CSC521 Final Project
Problem 2

Problem Descriptions

The first goal is to compute the average value and the value-at-risk (VAR) in one year of portfolio containing 5 stocks: AAPL, GOOG, MSFT, AMZN, and FB. The initial investment of portfolio is 5 million dollars, in which the we invest 1 million dollars in each stock. The second goal is to estimate the value of call option which will pay the investor 1 million dollars if after three months the 4 of 5 stocks in portfolio have a value below their current value. This paper will address the following questions:

- What is the daily log return and volatility of each stock?
- What is the yearly log return and volatility of each stock?
- What is the average portfolio value and 5% VAR in one year?
- For modeling future stock prices, which one is correct between Monte Carlo Simulation and Resampling?
- What is the estimated price of a call option with $1m payoff in three months if 4 of 5 stocks below their current values?

Dataset

The dataset contains 5-year daily prices of AAPL, GOOG, MSFT, AMZN, and FB stocks collected from May 25, 2013 to May 28, 2015 (1260 days or 252 trading days per year).

Solutions

- For estimating portfolio value and 5% VAR, we will use the daily prices to compute the average daily log return and volatility, as well as the average yearly log, and volatility. Then, we will use these data to simulate the stock prices to estimate the value of portfolio and 5% VAR in one year using Monte Carlo Simulation. We will simulate 1,000 times to model stocks and use Bootstrap to calculate the average and errors from our simulations.
- For pricing the call option, we will model the future stock prices using both Monte Carlo Simulation and resampling method, then we will compare the results of these methods.

Amy Aumpansub

Log Returns and Volatility

Python Codes listed in Appendix A

The log return assumes that the stock prices are normally distributed (Gaussian distribution). The daily log return of each stock is computed as the log of the current price divided by the previous price using the formula: $R_t = log\left(\frac{P_t}{P_{t-1}}\right)$ where P_t is stock price at time t and P_{t-1} is stock price at time t-1. After we find the log return of each day for each stock, we will find the mean and the volatility of daily log return where mean $\mu = \frac{1}{n}\sum x_i$ and volatility is $\sqrt{\sigma^2}$. For the annual log return, it is the sum of all daily log returns of 252 trading days. The annual log return and volatility are computed as $\mu_t = t \cdot \mu \quad and \; \sigma_t = \sqrt{t} \cdot \sigma$ where t is 252 trading days.

5-Year Historical Log Returns

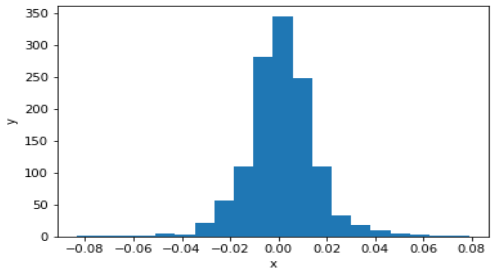

Figure 1: AAPL Daily Log Return

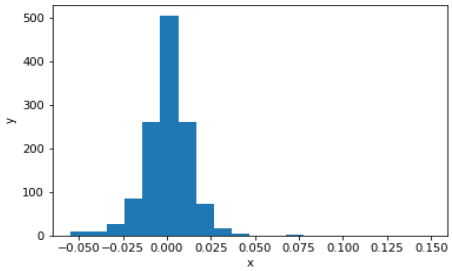

Figure 2: GOOG Daily Log Return

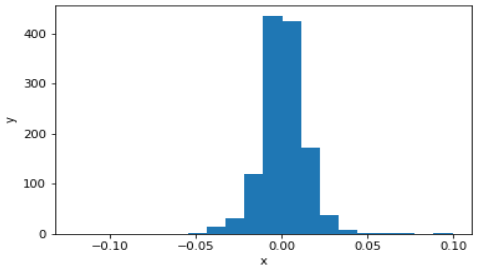

Figure 3: MSFT Daily Log Return

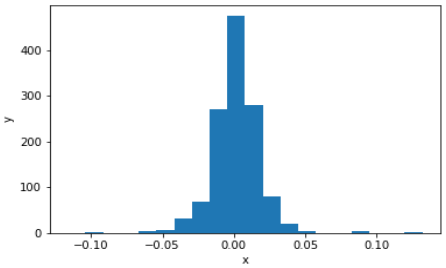

Figure 4: AMZN Daily Log Return

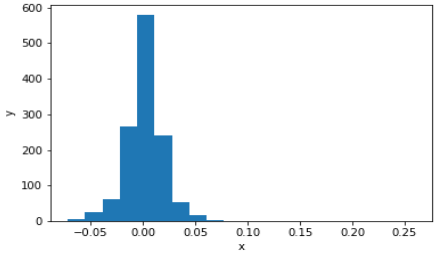

Figure 5: FB Daily Log Return

Log Returns and Volatility

Python Codes listed in Appendix A

The daily and yearly log returns and volatility are computed in Python using the function port_return and year_ret_std (Appendix A) which takes the adjusted daily closing price of each stock and date as inputs of function. The function, port_return returns the mean, daily log returns, and volatility of each stock. The function, year_ret_std return annual mean and volatility of each stock. The Python outputs are shown in the table. The Risk-return tradeoff theory states that the potential return increases with the increases in risk. Thus, the lower risk also associates with the lower return. The stocks are ranked by the highest return and the highest volatility as following: 1st FB, 2nd AMZN, 3rd AAPL, 4th MSFT, 5th GOOG.

	AAPL	GOOG	MSFT	AMZN	FB
Daily Log Return	0.105558882%	0.072900932%	0.092434690%	0.142921142%	0.165028526%
Daily Std	1.460185145%	1.418726499%	1.453958768%	1.825569461%	1.969286587%
Yearly Log Return	26.600838353%	18.371034837%	23.293541880%	36.016127681%	41.587188604%
Yearly Std	23.179720569%	22.521584976%	23.080879905%	28.980016773%	31.261455423%

From the table above, among 5 stocks, Facebook stock has the highest log return and volatility, indicating that Facebook stock is the most risky stock. Facebook has the daily log return of 0.16% and the annual log return of 41.58%. It has the daily volatility of 1.96% and 31.26%. The Google stock has the lowest daily log return of 0.07%, but it also has the lowest daily volatility of 1.41%, indicating that Google stock is the least risky stock compared to other stocks.

Amy Aumpansub

Future Stock Prices Simulation

Python Codes listed in Appendix B

To find the portfolio value in one year, we need simulate the future prices of each stock using Monte Carlo Simulation method, which randomly generates possible future scenarios. In Python, I wrote the function, Monte_once( ) which takes the inputs of the average log return and volatility of each stock to simulate the future log return of each stock. The function generates random numbers between 0 and 1, using R_t = random.gauss (mu, sigma) where mu is the average log return of each stock and sigma is the average volatility of each stock. Then, the future stock price (P_t) is calculated with the formula: $P_t = P_{t-1} \cdot e^{rt}$. Then, this function generates the series of future prices for the next 252 trading days of each stock as shown in the red line in following graphs. The blue line is the historical prices of stock in the last 5 years.

Future Stock Prices in One Year from One Simulated Scenario

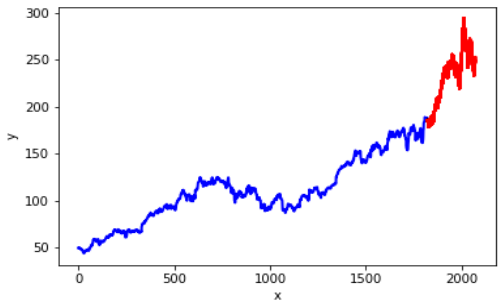

Figure 6: AAPL Stock

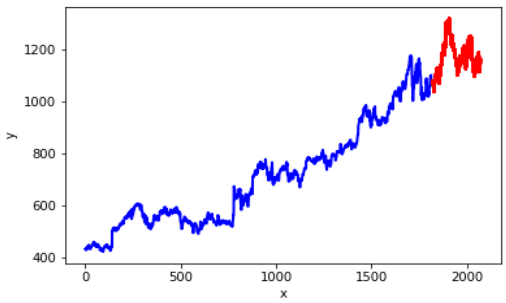

Figure 7: GOOG Stock

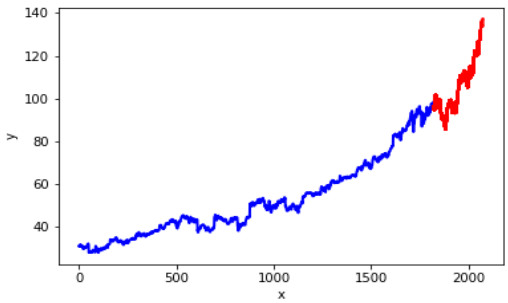

Figure 8: MSFT Stock

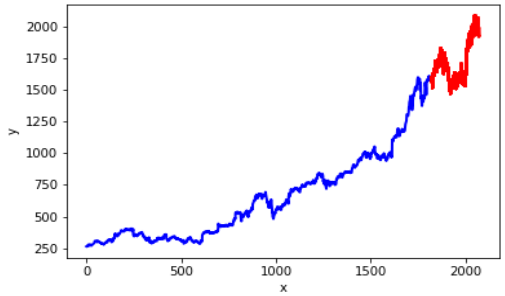

Figure 9: AMZN Stock

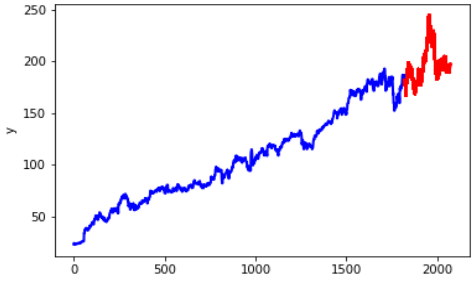

Figure 10: FB Stock

Portfolio Value in One Year

Python Codes listed in Appendix B

Monte Carlo Simulation method randomly generates possible future scenarios. After the function, Monte_once( ) simulates the future stock prices in one year. The function returns the value of portfolio in one year, in which the portfolio has an initial investment of 1 million dollars in each stock or 5 million dollars in total. The number of each stocks that we initially invested are calculated by dividing $1 million with the current price. After we know the number of stocks that we hold in portfolio, the value of portfolio in one year is calculated as a sum of the number of stocks of each firm multiplied by the future price in one year. The function, Monte_once( ) returns the future value of portfolio in one year for one simulation.

To compute the average value of portfolio in one year, I wrote the function, "Monte_many" which generates 1000 scenarios and return the average value of portfolio and the bound of results. The function "Monte_many" contains the Bootstrap function from nlib library which computes the error of results with 68% confidence level. The figure 6 shows the portfolio value in one year from 1000 simulations. The distribution is normally distributed. The average value of portfolio in one year is $ $6,880,948.230091944 which falls within the bound of $6,708,140.099286736 and $7,093,681.946693824 (see figure 11 on the next page).

5% Value at Risk in One Year

Python Codes listed in Appendix B

After we generated the series of future prices and compute portfolio value in one year, the portfolio values from 1,000 simulations are plotted on a histogram. We can calculate the 5% Value at Risk which is the value of portfolio in which no more than 95% from our simulation will fall below this value. In Python, I wrote the function "VAR_monte" which computes 5% VAR from our 1000 simulations. The result shows that 5% VAR is $5,762,910 .19506. This means from 1,000 simulations, no more than 950 simulations will have the portfolio value falling below $5,762,910 .19506 (see figure 11 on the next page).

Amy Aumpansub

1000 Simulations

Python Codes listed in Appendix B

Portfolio Value in One Year: $6,880,948 .23

Bound with 68% confidence level: ($6,708,140 .09 , $7,093,681 .94)

5% VAR: $ 5,762,910 .19

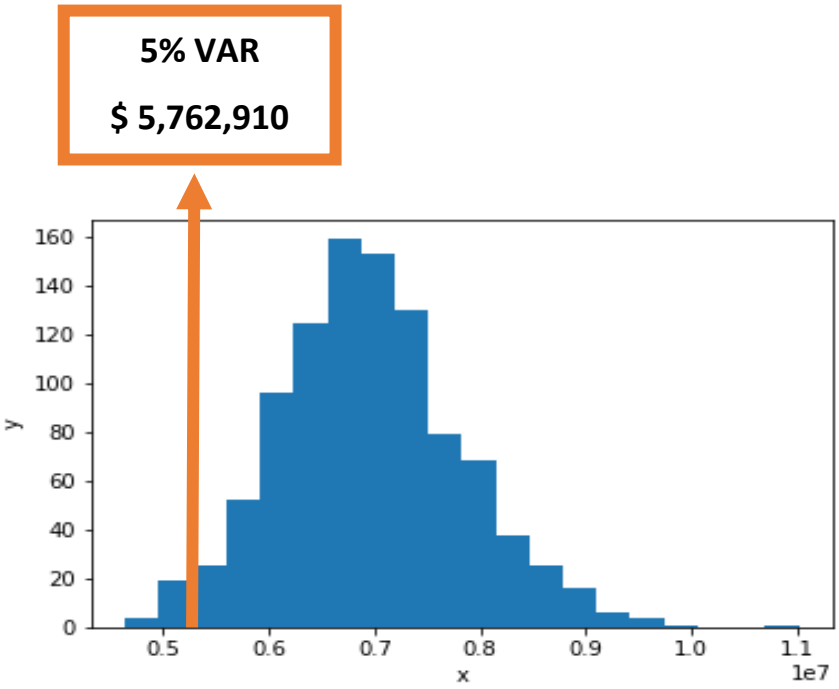

Figure 11: 5% VAR from 1000 Simulations

Portfolio Value in One Year (in $10 millions)

Call Option Price

Python Codes listed in Appendix C

The call option price is computed using data of future stock prices. We model the future stock prices using Monte Carlo Simulation and Resampling method. The call option has $1 million payoff, which will pay the option holder if the 4 of 5 stocks in portfolio has their value in three months falling below their current value (AAPL 188.149994, GOOG 1079.23999, MSFT 98.309998, AMZN 1603.069946, FB 185.929993). In Python, I wrote the class Option_Value which contains the simulate_one, simulate_many, and payoff functions. The expiration date is 90 days (T). The payoff function takes the list of future stock prices and return $1,000,000 if the simulated future price (in 3 months) of 4 from 5 stocks in portfolio falls below their currents value. The payoff function will return 0.0 if the condition is not met. The payoff at the next three months is discounted to the present value using the annual treasury yield of 2.6% and the discount factor is calculated as: $discount = e^{-rT} = e^{-\frac{0.026}{365} \cdot 90\ days}$

Monte Carlo Simulation

In this problem, we will generate future returns and prices for the next 90 days (T). Using Monte Carlo Simulation method, it randomly generates possible future log returns between 0 and 1 for each stock in each day: R_t = random.gauss (mu, sigma), where "mu" is the average daily log return and "sigma" is the daily volatility of each stock to simulate the future log return of each stock. Then, the future stock price (P_t) is calculated with formula: $P_t = P_{t-1} \cdot e^{rt}$. Then, this function generates the series of future prices of each stock for the next 90 days and return the discounted payoff of call option.

Resampling Method

The resampling method generates a sample by drawing a random sample from the population. For this problem, we will randomly pick the date from the past in the original data of historical stock prices. Then, we will use all daily log returns of 5 stocks in that picked (random) date as the future return of 5 stocks. Then, the future stock price (P_t) is calculated with the formula: $P_t = P_{t-1} \cdot e^{rt}$. The function simulate_once for resampling generates the list of future prices for 90 days and return the discounted payoff of call option.

Call Option Price

Python Codes listed in Appendix C

The current stock price is AAPL $188.149994, GOOG $1079.23999, MSFT $98.309998, AMZN $1603.069946, FB $185.929993 (as of May 25, 2018). The call option price that pays $1 million if the price in three months of any 4 of 5 stocks in portfolio are below their current value. The simulate_many function implements Monte Carlo simulation and resampling for 1,000 times and compute average and error of discounted payoff with Bootstrap function.

The price of this call option from Monte Carlo simulation is **$22,875.89.**
Whereas, resampling is correct, which the call option price from resampling **is $64,680.95.**

Call Option Price (1,000 simulations)

	Lower Bound	Average	Upper Bound
Monte Carlo	$ 18,897 . 4788803872	**$ 22,875 . 895486784**	$ 26,854 . 31209318191
Resampling	$ 58,488 . 0966295409	**$ 64,680 . 953919727**	$ 70,873 . 81120991426

Figure 12: Call Option Price (T = 90 days)

The resampling method is correct and should be used to model the stocks to compute the price of call option. It is faster than Monte Carlo Simulation and is easy to model the stocks. Moreover, the Resampling method takes into account the dependence of each stock including the correlation among stocks in portfolio because the future log return of each stock is picked from the historical return on the same date. On the other hand, the Monte Carlo simulation generates the future stock prices for each stock separately, so it does not capture the dependence and correlation among 5 stocks. Thus, resampling method is a correct method and the price of call option that pays you $1 million if the value of 4 from 5 stocks in three months are below their current value is $64,680 .95 (figure 12 and codes in Appendix C).

Conclusion

We use the historical daily prices of 5 stocks including AAPL, GOOG, MSFT, AMZN, and FB to compute the daily and yearly log return and volatility of each stock. Then, we assume that we invest in $5 million portfolio containing $ 1 million value of each stock at the current date (T= 0). Then, we implement Monte Carlo Simulation with 1,000 simulations to calculate the average portfolio value and 5% value at risk in one year. To price the call option, we model the future stock prices with Monte Carlo and Resampling method. The Resampling method is a correct procedure to use in pricing the stocks because it takes into account the dependence and correlations among stocks in portfolio.

Appendix A: Log Return and Volatility

```
import csv
import datetime
import math
from nlib import *

#Get the mu, sigma, log returns, historical stock price for each firm from combined csv. file
def Port_return(filename = 'combined_stocks.csv',
             start_date = datetime.datetime(2013,5,28),
             end_date = datetime.datetime(2018,5,25)):
    now = datetime.datetime.now()
    history = []
    log_portfolio = []
    log_aapl=[]
    log_goog=[]
    log_msft=[]
    log_amzn=[]
    log_fb=[]
    with open(filename) as myfile:
        reader = csv.reader(myfile)
        header = reader.next()
        rows = [dict(zip(header,row)) for row in reader]
        rows.reverse()
        for k,row in enumerate(rows):
            date = datetime.datetime.strptime(row['Date'], '%m/%d/%Y')
            close1, close2, close3, close4, close5 =
            float(row['Adj_appl']),float(row['Adj_goog']),float(row['Adj_msft']),float(row['Adj_amzn']),
            float(row['Adj_fb'])

            if k>1:
                log_return1,log_return2,log_return3,log_return4,log_return5 = math.log(close1/previous1),
                math.log(close2/previous2),math.log(close3/previous3),math.log(close4/previous4),math.log(
                close5/previous5)
            else:
                log_return1,log_return2,log_return3,log_return4,log_return5 = 0.0,0.0,0.0,0.0,0.0
            if start_date < date < end_date:
                history.append([(date-start_date).days,close1, close2, close3, close4, close5])
                log_portfolio.append([log_return1,log_return2,log_return3,log_return4,log_return5])
                log_aapl.append(log_return1)
                log_goog.append(log_return2)
                log_msft.append(log_return3)
                log_amzn.append(log_return4)
                log_fb.append(log_return5)
            previous1,previous2,previous3,previous4,previous5 = close1, close2, close3, close4, close5

    mu1,mu2,mu3,mu4,mu5 = mean(log_aapl),mean(log_goog),mean(log_msft),mean(log_amzn),mean(log_fb)
    sigma1,sigma2,sigma3,sigma4,sigma5 = sd(log_aapl),sd(log_goog),sd(log_msft),sd(log_amzn),sd(log_fb)

    return  mu1,mu2,mu3,mu4,mu5,sigma1,sigma2,sigma3,sigma4,sigma5, log_portfolio, history, log_aapl,
    log_goog,log_msft, log_amzn, log_fb


#Get the mu, sigma, log returns, current stock price for each firm
mu1,mu2,mu3,mu4,mu5,sigma1,sigma2,sigma3,sigma4,sigma5, log_portfolio, hist_port, log_aapl,
log_goog,log_msft, log_amzn, log_fb = Port_return()
current_price=hist_port[-1]
current_day,S_aapl1,S_goog1,S_msft1,S_amzn1, S_fb1 = current_price


#Calculate annual log returns and volatility for 5 stocks
T =252
def year_ret_std(log_returns=log_aapl,T=T):
    yearly_return = mean(log_returns)*T
    yearly_std = sd(log_returns)*math.sqrt(T)
    return yearly_return,yearly_std
```

Appendix B: Portfolio Value and 5% VAR

```
#Get the number of stocks invested 1M for each firm
Inv=1000000.0
numS1,numS2,numS3,numS4,numS5= Inv/S_aapl1, Inv/S_goog1, Inv/S_msft1,Inv/S_amzn1 ,Inv/S_fb1
T =252

#Monte Carlo Simulation for stock prices in one year, then return portfolio value
def Monte_once():
        future = []
        P_aapl,P_goog,P_msft,P_amzn, P_fb = S_aapl1,S_goog1,S_msft1,S_amzn1, S_fb1
        for k in range(1,T+1):
            R_aapl =  random.gauss(mu1,sigma1)
            R_goog =  random.gauss(mu2,sigma2)
            R_msft =  random.gauss(mu3,sigma3)
            R_amzn =  random.gauss(mu4,sigma4)
            R_fb =  random.gauss(mu5,sigma5)

            P_aapl = P_aapl*exp(R_aapl)
            P_goog = P_goog*exp(R_goog)
            P_msft = P_msft*exp(R_msft)
            P_amzn = P_amzn*exp(R_amzn)
            P_fb = P_fb*exp(R_fb)
            future.append([ P_aapl,P_goog,P_msft,P_amzn, P_fb])
        F_appl, F_goog, F_msft, F_amzn,F_fb =future[-1]
        Port_value = numS1*F_appl + numS2*F_goog + numS3*F_msft + numS4*F_amzn + numS5*F_fb
        return Port_value

def Monte_many(ap=0.1, rp=0.1, ns=1000):
        results = []
        s1=s2=0.0
        convergence=False
        for k in xrange(1,ns):
            x = Monte_once()
            results.append(x)
            s1 += x
            s2 += x*x
            mu = float(s1)/k
            variance = float(s2)/k-mu*mu
            dmu = sqrt(variance/k)
            if k>10:
                if abs(dmu)<max(ap,abs(mu)*rp):
                    converence = True
                    break
        results.sort()
        return bootstrap(results)


# 5% VAR of portfolio value in one year (1000 simulations)
def VAR_monte():
    scenarios = []
    for k in range(1000):
        y = Monte_once()
        scenarios.append(y)
        scenarios.sort()
    print '5% VaR =',scenarios[int(1000*0.05)]
    #Canvas().hist(scenarios).save('scenariosMonte1000.png')
```

Appendix C: Call Option Price

```
class Option_Value(MCEngine):
    def __init__(self, parameters):
        self.parameters = parameters

# Monte Carlo
    def simulate_once(self):
            future = []
            P_aapl,P_goog,P_msft,P_amzn, P_fb = S_aapl1,S_goog1,S_msft1,S_amzn1, S_fb1
            T = self.parameters['expiration']
            for k in range(1,T+1):
                R_aapl =  random.gauss(self.parameters['mu1'],self.parameters['sigma1'])
                R_goog =  random.gauss(self.parameters['mu2'],self.parameters['sigma2'])
                R_msft =  random.gauss(self.parameters['mu3'],self.parameters['sigma3'])
                R_amzn =  random.gauss(self.parameters['mu4'],self.parameters['sigma4'])
                R_fb =  random.gauss(self.parameters['mu5'],self.parameters['sigma5'])

                P_aapl = P_aapl*exp(R_aapl)
                P_goog = P_goog*exp(R_goog)
                P_msft = P_msft*exp(R_msft)
                P_amzn = P_amzn*exp(R_amzn)
                P_fb = P_fb*exp(R_fb)
                future.append([ P_aapl,P_goog,P_msft,P_amzn, P_fb])
            discount = exp(-self.parameters['Rfree']/365*T)
            return self.payoff(future) * discount

# Resampling
    def simulate_once(self):
            future = []
            P_aapl,P_goog,P_msft,P_amzn, P_fb = S_aapl1,S_goog1,S_msft1,S_amzn1, S_fb1
            T = self.parameters['expiration']
            for t in range (1,T+1):
                Rt= random.choice(log_portfolio)
                R_aapl=Rt[0]
                R_goog = Rt[1]
                R_msft = Rt[2]
                R_amzn = Rt[3]
                R_fb   = Rt[4]
                P_aapl = P_aapl*exp(R_aapl)
                P_goog = P_goog*exp(R_goog)
                P_msft = P_msft*exp(R_msft)
                P_amzn = P_amzn*exp(R_amzn)
                P_fb   = P_fb*exp(R_fb)
                future.append([ P_aapl,P_goog,P_msft,P_amzn, P_fb])
            discount = exp(-self.parameters['Rfree']/365*T)
            return self.payoff(future) * discount

 # Option Payoff and conditions (payoff =1 m if 4 of 5 stocks below current value, S)
    def payoff(self,future):
        F_appl, F_goog, F_msft, F_amzn,F_fb =future[-1]
        if F_appl<S_aapl1 and F_goog<S_goog1 and F_msft<S_msft1 and F_amzn<S_amzn1 and F_fb>S_fb1:
            return 1000000.0
        elif F_appl<S_aapl1 and F_goog<S_goog1 and F_msft<S_msft1 and F_amzn>S_amzn1 and F_fb<S_fb1:
            return 1000000.0
        elif F_appl<S_aapl1 and F_goog<S_goog1 and F_msft>S_msft1 and F_amzn<S_amzn1 and F_fb<S_fb1:
            return 1000000.0
        elif F_appl<S_aapl1 and F_goog>S_goog1 and F_msft<S_msft1 and F_amzn<S_amzn1 and F_fb<S_fb1:
            return 1000000.0
        elif F_appl>S_aapl1 and F_goog<S_goog1 and F_msft<S_msft1 and F_amzn<S_amzn1 and F_fb<S_fb1:
            return 1000000.0
        else:
            return 0.0
pricer =
Option_Value({'mu1':mu1,'mu2':mu2,'mu3':mu3,'mu4':mu4,'mu5':mu5,'sigma1':sigma1,'sigma2':sigma2,'sigma3':s
igma3,'sigma4':sigma4,'sigma5':sigma5,'expiration':90,'Rfree':0.026})
print pricer.simulate_many(ap=0.1, rp=0.1, ns=1000)
```