Submission: 03
Group: 27
Members:

Name	Country	Mail	Member not contributing(X)
Vivek Verma	India	vivektheintel@gmail.com	
Christian Chihababo	Congo	aganzechihababo@gmail.com	
Harshil Sumra	India	harshilsumra1997@gmail.com	

Integrity Statement:

Vivek Verma, Christian Chihababo, Harshil Sumra

Member not contributing:

1. Jointly simulate LIBOR forward rates, stock paths and counter-party firm values using monthly iterations

The parameters used for the pricing:-

Symbol	Description	Value
T	Option Maturity	1.0
S_0	Current Stock price	100.0
K	Option Strike price	100.0
v (σ)	Volatility	0.30
r	Risk-free rate	0.08
B_{uo}	Barrier up and out level	150.0
S_{cp}	Current Counter party firm value	200.0
K_{cp}	Counter party debt	175.0
v_{cp}	Initial vol of counter party	0.30
corr_cp_stk	Correlation of counter party and stock	0.20
rr	Recovery rate with counter party	0.25
γ	Gamma	0.75

Local volatility functions for both the stk and the counter party have the same form:

$$\sigma(t_i, t_{i+1}) = \sigma(S_{ti})^{\gamma-1}$$

Share path simulation formula:

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

where, $Z \sim N(0,1)$

The Zero Coupon Bond prices per $100 nominal in the market:

Maturity	Price
1 month	$99.38
2 months	$98.76
3 months	$98.15
4 months	$97.54
5 months	$96.94
6 months	$96.34
7 months	$95.74
8 months	$95.16
9 months	$94.57
10 months	$93.99
11 months	$92.42
12 months	$92.85

The continuously compounded interest rate for $[t_i, t_{i+1}]$:

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

where, $L(t_i, t_{i+1})$ is the LIBOR forward rate.

Python version: 3.8.10
For the given parameters,

- The Price of the Vanilla Call Option turned out to be **$15.7113**
- Mean value of the stock path post the option time period: **$103.4869**
- Mean value of the counterparty firm value path post the option time period: **$207.0046**

The code snippets are as follows:-

```python
# option params
S0 = 100.0
v = 0.30
r = 0.08
T = 1.0
K = 100.0
B_uo = 150.0
months = 12 * int(T)
delta_t = T / months
gamma = 0.75

# counterparty firm params
S_cp = 200.0
v_cp = 0.30
K_cp = 175.0
corr_cp_stk = 0.20
rr = 0.25

np.random.seed(0)
```

0.5s

Figure 1: Initializing the variables

```python
def calc_d1(S0, K, r, vol, T):
    """Function to calculate d1"""
    return (np.log(S0 / K) + (r + .5 * vol ** 2) * T) / (vol * np.sqrt(T))


def calc_d2(d1, vol, T):
    """Function to calculate d2"""
    return d1 - vol * np.sqrt(T)


def calc_cdf(val):
    """Function to calculate cumulative distributive function using scipy norm pkg"""
    return norm.cdf(val)


def calc_price_call(S0, r, T, vol, K, verbose=False):
    """Function to calculate price of Call option"""
    d1 = calc_d1(S0, K, r, vol, T)
    d2 = calc_d2(d1, vol, T)
    if verbose: print(f"d1: {d1}; d2: {d2}")
    price = S0 * calc_cdf(d1) - K * np.exp(-r * T) * calc_cdf(d2)  # compute price using all input and variables
    if verbose: print(f"Call option price: {price}, with a standard deviation of {vol} for time {T} and S0={S0} & K={K}")
    return price


def calc_A(t1, t2, alpha):
    return (1 - np.exp(-alpha * (t2 - t1))) / alpha


def calc_D(t1, t2, alpha, b, sigma):
    val1 = (t2 - t1 - calc_A(t1, t2, alpha)) * (sigma ** 2 / (2 * alpha ** 2) - b)
    val2 = sigma ** 2 * calc_A(t1, t2, alpha) ** 2 / (4 * alpha)
    return val1 - val2


def calc_bond_price(r, t, T, alpha, b, sigma):
    return np.exp(-calc_A(t, T, alpha) * r + calc_D(t, T, alpha, b, sigma))


def calc_F(x):
    r0 = x[0]
    alpha = x[1]
    b = x[2]
    sigma = x[3]
    return np.sum(np.abs(calc_bond_price(r0, 0, maturity, alpha, b, sigma) - zcb_prices))
```

0.1s

Figure 2: Functions written for compute of Bond prices

```
1  _ = calc_price_call(S0, r, T, v, K, verbose=True)
2
```

✓ 0.1s

```
d1: 0.4166666666666667; d2: 0.1166666666666667
Call option price: 15.711312547892973, with a standard deviation of 0.3 for time 1.0 and S0=100.0 & K=100.0
```

Figure 3: Call option Analytical price output

```
1  zcb_prices = np.array([100.0, 99.38, 98.76, 98.15, 97.54, 96.94, 96.34, 95.74, 95.16, 94.57, 93.99, 93.42, 92.85]) / 100  # zero coupon bond prices
2  maturity = np.array([delta_t * month for month in range(months + 1)])  # computing maturities using the time step of delta_t over the time frame of months
3
4  bounds = ((0, 0.2), (0, 5), (0, 0.5), (0, 2))  # boundary parameters for the r, alpha, beta and sigma
5  opt_val = scipy.optimize.fmin_slsqp(calc_F, (0.05, 0.3, 0.05, 0.03), bounds=bounds)  # optimizing and minimizing the option value
6  r = opt_val[0]
7  alpha = opt_val[1]
8  beta = opt_val[2]
9  sigma = opt_val[3]
10
11 print(f"Optimized values computed => r={round(r, 5)}, alpha={round(alpha, 5)}, beta={round(beta, 5)}, sigma={round(sigma, 5)}")
12
```

✓ 0.1s

```
Optimization terminated successfully.    (Exit mode 0)
            Current function value: 0.00024383593342125476
            Iterations: 11
            Function evaluations: 86
            Gradient evaluations: 11
Optimized values computed => r=0.07491, alpha=0.27904, beta=0.07065, sigma=0.03642
```

Figure 4: Bond price initialization and optimization

```
1  model_prices = calc_bond_price(r, 0, maturity, alpha, beta, sigma)
2
3  simulations = 100000 # sample size defined in the question - 10^5
4
5  predcorr_forward = np.ones([simulations, months]) * (model_prices[:-1] - model_prices[1:]) / (delta_t * model_prices[1:])
6  predcorr_capfac = np.ones([simulations, months + 1])
7  delta = np.ones([simulations, months]) * delta_t
```

✓ 0.1s

```
1  # Running Predictor-Corrector Prices using Monte Carlo Simulation
2
3  for i in range(1, months): # simulating on monthly basis
4      Z = sigma * np.sqrt(delta[:, i:]) * norm.rvs(size=[simulations, 1]) # generating the random store
5
6      # Monte Carlo predictor-corrector
7      mu_initial = np.cumsum(delta[:, i:] * predcorr_forward[:, i:] * sigma ** 2 / (1 + delta[:, i:] * predcorr_forward[:, i:]), axis=1)
8      temp = predcorr_forward[:, i:] * np.exp((mu_initial - sigma ** 2 / 2) * delta[:, i:] + Z)
9      mu_term = np.cumsum(delta[:, i:] * temp * sigma ** 2 / (1 + delta[:, i:] * temp), axis=1)
10     predcorr_forward[:, i:] = predcorr_forward[:, i:] * np.exp((mu_initial + mu_term - sigma ** 2) * delta[:, i:] / 2 + Z)
11
12 predcorr_capfac[:, 1:] = np.cumprod(1 + delta * predcorr_forward, axis=1)
13 predcorr_price = predcorr_capfac ** (-1)
14 forward_rate = np.mean(predcorr_forward, axis=0)
15 predcorr_price = np.mean(predcorr_price, axis=0)
16 capfac = np.mean(predcorr_capfac, axis=0)
17
```

✓ 0.8s

Figure 5: Bond price calculation compute

```
plt.subplots(figsize=(15, 9))
plt.plot(maturity, zcb_prices, label = 'Actual Bond Prices')
plt.plot(maturity, model_prices, 'D', label = 'Calibration Prices')
plt.plot(maturity, predcorr_price, '+',label = "Predictor-Corrector Prices")
plt.xlabel('Maturity')
plt.ylabel('Bond Price')
plt.legend()
plt.show()
```

✓ 0.4s

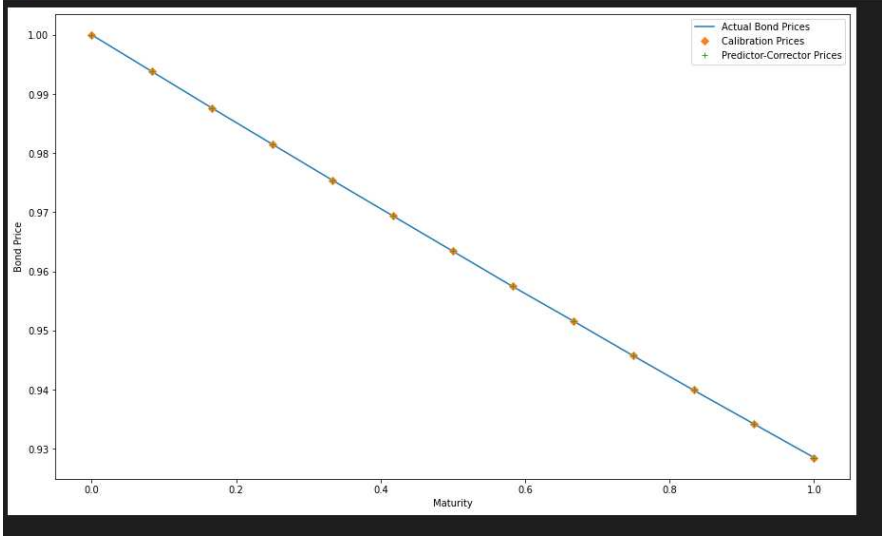


Figure 6: Plotting Bond prices

```python
def calc_vol_t_t_1(S_t, sigma, gamma):
    """Calculate the value of sigma(t, t+1)"""
    return sigma * S_t ** (gamma - 1)


def calc_price_t_1(S_t, r, sigma, gamma, delta_t, xi):
    """Calculate the value of price at t+1 using price at t"""
    sigma_t_t_1 = calc_vol_t_t_1(S_t, sigma, gamma)
    return S_t * np.exp((r - .5 * sigma_t_t_1 ** 2) * delta_t + sigma_t_t_1 * xi * np.sqrt(delta_t))


def calc_continuously_compounded_interest_rate(forward_rate, month, delta_t):
    """Calculate the continuously compounded interest rate using the forward rate, month and delta t"""
    return np.log(1 + forward_rate[month] * delta_t) / delta_t


def calc_combined_simulation(cycles=10):
    """Running the super combined simulations for cycles"""
    stock_paths = [None] * cycles
    firm_paths = [None] * cycles
    corr_matrix = np.linalg.cholesky(np.array([[1, corr_cp_stk], [corr_cp_stk, 1]]))

    for simulation_cycle in range(cycles): # Running through the cycles
        sim_cycle_stock_path = [None] * months
        sim_cycle_firm_path = [None] * months
        current_S = S0
        current_S_cp = S_cp

        for month in range(0, months): # Running through the entire length of option life in months step
            sim_cycle_stock_path[month] = current_S
            sim_cycle_firm_path[month] = current_S_cp
            xi = np.matmul(corr_matrix, norm.rvs(size=2))

            r = calc_continuously_compounded_interest_rate(forward_rate, month, delta_t)
            current_S = calc_price_t_1(current_S, r, v, gamma, delta_t, xi[0])
            current_S_cp = calc_price_t_1(current_S_cp, r, v_cp, gamma, delta_t, xi[1])

        # Saving the stock and firm path
        stock_paths[simulation_cycle] = sim_cycle_stock_path
        firm_paths[simulation_cycle] = sim_cycle_firm_path

    return stock_paths, firm_paths
```

✓ 0.1s

Figure 7: Functions written to simulate the stock and the counterparty firm value

```
# Firing the entire simulation
stock_paths, counterparty_firm_paths = calc_combined_simulation(cycles=100000)
```
✓ 1m 53.9s

```
print(f"Mean value of the stock path post the option time period: {np.mean(stock_paths)}")
print(f"Mean value of the counterparty firm value path post the option time period: {np.mean(counterparty_firm_paths)}")
```
✓ 0.4s

```
Mean value of the stock path post the option time period: 103.4869480839216
Mean value of the counterparty firm value path post the option time period: 207.00462645059602
```

Figure 8: Stock Counter party calculation compute

```
def plot_paths(paths, barrier_debt, text_barrier_debt, y_label, x_label, title):
    """Plotting the stock simulations"""

    months_to_plot = ["M1", "M2", "M3", "M4", "M5", "M6", "M7", "M8", "M9", "M10", "M11", "M12"]
    plt.subplots(figsize=(20, 10))
    paths = pd.DataFrame(paths, columns=months_to_plot)
    for _, row in paths.iterrows(): plt.plot(row, marker='o')

    plt.plot([barrier_debt] * len(months_to_plot), linestyle=':', color='r')
    plt.title(title)
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.grid(True)
    plt.show()
```
✓ 0.5s

Figure 9: Function to plot the graphs

```
plot_paths(stock_paths, B_uo, "Barrier Up-and-Out", "Price", "Months", f"Stock Price simulation, Barrier: {B_uo}")
```

✓ 7m 34.1s

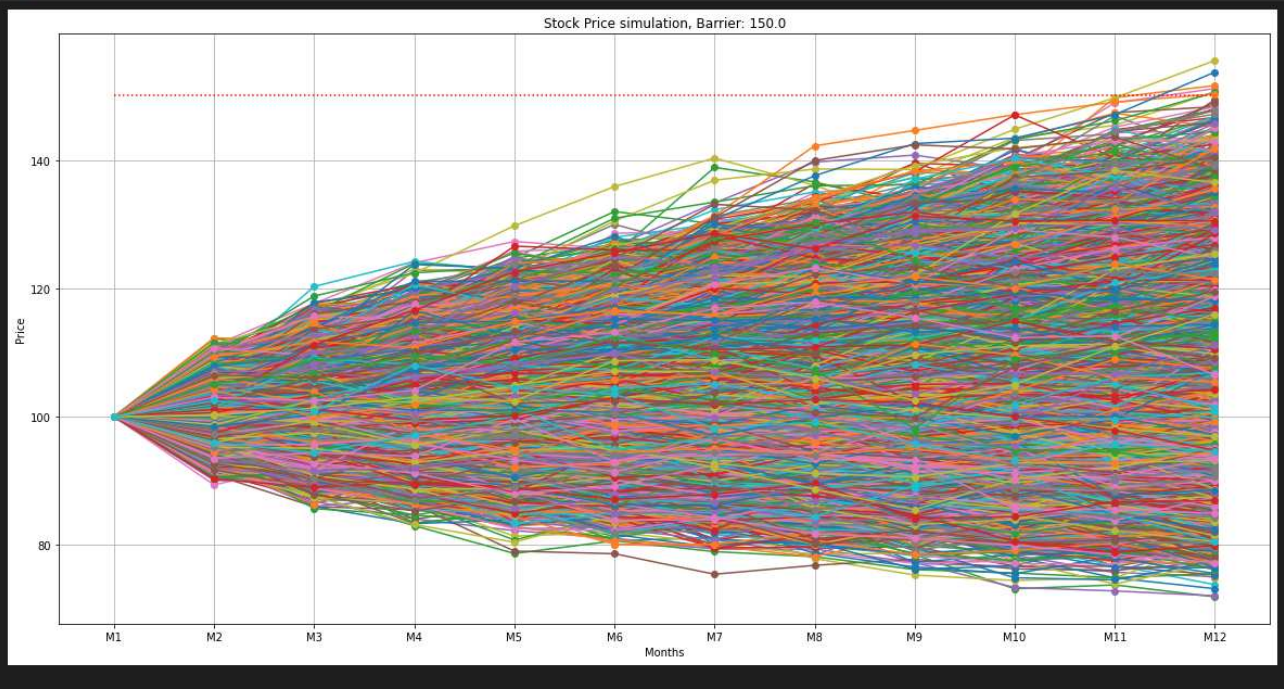


Figure 10: Plot of the stock price simulation

```
plot_paths(counterparty_firm_paths, K_cp, "Debt", "Price", "Months", f"Counter party Firm Price simulation, Debt: {K_cp}")
```

✓ 6m 31.3s

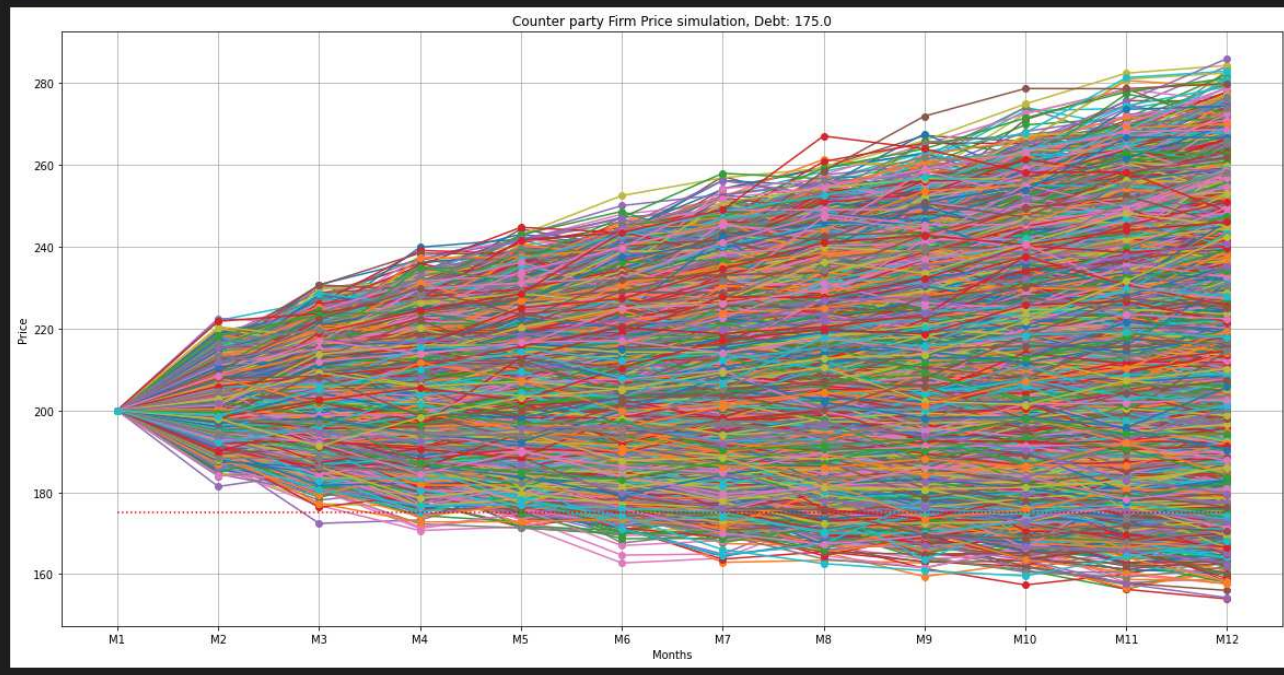


Figure 11: Plot of the counterparty firm value

$\therefore$ Concludes the Simulation of LIBOR forward rates, stock paths and counterparty firm values!

Ans.2.

a.

We first use our discount factors to calculate our continuously compounded interest rate. In the problem statement, we are given the following relationship:

$$e^{r(\Delta t)} = 1 + L\Delta t$$

Taking log on both sides and simplifying, we get:

$$r = \frac{\ln(1 + L\Delta t)}{\Delta t}$$

Here, L is forward rate and Δt is time difference between associated time points.

Code to do above is as follows:

```
In [14]: r_sim = np.log(1 + predcorr_forward*delta_t)/delta_t
         r_sim
```

```
Out[14]: array([[0.07485693, 0.07614214, 0.07563986, ..., 0.07617244, 0.07575892,
                 0.07601378],
                [0.07485693, 0.07506146, 0.07596397, ..., 0.07312996, 0.07361304,
                 0.07432558],
                [0.07485693, 0.07551802, 0.07529863, ..., 0.07306912, 0.0727583 ,
                 0.07312622],
                ...,
                [0.07485693, 0.07531839, 0.07573008, ..., 0.07713325, 0.07633884,
                 0.075565  ],
                [0.07485693, 0.07383794, 0.07293879, ..., 0.07605531, 0.07660264,
                 0.07606142],
                [0.07485693, 0.07374715, 0.07336699, ..., 0.07257093, 0.07347509,
                 0.07459168]])
```

b.

We use the above ‘r_sim’ to calculate the implied capitalisation factors and discount factors.

```
In [17]: #One year discount factors
         capfac_r_sim = np.cumprod(1 + delta_t*r_sim, axis = 1)
         discfac_r_sim = capfac_r_sim**(-1)

         discfac_1y = discfac_r_sim[:, -1]
         discfac_1y
```

```
Out[17]: array([0.92701989, 0.92824836, 0.92847842, ..., 0.92666555, 0.9280432 ,
                0.92922043])
```

We further extract our 1-year discount factors from the last column of discounted factor array/matrix.

c.

We now define a functions which takes ‘price path’, ‘discount factor’, ‘strike price’ and ‘barrier price’ as input to give us discounted payoffs as output.

In [21]:
```
#default-free UAO call value
def uao_call_disc_payoffs(price_paths, discfacs, strike, barrier):
    disc_payoffs = np.zeros([n, 2])
    price_paths = np.array(price_paths)
    path_no = -1

    for path in price_paths:
        path_no += 1
        if sum((path >= barrier)) == 0:
            disc_payoffs[path_no, 0] = np.maximum(path[-1] - strike, 0) * discfacs[path_no]
            disc_payoffs[path_no, 1] = 1

    return disc_payoffs

disc_payoffs = uao_call_disc_payoffs(stock_paths, discfac_1y, K, B_uo)
```

In above code, the 'for' loop aloows us to access one price path at a time and the 'if' condition helps us to check whether our price crosses the barrier followed by storing the discounted value of payoff in '0' indexed column and value '1' in the '1' indexed column to show whether the option is still active or not.

Then we use above output to calculate mean value and standard error for the cases when the '1' indexed column is '1', that is, the option is still active.

In [22]:
```
uao_call_meanval = np.mean(np.extract(disc_payoffs[:, 1] == 1, disc_payoffs[:, 0]))
uao_call_stderr = np.std(np.extract(disc_payoffs[:, 1] == 1, disc_payoffs[:, 0]))/np.sqrt(np.sum(disc_payoffs[:, 1]))

print('Default-free European UAO Call Value:', uao_call_meanval.round(3))
print('Default-free European UAO Call Std. Error:', uao_call_stderr.round(3))
```

```
Default-free European UAO Call Value: 7.677
Default-free European UAO Call Std. Error: 0.023
```

d.

To find the Default adjusted price, we first aim to find the Credit Valuation Adjustment (CVA) and then deduct it from default free price.

We know that our CVA is given as follows:

$$CVA = e^{-rt}(1-\delta)X_T\mathbb{I}_{[V_T<D]}$$

Where,

r= continuously compounded interest rate

T= Time to maturity

δ=retention rate

X_T= Option payoff at time 'T'

V_T= counterparty firm value at time 'T'

D= Debt due at time 'T'

We do above calculation in the code below,

```
In [25]: #CVA estimate
Sf_T = np.column_stack((np.array(counterparty_firm_paths)[:, -1], disc_payoffs[:, 1]))

#We only calculate the loss for those price paths where the call is activated, i.e. barrier is not breached
loss = (np.extract(disc_payoffs[:, 1] == 1, disc_payoffs[:, 0]) * (np.extract(Sf_T[:, 1] == 1, Sf_T[:, 0]) < Debt) *
        (1 - rr))

cva_meanval = np.mean(loss)
cva_stderror = np.std(loss)/np.sqrt(np.sum(disc_payoffs[:, 1]))

print('CVA Mean Value:', cva_meanval.round(3))
print('CVA Std. Error:', cva_stderror.round(3))
print(' ')
print('Default-adj UAO Call Value:', (uao_call_meanval - cva_meanval).round(3))
```

```
CVA Mean Value: 0.015
CVA Std. Error: 0.001

Default-adj UAO Call Value: 7.662
```

Summary output

European Up and Out call option	**Default free Value**	**CVA**	**Default adjusted value**
Mean	7.677	0.015	7.662
Standard error	0.023	0.001	

Ans.3.

In the answer number 2, we applied the concept of CVA, which is a measure of counterparty firms credit risk. Here, credit risk basically means the risk of the counterparty not being able to uphold the terms of the option contract. So, this CVA reduces the value of option contract to adjust for corresponding risk. If we observe carefully, we notice that since there are at least two parties involved in this transaction then either party could default. This is how the concept of Bilateral CVA came into the picture. It helps us define another adjustment 'debt value adjustment'(DVA) which is associated with our own credit risk. Since other party's credit risk leads to reduction of actual option price then our risk will lead to increase in the same.

That is,

Default Adjusted Option Price = Default Free Option price – CVA + DVA

Reference

\Mathematical Modeling and Computation in Finance: With Exercises and Python and MATLAB Computer Codes", by C.W. Oosterlee and L.A. Grzelak, World Scienti_c Publishing Europe Ltd, 2019.

Ans.4

The procedure for this solution is exactly same as answer number 2 post the required adjustment to the continuously compounded risk free interest rate.

```
In [26]: r_sim_plus25bps = r_sim + 0.0025
         r_sim_plus25bps
```

```
Out[26]: array([[0.07735693, 0.07864214, 0.07813986, ..., 0.07867244, 0.07825892,
                 0.07851378],
                [0.07735693, 0.07756146, 0.07846397, ..., 0.07562996, 0.07611304,
                 0.07682558],
                [0.07735693, 0.07801802, 0.07779863, ..., 0.07556912, 0.0752583 ,
                 0.07562622],
                ...,
                [0.07735693, 0.07781839, 0.07823008, ..., 0.07963325, 0.07883884,
                 0.078065  ],
                [0.07735693, 0.07633794, 0.07543879, ..., 0.07855531, 0.07910264,
                 0.07856142],
                [0.07735693, 0.07624715, 0.07586699, ..., 0.07507093, 0.07597509,
                 0.07709168]])
```

Discount rate calculation

```
In [27]: capfac_r_sim_plus25bps = np.cumprod(1 + delta_t*r_sim_plus25bps, axis = 1)
         discfac_r_sim_plus25bps = capfac_r_sim_plus25bps**(-1)

         discfac_1y_plus25bps = discfac_r_sim_plus25bps[:, -1]
         discfac_1y_plus25bps
```

```
Out[27]: array([0.92472003, 0.9259452 , 0.92617464, ..., 0.92436664, 0.92574058,
                0.92691465])
```

Default free prices and standard error calculation

```
In [28]: #default-free UAO call value post interest rate increase
         disc_payoffs_plus25bps = uao_call_disc_payoffs(stock_paths, discfac_1y_plus25bps, K, B_uo)
```

```
In [29]: uao_call_meanval_plus25bps = np.mean(np.extract(disc_payoffs_plus25bps[:, 1] == 1, disc_payoffs_plus25bps[:, 0]))
         uao_call_stderr_plus25bps = (np.std(np.extract(disc_payoffs_plus25bps[:, 1] == 1, disc_payoffs_plus25bps[:, 0]))/
                                      np.sqrt(np.sum(disc_payoffs_plus25bps[:, 1])))

         print('Default-free European UAO Call Value after Interest Rate Increase:', uao_call_meanval_plus25bps.round(3))
         print('Default-free European UAO Call Std. Error after Interest Rate Increase:', uao_call_stderr_plus25bps.round(3))
```

```
Default-free European UAO Call Value after Interest Rate Increase: 7.658
Default-free European UAO Call Std. Error after Interest Rate Increase: 0.023
```

Summary output

European Up and Out call option post 25 bps increase	Default free Value	CVA	Default adjusted value
Mean	7.658	0.015	7.643
Standard error	0.023	0.001	

In [30]:
```
#default-adjusted UAO call value post interest rate increase
#CVA estimate
Sf_T_plus25bps = np.column_stack((np.array(counterparty_firm_paths)[:, -1], disc_payoffs_plus25bps[:, 1]))

#We only calculate the loss for those price paths where the call is activated, i.e. barrier is not breached
loss_plus25bps = (np.extract(disc_payoffs_plus25bps[:, 1] == 1, disc_payoffs_plus25bps[:, 0]) *
                  (np.extract(Sf_T_plus25bps[:, 1] == 1, Sf_T_plus25bps[:, 0]) < Debt) * (1 - rr))

cva_meanval_plus25bps = np.mean(loss_plus25bps)
cva_stderror_plus25bps = np.std(loss_plus25bps)/np.sqrt(np.sum(disc_payoffs_plus25bps[:, 1]))

print('CVA Mean Value after Interest Rate Increase:', cva_meanval_plus25bps.round(3))
print('CVA Std. Error after Interest Rate Increase:', cva_stderror_plus25bps.round(3))
print(' ')
print('Default-adj UAO Call Value after Interest Rate Increase:', (uao_call_meanval_plus25bps -
                                                                   cva_meanval_plus25bps).round(3))
```

```
CVA Mean Value after Interest Rate Increase: 0.015
CVA Std. Error after Interest Rate Increase: 0.001

Default-adj UAO Call Value after Interest Rate Increase: 7.643
```

Summary of answer 2 and answer 4 output

QUESTION 5: Illustration and explanation

a. Differences between vanilla options and barrier options (payoffs and uses)

Vanilla Options

By definition, a vanilla option gives its holder the right, but not the obligation, to buy or sell the underlying asset at a predefined price within a given timeframe. These options are used by financial institutions and investors for hedging against exposure to a particular asset or for speculating on the expected price movement of a financial instrument. Also, vanilla options can be standardized if traded on an exchange.

For the next section, we will consider S to be our risky asset with a geometric Brownian motion.

$dS(t) = rS(t)dt + \sigma S(t)dW(t)$ with W being the standard Brownian motion under risk-neutral measure P.

Barrier Options

A barrier option is a type of derivative where the payoff depends on whether or not the underlying asset has reached or exceeded a predetermined price. This type of option can either be a knock-out (i.e. expiring worthless if the underlying goes beyond a certain price level, commonly called barrier level) or a knock-in (i.e. only valuable if the underlying reaches a given price). Compared to other options, barrier options are used for hedging positions. The payoff of barrier options depends on the price path of the underlying asset. One of the advantages of a barrier option is that it offers a lower premium with speculative risk. It should also be mentioned that barrier options are traded in the OTC (over-the-counter) market.

It should be mentioned that barrier options are similar to vanilla European options except for the barrier level which can leave the option non-existent if the barrier level is not reached. Specifically speaking, for knock-in barrier options, the down-and-in put option is given by:

$C_{d\&in}^{put} = (K - S_T)^+ 1S_{min_T} \leq B = \begin{cases} (K-S_T)^+, & if\ min_{te[0,T]} S_t \leq B \\ 0, & otherwise \end{cases}$ where K is the strike price, T: maturity and B the barrier level (B < K)

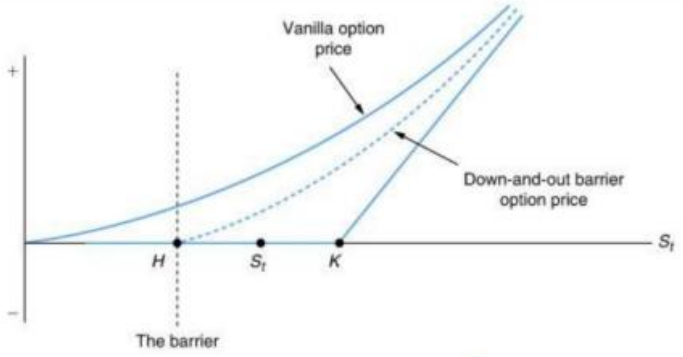
While for an up-and-out barrier call option with strike K, maturity T and barrier level B > K, we have:

$C_{d\&in}^{call} = (S_T - K)^+ 1S_{max_T} < B = \begin{cases} (S_T-K)^+, & if\ max_{te[0,T]} S_t < B \\ 0, & otherwise \end{cases}$

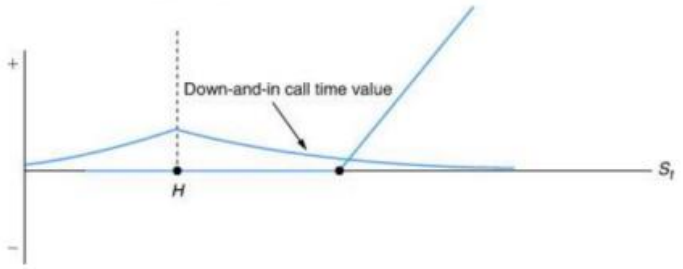
In summary, barrier options are cheaper than vanilla options given the risk related to speculation (getting knocked-out or never getting knocked-in). However, as long as the barrier option hasn't been knocked-out, or is still in, its benefits are the same as the ones offered by vanilla options. As mentioned earlier, barrier option payoff is dependent on the price path, on whether it reaches the pre-specified barrier level. Mathematically, the price of a vanilla option is the sum of a knock-in barrier option and the one of a knock-out option and is given by: $C = C_{in} + C_{out}$

The figures below give a graphical demonstration and comparison between vanilla options and barrier options.

The opposite figure shows that the knock-out barrier call option never reaches or never goes below the barrier level, denoted by H in the figure. It should be mentioned that 2 European barrier options were considered with strike price K, S_t, the underlying asset with the assumption that all Black-Scholes assumptions are met.



For the second opposite figure, the price of the knock-in barrier call option gives the standard payoff of a vanilla call if and only if S_t falls below H, the barrier level.



b. Differences between pricing out-of-the-money calls and at-the-money calls (volatility smile)

The term "volatility smile" refers to the effect of an increase in the implied volatility of an option as its strike price moves away from the price of the underlying asset, represented graphically as a curve relating the strike prices of a series of options with the same maturity and underlying asset to their respective implied volatility.

The graph resulting from plotting the strike price and implied volatility with the same underlying asset and maturity is named volatility smile because it has the shape of a smiling mouth. It's important to mention that not all options have a volatility smile applied to them. Also, implied volatility rises when the underlying asset of an option is further out of the money (OTM), or in the money (ITM), compared to at the money (ATM).

For the purpose of this assignment, we will focus on two components (ITM and ATM) of the smile while considering a call option:

1. ITM is a favorable state of the option where the market price of the underlying is higher than the strike price of the option (In the money, strike price < spot price). So, the difference between the market price and the strike price gives an intrinsic value to the call option:

$$C_{payoff} = (S_T - K)^+ > 0$$

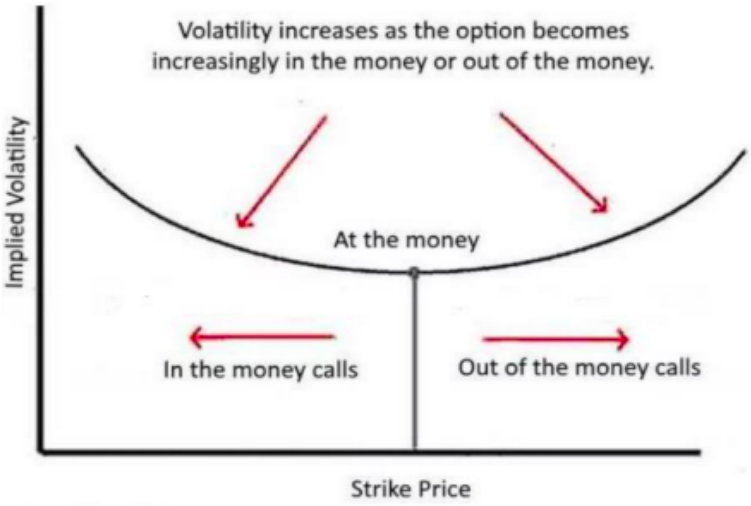
2. ATM only means that the market price is equal to the strike price of the underlying asset (At the money, strike price = spot price).

$$C_{payoff} = (S_T - K)^+ \approx 0$$

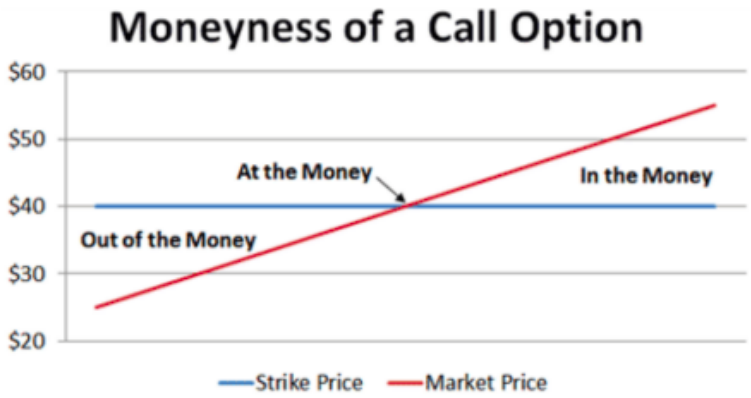
In terms of costs, the ATM option is cheaper than the ITM option as those investing are somehow willing to pay the profit related to the ITM contract.

The first opposite graph shows how volatility is at its lowest at-the-money while it increases as the option moves more and more in-the-money.

Looking at the vertical line in the graph: on the right side, compared to ATM holders, investors holding an ITM call option have a chance to buy shares below market prices and make some profit.



This second opposite graph shows where investors can profit on the volatility smile curve.



c. Differences between default-free option pricing and CVA option pricing

The default-free options pricing relies on risk-neutral pricing frameworks like the Black-Scholes, which assumes that funds are available at a risk-free rate and that traders can replicate those derivatives for hedging purposes (Macbeth & Merville, 1979). From the Black-Scholes, the fair value of the European Call and Put is derived as follows:

$$C = S_t e^{-\delta T}\Phi(d_1) - Ke^{-rT}\Phi(d_2) \quad \text{and} \quad P = Ke^{-rT}\Phi(-d_2) - S_t e^{-\delta T}\Phi(-d_1)$$

With risk-adjusted computed with a normal cumulative distribution:

$$d_1 = \frac{ln(\frac{S}{K})+(r-\delta+0.5\sigma^2)(T)}{\sigma\sqrt{T}} \text{ and } d_2 = d_1 - \sigma\sqrt{T}$$

Credit Valuation Adjustment (CVA) was introduced as a new requirement for fair value accounting during the 2007/08 Global Financial Crisis. Since its introduction, it has attracted dozens of derivatives market participants, and most of them have incorporated CVA in deal pricing.

the adjustment applied to the value of an option to account for the counterparty default risk. The CVA, therefore, allows the counterparty risk to be priced into the option. CVA is the market value of the counterparty's default risk and it is determined by subtracting the true risk-adjusted value from the risk-free value. The CVA is given by the mathematical approach as shown below:

Let L* be the actualized losses that can occur within the given time interval [0, T]:

$L^* = LGD \times EE \times PD$ with LGD: Loss Given Default, EPE: Expected Positive Exposure at time T, PD: Marginal Probability of Default

Having that, the formula of the Credit value Adjustment (CVA) is given by:

$CVA = E^{Q}[L * |t = \tau]$ where Q is the risk neutral measure and τ is the time when default will occur

By assuming that the recovery rate at default is known and constant, the credit exposure approach to find the CVA is given by:

$$CVA = LGD * \int_{0}^{T} E^{Q}[D(t) * EE(t)|t = \tau]dP(0,t)$$

The conditional expectation in the mathematical expression above highlights that market factors and credit factors jointly drive the potential default of the counterparty. It shows the existence of a correlation between the Bank exposure and the credit quality of the counterparty materialized by its expected default frequency.

Although the default-free option pricing is straightforward, the computation of the CVA is somewhat tricky as it requires the necessary market data, especially the expected exposure components and the probability of default, which might not be easy to find.

d. Differences between analytical-pricing methods and Monte Carlo simulation methods

In mathematical finance, option pricing is one of the critical concepts. Because of that, a lot of research has being done in order to find better pricing techniques and methods: analytical and numerical. For this assignment, we will consider the Black-Scholes Merton (BSM) model which is in fact an analytical pricing method and then compare it to Monte Carlo simulation methods (Heston, 1993).

The Black-Scholes assumptions for interpreting the relationship between the underlying asset price and the price of the option :

(1) no arbitrage opportunity.

(2) the underlying stock follows a geometric Brownian Motion, $dS = \mu Sdt + \sigma SdW$ with μ being the average growth rate of the underlying, σ: volatility, W: a Brownian motion

(3) Risk-free interest rate. Under the risk-neutral measure, we have, $dS_t = (r - \delta)S_t dt + S_t dW_t^Q$; with r: risk-free rate, δ: dividend yield, W_t^Q: Brownian motion under risk neutral measure.

The value of European Call option and Put option for a dividend-paying stock according to Black-Scholes parameters is as follows (Jiang, 2019):

$$C(S_t, t) = S_t e^{-\delta(T-t)}\Phi(d_1) - Ke^{-r(T-t)}\Phi(d_2) \quad \text{and}$$

$$P = Ke^{-r(T-t)}\Phi(-d_2) - S_t e^{-\delta(T-t)}\Phi(-d_1)$$

With risk-adjusted computed with a normal cumulative distribution:

$$d_1 = \frac{ln(\frac{S}{K})+(r-\delta+0.5\sigma^2)(T-t)}{\sigma\sqrt{(T-t)}} \text{ and } d_2 = d_1 - \sigma\sqrt{(T-t)}$$

Φ is the cumulative density function (CDF) of the standard normal distribution.

On the other hand, not all integrals have a closed-form solution; and in such cases, the implementation of numerical integration is an alternative, and Monte Carlo is one of the numerical integration methods. The Monte Carlo is a computerized mathematical technique that relies on random sampling to enable people to solve problems that contain random elements, e.g., perform risk analysis using quantitative methods. In option pricing, Monte Carlo

methods are used to calculate the value of an option with multiple sources of uncertainty; quite an advantage to the BSM. With Monte Carlo simulation, the price of any financial derivative can be expressed as an integral. The Central Limit Theorem together with the Law of Large Numbers are at the core of Monte Carlo's strength.

Consider the integral $\int_A g(y)f(y)dy = \bar{g}$ with $g(y)$: an arbitrary function, $f(y)$: the probability density function (PDF), and A: the range of integration: $\int_A f(y)dy = 1$

The estimate of $\bar{g}$, is given by $\bar{g} = \frac{1}{n}\sum_{i=1}^{n} g(y_i)$,
Where a number (n) of sampled values (y_i) is randomly picked from the PDF.

Because of its numerical nature, very few values are required. One of the benefits of Monte Carlo is its flexibility to handle distributions used to generate underlying stock's returns.

e. Differences among Black-Scholes volatility, Heston volatility, and local volatility

Black-Scholes volatility

The Black-Scholes Partial Differential Equation for an option price is given by:

$$\frac{\partial V(S_t,t)}{\partial t} + rS\frac{\partial V(S_t,t)}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V(S_t,t)}{\partial^2 S} - rV(S_t,t) = 0$$

In the Black-Scholes model, $dS_t = \mu S_t dt + \sigma S_t dW_t$, the constant σ is the spot volatility of S, and we can express σ as $\sigma^2 = \frac{d}{dt}[log(S)]_t$

The Black-Scholes formula implies an inverse function as a function of σ. You will find that for each option price there is a unique standard deviation (σ). Given the stock price in Black-Scholes would have random walks, the implied volatility which is derived from the Black-Scholes formula, is an estimate of the future variability for the asset underlying the options contract. Implied volatility is modelled as a function of the ratio of the option strike price to spot price. The realized volatility is based on past market observations whilst implied volatility is based on the current market prices. In the context of the volatility smile, we would observe an increasing implied volatility as the strike values move away from the spot price.

Heston volatility

Although the Black-Scholes assumes constant volatility, the real volatility of the underlying asset varies over time. When this volatility has its random-ness which depicts the Wiener process, it is said to be referred to as stochastic volatility.

The Heston model is a stochastic volatility model, that explains the evolution of the volatility of the underlying asset; it assumes that the spot price of the underlying asset is determined by a stochastic process $dS_t = \mu S_t dt + \sqrt{v_t} S_t dW_t^S$ with W_t^S: *a Wiener process*, v_t: the instantaneous variance which follows the Cox-Ingersoll-Ross process: $dv_t = k(\theta - v_t)dt + \xi\sqrt{v_t}dW_t^v$ where W_t^v: is a Wiener process and ξ: the volatility for determining the variance v_t

The fundamental Partial Differential Equation for the Heston model is given by:

$$\frac{1}{2}YS^2\frac{\partial^2 V}{\partial^2 S} + pvYS\frac{\partial^2 V}{\partial S \partial Y} + \frac{1}{2}v^2 Y\frac{\partial^2 V}{\partial^2 Y} - rS\frac{\partial V}{\partial S} + \Phi(S,V,t)\frac{\partial V}{\partial Y} - rV + \frac{\partial V}{\partial t} = 0$$

The Heston model appears to be flexible, provided it has the appropriate choice of parameters. With the Heston model, the volatility is mean-reverting and considers the leverage effect between asset returns and volatility.

The Black-Scholes formula produces option prices that are approximately identical to the Heston models for at-the-money options; understandably so because their volatility is similar at that point.

Local volatility

Although the Black-Scholes assumes constant volatility, the real volatility of the underlying asset varies over time. And in a case where such volatility is only a function of the underlying asset spot price and time, then we have local volatility. The local volatility refers to a set of diffusion coefficients, $\sigma_t = \sigma(S_t, t)$, consistent with the market prices of the underlying's options. This changes the Black-Scholes formula to be:

$$\frac{\partial V(S_t,t)}{\partial t} + rS\frac{\partial V(S_t,t)}{\partial S} + \frac{1}{2}\sigma^2 S^2\frac{\partial^2 V(S_t,t)}{\partial^2 S} - rV(S_t,t) = 0$$

The local volatility model calculates the volatilities for different combinations of strike prices (K) and expiry dates (T). So for each combination of the date, time, underlying asset spot price, local volatility is calculated for each option price to reflect the market price. The local volatility model can be considered as the generalisation of the Black-Scholes model wherein volatility remains constant. The Forward Kolmogorov formula shows that for a Stochastic Differential Equation: $dx_t = b(x,t)dt + \sigma(x,t)dW_t$, the transition probability satisfies the Partial Differential Equation:

$$\frac{\partial p(x,t;y,T)}{\partial T} = \frac{\partial(b(,T)p(x,t;y,T)}{\partial y} + \frac{1}{2}\frac{\partial^2(\sigma(y,T)p(x,t;y,T))}{\partial y^2}$$

Let $C(x,t; K, T)$ be the call option price at time t. Therefore, by Feynmann-Kac formula we have:

$$C(x,t;K,T) = e^{-rT}\int_K^\infty (y-K)p(x,t;y,T)dy$$

Let's say we know all market prices calls of all strike prices at time t, then we are able to determine the partial derivatives $C_K(x,t; K, T)$ and $C_{KK}(x,t; K, T)$, giving us the following: $p(x,t;y,T) = e^{-r(t-t)}C_{KK}(x,t;K,T)$; The $p(x,t; K, T)$ satisfies the Forward Kolmogorov formula:

$$\frac{\partial p(x,t;y,T)}{\partial T} = -\frac{\partial}{\partial y}(ryp(x,t;yT) + \frac{1}{2}\frac{\partial^2}{\partial y^2}(\sigma^2(y,T)p(x,t;y,T))$$

Differentiating the F-K formula gives us:

$$C_T(x,t;K,T) = -rC(x,t;K,T) + e^{-rT}\int_K^\infty (y-K)p_T(x,t;y,T)dy$$

And solving this formula, we get a unique solution, giving a local volatility model:

$$C_T(x,t;K,T) = -rC(x,t;K,T) + \frac{1}{2}\sigma^2(K,T)K^2 C_{KK}(x,t;K,T)$$

$$\sigma(K,T) = \sqrt{2\frac{C_T(x,t;K,T)+rKC_K(x,t;K,T)}{K^2 C_{KK}(x,t;K,T)}}$$

Done By

Part	Creator
1	Vivek
2, 3, 4	Harshil
5	Christian
Compilation	Vivek

References

[1] Chen, James. "Vanilla Option Definition." Investopedia, Investopedia, 21 Apr. 2022, https://www.investopedia.com/terms/v/vanillaoption.asp.

[2] Mitchell, Cory. "Volatility Smile Definition and Uses." Investopedia, Investopedia, 8 Feb. 2022. https://www.investopedia.com/terms/v/volatilitysmile.asp

[3] Chen, James. "What Is a Barrier Option?" Investopedia, Investopedia, 11 Apr. 2022, https://www.investopedia.com/terms/b/barrieroption.asp

[4] Heston, S. L., 1993. A Closed-form Solution for Options with Stochastic Volatility with Applications to Bond and Currency Options. The Review of Financial Studies, 6(2), pp. 327-343

[5] Jiang, Q., 2019. Comparison of Black–Scholes Model and Monte-Carlo Simulation on Stock Price Modeling. Jiangsu, China, Atlantis Press. 4. Macbeth, J. D. Merville, L. J., 1979. An Empirical Examination of the Black-Scholes Call Option Pricing Model. The Journal of Finance, Dec, 34(5), pp. 1173-1186.