WQU Master’s in Financial Engineering

**Course: Computational Finance**

**Group work Assignment Submission 3 M 7**

**Group members**:

[**Tewodros Ayalew (ttaayyll32@gmail.com)**](https://masters.wqu.org/user/view.php?id=2308&course=99)[**Nishan Agarwal (agarwalnishan5@gmail.com)**](https://masters.wqu.org/user/view.php?id=14514&course=99)[**Dhruv Agrawal (dhruva1@stanfordalumni.org)**](https://masters.wqu.org/user/view.php?id=21612&course=99)

**Nicholas Munu (**[**nicholasmunu@yahoo.com**](mailto:nicholasmunu@yahoo.com)**)**

**Akshay Kulkarni (akshaypk2012@gmail.com)**

**Contents**

**1. Introduction**

**2. Task 1**

**3. Task 2**

**4. Summary and Conclusion**

**5. References**

**1. Introduction**

In this project we aim to :

1. Using a sample size of 100000, jointly simulate LIBOR forward rates, stock paths, and counterparty firm values.
2. Calculate the one year discount factor which applies for each simulation, and use this to find first the value of the barrier call option for the jointly simulated stock and firm paths with no default risk, and then the value of the barrier call option with counterparty default risk.

With the given parameters in the assignment

**2. Task 1**

|  |
| --- |
| import matplotlib.pyplot as plt import numpy as np from scipy.stats import norm import math import random |

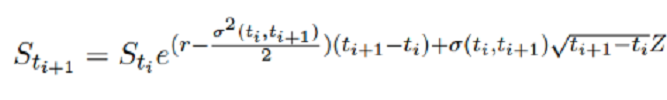
We import the necessary packages

|  |
| --- |
| def terminal\_value(St, r, sigma, Z,T, gamma):          sigmat = sigma\*((St)\*\*(gamma - 1))     return St\*np.exp((r  - sigmat\*\*2/2)\*T + sigmat\*np.sqrt(T)\*Z)  #Calculating next month's stock value using previous month's stock value,  #risk free rate, sigma,simulated matrix of N(0,1) random variables,  #time frame of 1 month (from the previos month), and gamma |

We need to define the above functions so that we can create a function (based on these functions) for the characteristic function. The function terminal\_value() is used to calculate the share price value for the underlying share using the value at the previous time step after updating the value of volatility for this time step.

The equation for volatility is given by



We return the stock value for the next month using the value in the previous month with the equation below :

|  |
| --- |
| def option\_payoff(S\_T,K):     return np.maximum(S\_T - K,0)  #The option\_payoff function finds the payoff for a call option at terminal  #time for given terminal stock price and strike price |

The function option\_payoff() defines the payoff for a call option at a terminal share price for a given strike price.

|  |
| --- |
| #Initializing the variables  T = 1               # Time to maturity of option strike = 100        # Strike price of option S0 = 100             #Initial stock value  BarrierLevel=150        #up and out barrier level for the option is 150 debt = 175              #Counterparty Firm debt recovery\_rate = 0.25#Recovery rate from counterparty in event of default correlation =  0.2          #Counterparty firm and stock correlation V0 = 200 # the firm value of the counterparty in the beginning is 200 sigma = 0.3             # Stock and firm volatility gamma = 0.75 delta = 1/12   #it is the time step difference (1 month or 1/12 years) between       #two bond prices of closest maturity dates bond\_prices = [99.38,98.76,98.15, 97.54, 96.94, 96.34, 95.74, 95.16, 94.57, 93.99, 93.42, 92.85] #zero coupon bond prices with maturities 1 month, 2 months, 3 months, .......12 months  M1r = np.log(100/bond\_prices[0])/delta #risk free rate for month 1   r = M1r    # dummy initial value for risk free rate  np.random.seed(0)       #fixing the seed for result replication  moption\_estimates = [None] #monte carlo option estimates not incorporating default risk moption\_std = [None]    #standard deviation of monte carlo option estimates cva\_estimates = [None] #cva estimates  cva\_std = [None]        # standard deviation of cva estimates CVAadjustedOptionval = [None] #Option value incorporating default risk CVAadjustedOptionval\_std = [None] #standard deviation of Option value incorporating default risk size=10000 prevstock\_val = S0 prevfirm\_val = V0  n\_steps = 12 #number of steps - different bond prices that are there (12 in this case) n\_simulations = 10000 # number of simulations is equal to sample size in this case  # The value of stock, firm, maxST ( maximum stock value for each simulation path)  # and option value without default risk of counterparty is initialized to zero  maxST = np.zeros(size)     stock\_val = np.zeros(size) firm\_val = np.zeros(size)                 option\_val = np.zeros(size) |
|  |

In the above code, we initialize the relevant variables for stock, firm value, and consequently option value simulation. We also fix the seed for random number generation to enable result replication.

|  |
| --- |
| #Initializing variables for the  predictor corrector method  mc\_forward = np.ones([n\_simulations,n\_steps-1])\* np.divide(np.subtract(bond\_prices[:-1],bond\_prices[1:]), np.multiply(delta,bond\_prices[1:]))  predcorr\_forward = np.ones([n\_simulations,n\_steps-1])\* np.divide(np.subtract(bond\_prices[:-1],bond\_prices[1:]), np.multiply(delta,bond\_prices[1:]))  predcorr\_capfac = np.ones([n\_simulations,n\_steps])  mc\_capfac = np.ones([n\_simulations,n\_steps])  Delta = np.ones([n\_simulations,n\_steps - 1])\*(1/12) #Vectorizing Delta  sigmaj =0.2 #Using value of sigmaj = 0.2 from notes |

In the above code we initialize the relevant variables for the predictor corrector method for simulating LIBOR forward rates.

|  |
| --- |
| for i in range(1,n\_steps):     Z = norm.rvs(size = [n\_simulations,1])          # Explicit Monte Carlo simulation     muhat = np.cumsum(Delta[:,i:]\*mc\_forward[:,i:]\*sigmaj\*\*2/(1+Delta[:,i:]\*mc\_forward[:,i:]),axis = 1)      mc\_forward[:,i:] = mc\_forward[:,i:]\*np.exp((muhat-sigmaj\*\*2/2)\*Delta[:,i:]+sigmaj\*np.sqrt(Delta[:,i:])\*Z)          # Predictor-Corrector Montecarlo simulation     mu\_initial = np.cumsum(Delta[:,i:]\*predcorr\_forward[:,i:]\*sigmaj\*\*2/(1+Delta[:,i:]\*predcorr\_forward[:,i:]),axis = 1)      for\_temp = predcorr\_forward[:,i:]\*np.exp((mu\_initial-sigmaj\*\*2/2)\*Delta[:,i:]+sigmaj\*np.sqrt(Delta[:,i:])\*Z)      mu\_term = np.cumsum(Delta[:,i:]\*for\_temp\*sigmaj\*\*2/(1+Delta[:,i:]\*for\_temp),axis = 1)     predcorr\_forward[:,i:] = predcorr\_forward[:,i:]\*np.exp((mu\_initial+mu\_term-sigmaj\*\*2)\*Delta[:,i:]/2+sigmaj\*np.sqrt(Delta[:,i:])\*Z) |

For each time step, we set our initial forward rates to the previous times’ forward rates, and create temporary simulated forward rates using our first drift estimate. We then use these new rates to create a second drift estimate. Finally, we use the average of these drift estimates to simulate the next time step forward rates from our previous forward rates.

The np.cumsum function returns the cumulative sum of the elements in a matrix. For example, if we have a vector, then the second element of the returned vector is the sum of the first and second element in the original vector, the third element of the returned vector is the sum of the first, second, and third elements in the original vector and so on. The axis = 1 command results in the np.cumsum function returning the cumulative sum over columns for each row in the matrix.

|  |
| --- |
| for month in range(1,12\*T+1):     #simulation from month 1 to month 12\*T                                          #where T is number of years (here T = 1)          if month == 1 :         r = M1r     else :         r = np.log(1+(predcorr\_forward[:, (month-2)])\*delta)/delta              corr\_matrix = np.array([[1,correlation],[correlation,1]])     norm\_matrix = norm.rvs(size = np.array([2,size]))     corr\_norm\_matrix = np.matmul(np.linalg.cholesky(corr\_matrix),                                           norm\_matrix)                  #The statement above creates a 2 × size array of standard      #normal random variables with a given correlation                               stock\_val = terminal\_value(prevstock\_val,r, sigma,corr\_norm\_matrix[0,],1/12.0, gamma)          #We calculate the stock's value for this month using previous month's stock      #value, risk free rate, sigma, simulated matrix of N(0,1) random     # variables, time frame of 1 month (from the previous month), and gamma                                     firm\_val = terminal\_value(prevfirm\_val,r, sigma, corr\_norm\_matrix[1,],1/12.0, gamma)                                         #We calculate the counterparty firm's value for this month using a similar      #mechanism that we employed for calculating the stock's value for this month       prevstock\_val = stock\_val              prevfirm\_val = firm\_val          #Values of stock and firm for the prior month are initialized          for j in range(0,size):             if(stock\_val[j] > maxST[j]):                     maxST[j] = stock\_val[j]          #updating maxST to the max value of the respective stock price path           #(in case this month's value is higher than the maxST on record) |

We will simulate paths for the underlying share using sample size of 100,000 over one year period for month over month. We set the relevant risk free rate to be be month 1 risk free rate if it’s the first month and the rti imputed from



(Where L(ti, ti+1) is the LIBOR forward rate calculated using predictor corrector method)

when its not the first month .

For each sample we will simulate the stock price for a year using monthly simulations. Storing the previous value of the stock price on this path (and using the random normal variable that has been generated for that particular path) we calculate the next period value of the stock price using the terminal\_value() function. A similar technique is utilized to simulate firm values. We also update maxST to the max value of the stock price path in each simulation (in case this month's value is higher than the maxST on record).

**3. Task 2**

|  |
| --- |
| DF = bond\_prices[12\*T-1]/100 #one year discount factor  M0M12r =  np.log(100/bond\_prices[12\*T-1])/T # calculating risk free rate continuously compounded from month 0 to month 12   r = M0M12r #updating r for calculation of option value and CVA |

We calculate the one year discount factor to be 0.9285 above

We calculate the continuously compounded risk free rate from month 0 to month 12 and set r equal to it for calculating CVA and option value.

|  |
| --- |
| for j in range(0,size):          if(maxST[j] > BarrierLevel):         option\_val[j] = 0           #if stock price has exceeded the barrier level then option value is 0         else:         option\_val[j] = np.exp(-r\*T)\*option\_payoff(stock\_val[j],strike)            #finds the discounted call payoff |

We go through each simulation, check if maxST has exceeded the barrier level and if so then set the option value to 0. Otherwise we set the option value to the discounted call payoff using the given terminal stock value and strike price.

This way, barrier option logic is implemented.

|  |
| --- |
| amount\_lost = np.exp(-r\*T)\*(1-recovery\_rate)\*(firm\_val < debt)\*option\_val  #estimates for CVA based on firm value  cva\_estimates = np.mean(amount\_lost) # We calculate the CVA estimate to be the mean of the amount\_lost across  # simulations and also calculate its standard deviation.  cva\_std = np.std(amount\_lost)/np.sqrt(size)      #default-free value of the option moption\_estimates = np.mean(option\_val) moption\_std = np.std(option\_val)/np.sqrt(size)      #Monte Carlo estimates for the price of the option incorporating counterparty risk, given by the default-free price less the CVA #We calculate CVA Adjusted option value to be the default free value of the option minus the CVA estimate CVAadjustedOptionval = moption\_estimates - cva\_estimates      CVAadjustedOptionval\_std = np.std(option\_val - amount\_lost)/np.sqrt(size) |

In amount\_lost we calculate the CVA estimate for that particular simulation using the formula

𝔼Q[𝑒−𝑟𝑇(1−δ)𝑋T𝕀{𝑉T<𝐷}]

Where r is risk free rate, T is time to maturity, 𝑋𝑇 is the value of our portfolio at time 𝑇(here portfolio is call option), and 𝕀{𝑉T<𝐷}] is an indicator which is 1 if value of firm at time T is less than debt and zero otherwise. (This is essentially our original CVA equation, but with all the τ's replaced by capital 𝑇's, since we are assuming that the only possible default time is capital 𝑇). We calculate the **CVA estimate** to be the mean of the amount\_lost across simulations and also calculate its standard deviation.

We calculate **the estimate of the default free value of the option** to be the mean of the default free value of the option for each simulation and also calculate its standard deviation.

We finally calculate the **CVA Adjusted option value** to be the default free value of the option minus the CVA estimate.

|  |
| --- |
| print("The CVA based on firm value is ",cva\_estimates)  print("The option value with no default risk is ",moption\_estimates)  print("The option value with counterparty default risk is ",CVAadjustedOptionval) |

Our final estimates are below:

1. CVA - 0.01927577225387991
2. Option value assuming no default risk of counterparty - 8.294279264008892
3. Option value assuming default risk of counterparty - 8.275003491755012

**4. Conclusion and Summary**

In this exercise we priced a call option considering time varying volatility and interest rates. We used LIBOR forward rate model to simulate interest rates and CEV model to simulate volatility.

We simulate the stock price and counterparty firm value assuming they do not have any correlation with the LIBOR rate.

The one year discount factor is 0.9285

Our final estimates are below:

1. CVA - 0.01927577225387991
2. Option value assuming no default risk of counterparty - 8.294279264008892
3. Option value assuming default risk of counterparty - 8.275003491755012

**5. References**

1)Boyd, D. W. (2001) *Model Calibration - an overview | ScienceDirect Topics*, *System Analysis and Modeling*. Available at: https://www.sciencedirect.com/topics/engineering/model-calibration (Accessed: 26 October 2019).

2) Brigo, D. (2007) ‘Interest Rate Models : Paradigm shifts in recent years’, *Theory and Practice*, 2001.

3) Geng, J. (2013) ‘Calibration of Local Volatility Models andProper Orthogonal Decomposition Reduced Order Modeling for Stochastic Volatility Models’.