1. Portfolio management
2. Heard that this is an important course for quants.
3. Python(learned 2 semesters)

R(learned 2 semesters)

1. I really don’t know what this means.

(a). r is interest rate.

S0 is stock price at time 0.

β is relationship between volatility and price

σ is volatility

(b). The price is about $10. (Simulated by Python)

(c). The price is $9.95. It is the same as the result obtained via simulation. The two result should be the same. (Calculated by Python)

(d). The delta is 0.5497. (Calculated by Python)

(e). If consider the one unit of the call option means 100 shares.

I should buy -55 shares of stock to get a delta neutral portfolio.

(f). The estimation of the payoff the delta neutral portfolio is 987 which is very close to the call option cost calculated by BS model.

(x-100)+ - 0.55\*(x-100)-9.95>0

When the stock price is above 122.11 or below 81.90, the portfolio make money. When the stock price between 81.90 and 122.11, the portfolio lose money.

(g). The payoff is 99.59. Payoff is lower when beta is lower. Since beta is used to measure the systematic risk, for lower risk, the possible payoff will be lower.

(h). The payoff is 1567. Payoff is higher when sigma is higher. Since sigma is used to measure the stock volatility, for higher volatility, the possible payoff will be higher.

import numpy as np

import scipy.stats as si

S0=100

r=0.0

beta=1

sigma=0.4

T=1

K=100

def way():

way=[S0]

for i in range(252):

way.append(way[i]+way[i]\*r\*(1/252)+(way[i]\*\*beta)\*sigma\*np.random.normal(0,(1/252)\*\*0.5))

return way[252]

def b():

record=[]

temp=0

for i in range(100):

temp=way()

if (temp>=100):

record.append(temp-100)

else:

record.append(0)

print(record)

print(np.mean(record))

return np.mean(record)

def black\_scholes\_call(S0, K, T, r, sigma):

''' return put price calculated by black scholes model

'''

d1 = (np.log(S0 / K) + (r + 0.5 \* sigma \*\* 2) \* T) / (sigma \* np.sqrt(T))

d2 = (np.log(S0 / K) + (r - 0.5 \* sigma \*\* 2) \* T) / (sigma \* np.sqrt(T))

call = (S0 \* si.norm.cdf(d1, 0.0, 1.0)-K \* np.exp(-r \* T) \* si.norm.cdf(d2, 0.0, 1.0))

#print("delta= ",si.norm.cdf(d1, 0.0, 1.0))

return call

def delta(S0, K, T, r, sigma):

d1 = (np.log(S0 / K) + (r + 0.5 \* sigma \*\* 2) \* T) / (sigma \* np.sqrt(T))

return si.norm.cdf(d1, 0.0, 1.0)

def c():

#print(black\_scholes\_call(S0, K, T, r, sigma))

return black\_scholes\_call(S0, K, T, r, sigma)

def f():

payofflist=[]

for i in range(10000):

price=way()

if (price>=100):

payoff=price-100

else:

payoff=0

payoff=payoff-(price-100)

payofflist.append(payoff)

#print(payofflist)

#print(np.mean(payofflist))

return np.mean(payofflist)

c()

print(f())