Groupwork Assignment Submission 3 M7 **Prateek Chandra Jha**
MScFE 630 Computational Finance (C19-S2)
Dr. Sergio Garcia
Course #5 Worldquant University
...
...
...

Submission Date: 11/02/20
Due Date: 11/02/20

Introduction

The goal of the final submission is to consolidate all the concepts learnt in the previous submissions and bring forth all of them to use in repricing the risky up-and-out call option from Submission 1, but now implementing with a variable interest rate regime and local volatility effects.

The local volatility functions for both the stock and the counterparty have the same form as in part 2, namely $\sigma\left(t_i, t_{i+1}\right) = \sigma\left(S_{ti}\right)^{\gamma-1}$. For the stock $\sigma_S\left(t_i, t_{i+1}\right) = 0.3\left(S_{ti}\right)^{\gamma-1}$, and for the counterparty, $\sigma_V\left(t_i, t_{i+1}\right) = 0.3\left(V_{ti}i\right)^{\gamma-1}$, where $\gamma = 0.75$. We can simulate the next step in a share price path using the following formula:

$$S_{t_{i+1}} = S_{t_i} e^{\left(r - \frac{\sigma^2(t_i, t_{i+1})}{2}\right)(t_{i+1}-t_i)+\sigma(t_i,t_{i+1})\sqrt{t_{i+1}-t_i}Z}$$

where S_{ti} is the share price at time $t_i, \sigma\left(t_i, t_{i+1}\right)$ is the volatility for the period $\left[t_i, t_{i+1}\right], r_{ti}$ is the risk-free interest rate, and $Z \sim N(0,1)$. The counterparty firm values can be simulated similarly.

We are required to use a LIBOR forward rate model to simulate interest rates. The initial values for the LIBOR forward rates need to be calibrated to the market forward rates which can be deduced through the market zero-coupon bond prices which are mentioned in the table form in the project document. This continuously compounded interest rate for $\left[t_i, t_{i+1}\right]$ at time t_i, is then given by the solution to:

$$e^{r_{ti}(t_{i+1}-t_i)} = 1 + \mathrm{L}\left(t_i, t_{i+1}\right)\left(t_{i+1} - t_i\right)$$

Where $\mathrm{L}\left(t_i, t_{i+1}\right)$ is the LIBOR forward rate which applies from t_i to t_{i+1}, at time t_i. The note should be taken of the fact that these LIBOR rates are updated as we run through the simulation, and so our continuously compounded rates are updated accordingly.

Part I - LIBOR Forward Rates + Up & Out Call Option Prices Calculation

```
1
2 import pandas as pd
3 import numpy as np
4 import math
5 import random
6 from scipy.stats import norm,ncx2
7 import matplotlib.pyplot as plt
8
9 # Question requires us to make monthly simulations
```

```
10
11 t = np.linspace(0,12,13)
12
13 sigmaj = 0.20
14
15 # Setting up the simulation parameters like seed, # of simulations and step sizes
16
17 np.random.seed(0)
18 n_simulations = 100000
19 n_steps = len(t)
20
21 # You observe the following zero-coupon bond prices (per $100 nominal) in the market:
22
23 obs_bond_px = np.array
       ([100,99.38,98.76,98.15,97.54,96.94,96.34,95.74,95.16,94.57,93.99,93.42,92.85])
24
25 # Code for Calculation of forward rates with the given observed bond prices obs_bond_px
26
27 mc_forward = np.ones([n_simulations,n_steps-1])*(obs_bond_px[: -1] -obs_bond_px[1:])/(
       obs_bond_px[1:])
28
29 mc_capfac = np.ones([n_simulations,n_steps])
30
31 delta = np.ones([n_simulations,n_steps-1])*(t[1:]-t[:-1])
```

Listing 1: Code for Setting Up Monthly Simulations - Observed Bond Prices Array - Forward Rates Calibration with the Initially Observed Bond Prices

The next step is to define a working function that implements the Monte Carlo estimate for the LIBOR Forward Rates: $\hat{F}_j(t_i) = \hat{F}_j(t_{i-1})\, e^{\left[\left(\mu_j^1(t_i-1)-\frac{1}{2}\sigma^2\right)\delta_{i-1}+\sigma_j\sqrt{\partial_i-1}Z_i\right]}$ where:

$$\hat{\mu}_j(t_{i-1}) = \sum_{k=i}^{j} \frac{\delta_k \hat{F}_j(t_{i-1})\,\sigma_k\sigma_j}{1+\delta_k\hat{F}_j(t_{i-1})}$$

Finally in this section, to imply calculation of capitalization we define another equation that calculates the following below:

$$C(t_0,t_n) = \prod_{k=1}^{n}(1+\delta_k F_k(t_k))$$

The function is coded in the code listing below:

```
1
2 # Code that defines the function based on Forward-LIBOR Market Model (LFMM)
3
4 def LFMM(q,Z):
5     # Code below runs Monte Carlo Simulations
6     for i in range(1,n_steps):
7         muhat = np.cumsum(delta[q,i:]*mc_forward[q,i:]*sigmaj**2/(1+delta[q,i:]*
      mc_forward[q,i:]))
8         mc_forward[q,i:] = mc_forward[q,i:]*np.exp((muhat-sigmaj**2/2)*delta[q,i:]+
      sigmaj*np.sqrt(delta[q,i:])*Z[i-1])
9
10    # Using forward rates to calculate factors related to Capital
11
12    mc_capfac[q,1:] = np.cumprod(1+delta[q,:]*mc_forward[q,:])
13
14    # Calculating Bond prices (discount factors as is visible by usage of **(-1))
15
16    mc_price = mc_capfac[q,1:]**(-1)
17
18    return mc_price
```

Listing 2: Code that defines the function based on Forward-LIBOR Market Model (LFMM) - Using forward rates to calculate factors related to Capital - Calculating Bond prices (discount factors as is visible by usage of **(-1))

Now we have got to define a function that implements an Up & Out barrier call option with volatility effects. The Up & Out call option is a barrier option where the price starts from down a barrier level and it is knocked out once it reaches the barrier. The payoff of an up and out call option with maturity T, strike K and barrier B is given as follows:

$$C = (S_T - K)^+ \mathbb{I}_{\left\{\max_{0\leq t\leq T} S_t \leq B\right\}} \tag{1}$$

The following code given below implements the Up & Out Barrier call option with varying volatility.

```

# Code Describing and Calculating Up & Out Option Prices
# For that purpose, we define a separate function

def up_and_out_call(current_price, strike_price, delta, risk_free_rate, share_volatility
    , barrier,n_steps,e):
    out=False
    sT = current_price
    for i in range(n_steps-1):
        CEV_vol= share_volatility*sT**(gamma-1)
        sT*=np.exp((risk_free_rate[i]-0.5*CEV_vol**2)*delta[i]+CEV_vol*e[i]*np.sqrt(
    delta[i]))
        if sT > barrier:
            return 0
    if out==False:
        return np.exp(-np.sum(risk_free_rate * delta))*np.maximum(sT-strike_price,0)
```

Listing 3: Code Describing and Calculating Up & Out Option Prices - For that purpose we define a separate function

Part II - Discounted Factors Calculation

We begin this part by evaluating the Monte Carlo Estimates of default-free value of the Up & Out option for the firm using the same procedure we exploited for the stock in Part-I.

```

# Firstly we perform and calculate Monte Carlo Estimates of default-free price of
# the Up & Out Barrier Option
# For that purpose we set up the Share and Barrier Option Parameters below

share_volatility = 0.3
current_price = 100
strike_price = 100
T = 1
barrier = 150

# We need to set up the Firm level parameters in advance as well

sigma_firm = 0.3
debt = 175
recovery_rate = 0.25
v_0= 200
corr = 0.2

# We set up two arrays to store simulation results - one for Call Value and one for CVA
call_val = [None]*n_simulations
cva = [None]*n_simulations

gamma= 0.75

risk_free_rate= [None]*n_steps
disc_rate = np.ones([n_simulations,n_steps-1])

# Function Setting Up the Terminal Share Price Value in a similar way as Submission 2

```

```
def terminal_value(current_price, risk_free_rate, share_volatility, Z, delta,n_steps):
    sT= current_price
    for i in range(n_steps-1):
        CEV_vol= share_volatility*sT**(gamma-1)
        sT *= np.exp((risk_free_rate[i] - CEV_vol**2/2)*delta[i] + CEV_vol * np.sqrt(
    delta[i])*Z[i])
    return sT
```

Listing 4: Code for performing and calculating Monte Carlo Estimates of default-free price of the Up & Out Barrier Option - For that purpose we set up the Share and Barrier Option Parameters in the code - Function Setting Up the Terminal Share Price Value in a similar way as Submission 2 - Simply Up & Out Function for the Firm Level

```

# Here we assume that the forward rates simulated under LFMM are uncorrelated
# with both the counterparty share price as well as with the firm value processes/values

# Finally we conduct the Monte Carlo simulations

for m in range(0,n_simulations):

    # Random Variables Simulation
    Z_LF= norm.rvs(size=n_steps)

    # Rates Calculation
    disc_rate[m,:]= LFMM(m,Z_LF)
    risk_free_rate = -np.log(disc_rate[m,:])/(delta[m,:])

    # Correlation Matrices and Cholesky Decomposition
    correlation_matrix = np.array([[1, corr], [corr, 1]])
    norm_matrix = norm.rvs(size = np.array([2, n_steps]))
    corr_norm_matrix = np.matmul(np.linalg.cholesky(correlation_matrix), norm_matrix)

    # Final Call Values and CVA Values Calculation of Estimates
    call_val[m] = up_and_out_call(current_price, strike_price, delta[m,:],
                                  risk_free_rate, share_volatility, barrier,n_steps,
    corr_norm_matrix[0,])
    term_firm_val = terminal_value(v_0, risk_free_rate,sigma_firm , corr_norm_matrix
    [1,], delta[m,:], n_steps)
    cva[m] = call_val[m] *(term_firm_val <debt)*np.exp(-np.sum(risk_free_rate * delta[m
    ,:]))*(1-recovery_rate)
```

Listing 5: Code for Simulations using the Up & Out Function for the Firm Level and storing the estimates in the call_val and cva arrays

Part III - Calculating and Plotting the Calibrated LIBOR Forward Rate

```

mc_final = np.mean(np.column_stack((np.ones([n_simulations]),disc_rate)),axis = 0)*100
plt.xlabel("Maturity in months")
plt.ylabel("Bond Price")
plt.plot(t,obs_bond_px,'x', label = "Observed Bond Prices")
plt.plot(t,mc_final, label = "Simple Monte Carlo Bond Prices")
plt.legend()
plt.show()
```

Listing 6: Code for generating the final Plots of Observed and Monte Carlo Estimates of Bond Prices vs. Maturity in Months (Time)

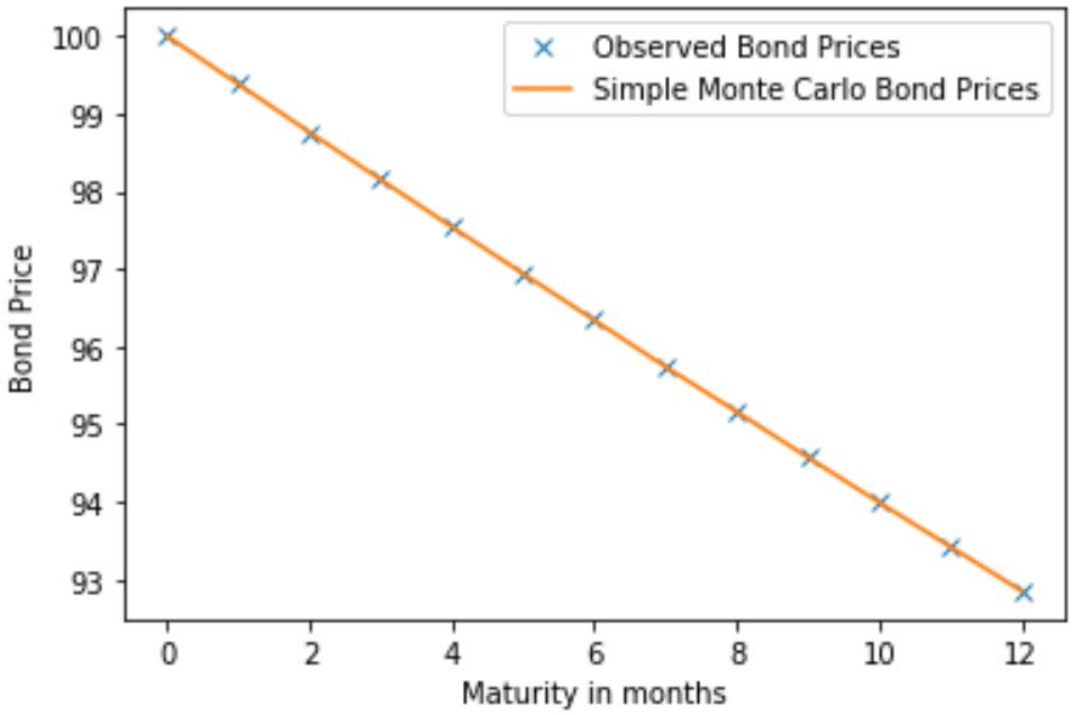

Figure 1: Plots of the Actual observed bond prices and the Monte Carlo estimates generated by Simulations in the Codes above

Part IV - Default Free Option Price vs. Adjusted for Credit Valuation Option Price

In the last part of the code, we hereby calculate the Default free Up & Out European call option price and the Option price which is adjusted for credit valuation (CVA):

```
1
2 # Let's have a look at default-free Option Prices vs. Prices adjuated for Credit
      Valuation
3 # Adjustment i.e. CVA
4
5 print("Default free Up and Out European call option price:", np.mean(call_val))
6
7 print("Option price adjusted for credit valuation:", (np.mean(call_val)-np.mean(cva)))
```

Listing 7: Code for generating the final look at default-free Option Prices vs. Prices adjuated for Credit Valuation Adjustment i.e. CVA

Final Calculations/Estimates are as follows:

- Default free Up and Out European call option price : 5.232916997381223 (as can be observed, this is higher than the CVA adjusted price we obtain below)
- Option price adjusted for credit valuation : 5.129199235399054 (The reason for a lower option price is the factoring of credit valuation which readjusts the option price)

References

[1] WQU Lecture Notes in Computational Finance (2020). [Online] Available at: `https://masters.wqu.org/course/view.php?id=158` [Accessed 02-11 Feb. 2020].

[2] Glasseman, P. (2013). Monte Carlo methods in financial engineering. Springer Science and Business Media. [Online PDF] [Accessed 04-07 Feb. 2020].

[3] Hilpisch, Y. (2016). Derivatives analytics with Python. Chichester: John Wiley and Sons Ltd. [Online PDF] [Accessed 05-08 Feb. 2020].

[4] Privault, N. (2014). Stochastic Finance: An Introduction with Market Examples. Oxfordshire: Chapman and Hall. [Online PDF] [Accessed 09-11 Feb. 2020].