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

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()
```