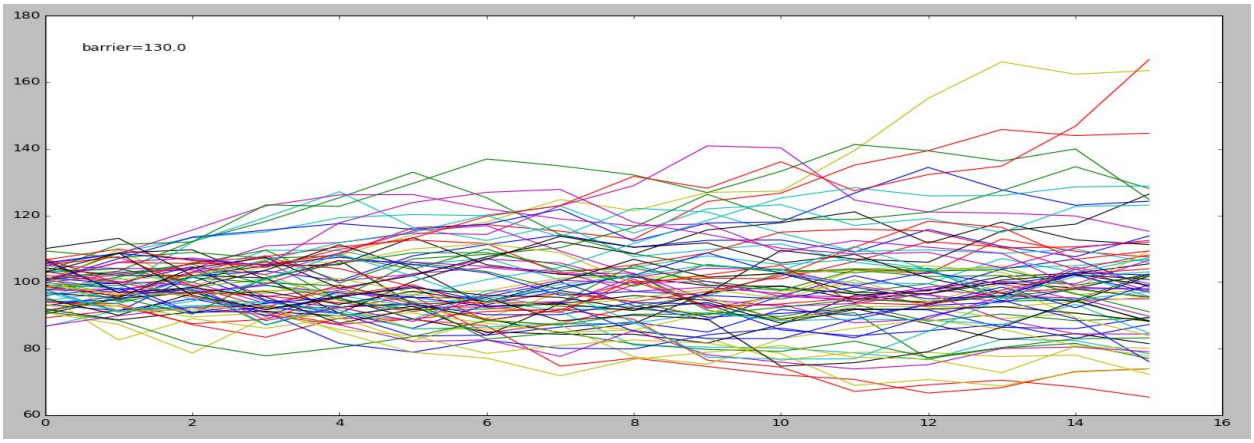
4.0 Coding for Python

The following is our coding used in Python to calculate the Monte Carlo calculation for European barrier options:

```
import matplotlib.pyplot as pl
import numpy as np
from numpy import *
import scipy as sp
import scipy.stats as stats


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
      Example 1:
       >>>bs_call(40,40,1,0.1,0.2)
       5.3078706338643578
    """

    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)

S0 = 100.0
x = 120.0
barrier = 130.0
T = 4
n_steps = 4
r = 0.05
sigma = 0.2

sp.random.seed(125)
n_simulation = 70
dt = T / n_steps
```

```
S = sp.zeros([int(n_steps)], dtype=float)
time_ = range(0, int(n_steps), 2)
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, 1)) * dt + sigma * sp.sqrt(dt) * e)
   if S[i + 1] > barrier:
      outStatus = False
      inStatus = True
   if outStatus == True:
      outTotal += c;
      n_out += 1
   else:
      inTotal += c;
      n_in += 1

   S = sp.zeros(int(n_steps)) + barrier
   upOutCall = round(outTotal / n_simulation, 3)
   upInCall = round(inTotal / n_simulation, 3)
   print 'up_and_out_call=' + str(upOutCall)
   print 'up_and_in_call=' + str(upInCall)
pl.figtext(0.15, 0.83, 'barrier=' + str(barrier))
k = (cumprod(1+random.randn(n_simulation, T*n_steps)*sigma/sqrt(T*n_steps),1)*S0)
for i in k: pl.plot(i)
pl.show()
```

Result from simulation



Up and Out_Call = 0.237
Up and In_Call = 0.0

5.0 References

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