APS 1022 – Week 1 Project

Option valuation project

Fendi Halim Tjoa (1007535017) & Yue Zhang (1007580823)

Instructor: Dr. Chi-Guhn Lee

Part 1. Introduction of Option Pricing

Option is a financial derivative that gives the right but not obligation to the option owner to buy or sell an underlying asset under specified terms. An underlying asset for this project is a stock. There are two basic types of options called call option and put option. A call option allows the option owner the right to purchase a stock at the exercise price (K) and by a certain date (Maturity). On the other hand, a put option owner can sell a stock at the exercise price and by a certain date. In order to prevent arbitrage, options will have a premium and is often called option price. Depending on the type of the options (e.g. American, European, Asian, etc.), the exercise time will vary and that will affect the price of the options.

Pricing a derivative such as option is challenging because the value of the option depends on the current stock price, changing price of the underlying asset, strike price, time to expiration, volatility, interest rate and option type. In this project, there would be two approaches in pricing options which are Monte Carlo Simulation and Lattice.

Part 2a. Monte Carlo Simulation

Monte Carlo Option Pricing simulates as many different pricing paths (following normal probability) as the user wants in order to estimate price of the stock in future times.

Table 1: Table of Known Variables

Known Variables	
Initial Price (**S**)	\$100
Maturity Time (**T**)	(2/12) years
Strike Price (**K**)	\$105
Risk-free rate (**r**)	2%
Volatility (**sigma**)	25%
Number of Simulation (**n_simulation**)	10,000 paths
Number of Steps (**n_step**)	8 steps(weeks)

The general equation to simulate a path of an asset price is:

$$S_{t_j} = S_{t_{j-1}} e^{\left(r-\frac{\sigma^2}{2}\right)dt+\sigma\sqrt{dt}\,Z_j}$$

$$Z_j \sim N(0,1)$$

$$j = 1, 2, 3, \dots, 8$$

where S_{t_j} denotes stock price at time t_j, dt denotes time step between each time point, Z_j denotes normally distributed random generated variable, r denotes risk free rate of market, σ denotes volatility of underlying stock.

The $\left(r - \frac{\sigma^2}{2}\right) dt$ part is also referred as drift, and the $\sigma\sqrt{dt}$ is referred as alpha.

Table 2: Variables for Path Simulation

Variables for Path Simulation	
$dt = \frac{T}{n\_step}$	1/48
$(r - \frac{\sigma^2}{2})dt$ **(drift)**	-0.000234375
$(\sigma\sqrt{dt})$ **(α)**	0.036084

Once all the variables are ready, stock prices (figure 2) can be simulated using Python and stored into 2-D Numpy arrays. Simulated price and path scenarios are visualized below.

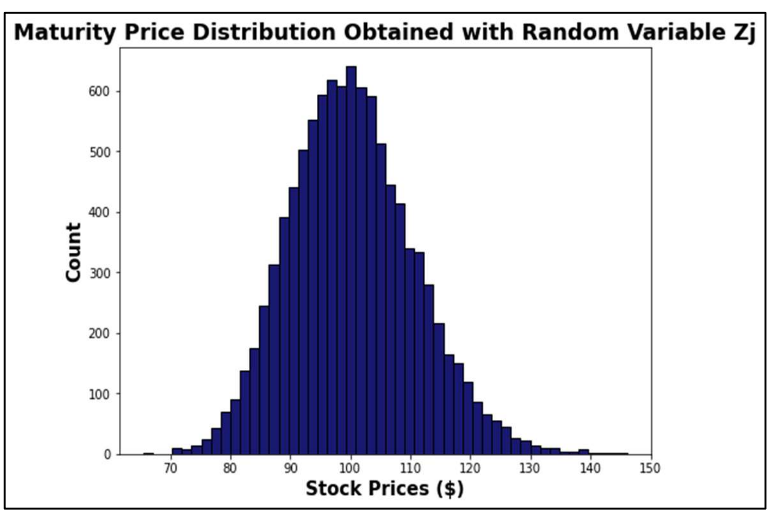


Figure 1: Z_j Random Samples from a Normal Distribution

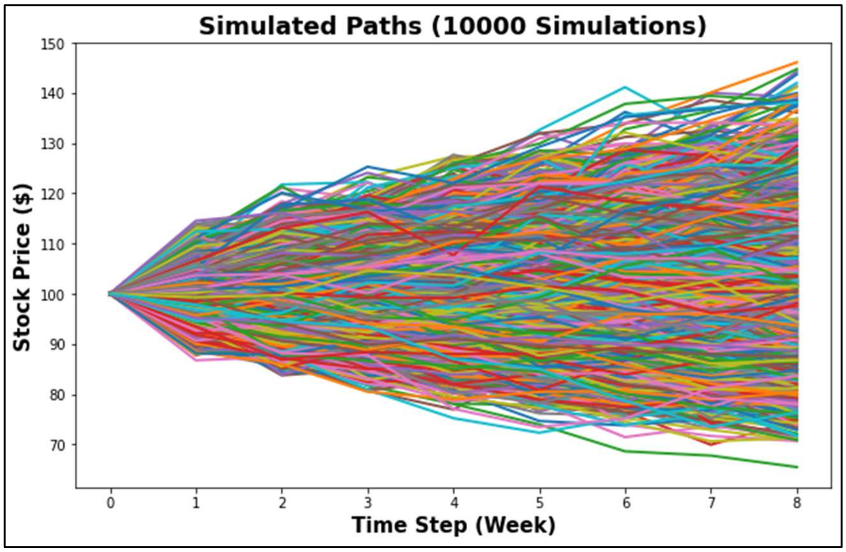


Figure 2: Simulated Stock Price Paths

With the simulated stock prices, different options could be priced according to these scenarios. There are six different types of Asian options that were priced using Monte Carlo simulation (table 3). $\bar{S}, S_{max}, S_{min}$ respectively denote the average, maximum, and minimum values of a particular stock price path over the set of times. By using the equation from table 3, these six option types can be priced accordingly. In Appendix, cell 7 represents the payoff calculation using the equation from table 3. Cell 8 represents the calculation of discounted price of each payoff, assuming discontinuously compounding. Cell 9 represents sample mean calculation of each option type and that will become option price of each option type. Cell 10 represents sample variance calculation, and they will be used to calculate confidence interval of each option price.

Table 3: Payoffs from Path-dependent Options

Option Type	Payoff
Asian Call	$(\bar{S} - K)^+$
Asian Put	$(K - \bar{S})^+$
Lookback Call	$(S_{max} - K)^+$
Lookback Put	$(K - S_{max})^+$
Floating Lookback Call	$(S_T - S_{min})^+$
Floating Lookback Put	$(S_{max} - S_T)^+$

Part 2b. Pricing American Put Option Using Monte Carlo Simulation

American put option pricing is trickier than the previous option types because it can be exercised at any time before the maturity date. The simulated stock price from previous part was adopted for pricing American put option. The binominal tree model (from Pricing American Options Using Monte Carlo Simulations by Nairn Mcwilliams) was used to calculate option price. For each simulated scenario, this method uses Black-Scholes model to calculate option price at time t, then compares the calculated price with payoff from early exercise at time t. If payoff from early exercise is greater, time t would be set at the optimal stopping time. However, if an optimal stopping time was not found for a path, set maturity time T and optimal stopping time. Once optimal stopping time for each path was found, the average option payoff could be calculated to estimated certain American put option.

The Black-Scholes formula used here for is:

$$\mathbb{E}[S_T \mid S_t] = \begin{cases} S_t N(d_1) - K e^{-r(T-t)} N(d_2) & \text{for a call option} \\ K e^{-r(T-t)} N(-d_2) - S_t N(-d_1) & \text{for a put option} \end{cases}$$

where

$$d_1 = \frac{\log(\frac{S_t}{K}) + (r + \frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{T-t}} \qquad d_2 = d_1 - \sigma\sqrt{T-t}$$

By following the binominal tree model method, the sample mean of the put option price, variance and 95% confidence intervals obtained will be shown under part 3.

Part 2c. Lattice Approach

Lattice approach in pricing is another method to determine the price of the option. Lattice approach is widely used because of its simplicity, easiness, and visibility in terms of calculation. At every stage, a stock price can go up and go down and movement is calculated by multiplying the current stock price with either up (u) or down (d) multiplier. By having up and down movement on each stage, there will be $2^8 = 256$ different paths since there are 8 periods (figure 3). To create lattice binomial stock price tree, *np.zeros* and *np.array* and the outcome can be depicted on cell output 20 in Appendix. Moreover, binomial trees need to be expanded in order to calculate $\bar{S}, S_{max}, and S_{min}$ per path (cell input and output 14 in Appendix). Payoff at final stages per path can be calculated using equation from table 3. That payoff calculation can also be seen on cell 16 in Appendix. Once they are established, payoff per price and stage can be calculated backwards (dynamic programming) using the algorithm of cell 17 in Appendix. The equation is:

$$C = (C_u * q + C_d * (1 - q)) * e^{-r\Delta t}$$

Where C_u is the up-payoff after one time step, C_d is the down-payoff after one time step, $\boldsymbol{q}$ is the risk neutral probability. The calculated price is then discounted to present value, assuming continuously compounding.

By doing this calculation from last stage all the way to the initial stage will generate price for given options. Parameters used for the lattice approach and the structured lattice path are shown in Table 4 and Figure 3 below.

Table 4: Variables for Lattice

Variables for Lattice	
$u = e^{\sigma\sqrt{dt}}$ up multiplier (**u**)	1.03674
$d = \frac{1}{u}$ down multiplier (**d**)	0.96456
$R = 1 + r\frac{2}{\frac{12}{8}}$ (**R**)	1.000417
$q = \frac{R - d}{u - d}$ Risk neutral probability (**q**)	0.496752

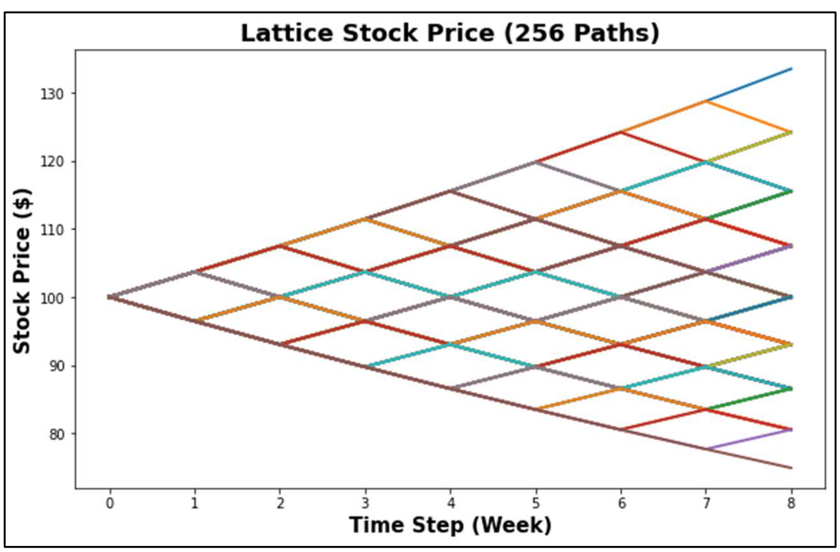


Figure 3: 8-period Binomial Lattice of Stock Price

Part 2d. Pricing American Put Option Using Lattice Approach

Pricing American put option using lattice follows the same principle as part 2c which the payoff is calculated from the last stage to the beginning of the stage (dynamic programming). Using the non-expanded binomial tree that looks like cell output cell 20 in Appendix, the first thing to do is to calculate the payoff at the final stage:

$$payoff_{S_8} = \max(K - S_8, 0)$$

Next, is to calculate the payoff per stage and per stock price using equation:

$$payoff_{stock\ pric\ \ k, stage_i} =$$

$$\left[payoff_{stock\ pric\ \ k, stage_{i+1}} * q + \ payoff_{stock\ price_{k+1}, stag\ \ i+1} * (1-q)\right] * \ e^{r*dt}$$

After that, there is a condition that need to be met because American option put can be exercised at any time. If the first condition is met, then follow the description of the condition. If not, check the next condition until the condition is met. Once one condition is met starting from the first condition, then stop checking next condition.

- **First Condition**: If the payoff is greater than or equal to zero, and if the strike price is greater than the current stock price, and the current stock price is not zero, then the payoff for that particular price and stage is $\max(payoff_{stock\ price_k, stage_i}, K - stockprice_k, stage_i)$.
- **Second Condition:** If the payoff is greater than or equal to zero, and if the strike price is less than the current stock price, and the current stock price is not zero, then the payoff for that stock price and stage remains the same.
- **Third Condition:** if none of the above is met, then the payoff for that stock price and stage is zero.

$(K - stockprice_k, stage_i)$ on the first condition denotes value of early exercise. In order to follow no arbitrage profit, the payoff that is the higher over the other will be the chosen payoff. If that is not followed, the arbitrage profit will be (higher payoff – lower payoff). All the algorithms above are explained on cell 21 in Appendix.

Part 3. Results and Discussion

Option prices calculated based on Monte Carlo simulation and Lattice approach were shown below.

Table 5: Option Pricing Results

Option Type	Lattice Approach Option Price ($)	Monte Carlo Approach Option Price ($)	Variance from Monte Carlo	95% Confidence Interval (Lower Bound & Upper Bound)
Asian Call	0.692490	0.712330	3.876655	0.673739 & 0.750921
Asian Put	5.509542	5.538709	21.638668	5.447535 & 5.629883
Lookback Call	3.574964	3.393235	29.841675	3.286165 & 3.500305
Lookback Put	11.248501	11.035587	31.800712	10.92506 & 11.14612
Floating Lookback Call	6.597918	6.361973	44.118722	6.231786 & 6.492160
Floating Lookback Put	6.547841	6.296680	33.406887	6.183395 & 6.409965
American Put	7.032166	6.963289	35.595983	6.84635 & 7.080227

Using Monte Carlo simulation with 10,000 simulations, the option prices obtained were relatively close to the results from the lattice approach. However, number of computations by the lattice approach is only dependent on size of time steps, while number of computations by Monte Carlo simulation is dependent on both number of time steps and number of simulations. This may imply that lattice approach generally requires less computations than Monte Carlo simulation in order to estimate option prices with similar level of accuracy. On the other hand, using Monte Carlo simulation could generate variances and confidence intervals of the sample prices.

The challenges to price these methods are they require good estimation of parameters such as stock volatility and risk-free rate. One of the limitations is that calculation may require high computational power as time step gets very small as well as number of simulations gets very large.

Appendix: Python code

Monte Carlo Simulation

1a) Compute the price of the following options using Monte Carlo Simulation with the sample size 250 or larger.

- Asian Call
- Asian Put
- Lookback Call
- Lookback Put
- Floating Lookback Call
- Floating Lookback Put

In [1]:
```
# known variables
S = 100 # initial price
T = 2/12 # converted to year
K = 105 # strike price
r = 0.02 # risk-free rate
sigma = 0.25 # volatility

n_simulation = 10000 # minimum sample size of 250
n_step = 8 # unit time is per week. So there are 8 weeks per 2 months
```

In [2]:
```
import numpy as np
import pandas as pd
dt = T / n_step
drift = (r - ((sigma**2)/2)) * dt
a = sigma*np.sqrt(dt)

np.random.seed(153) # consistent random seed
z = np.random.normal(0,1,(n_simulation,n_step))

z # x axis is number of steps, y axis is number of simulations
z.shape # 10000 simulations, 8 diff new values
```

Out[2]:
```
(10000, 8)
```

Creating stochastic stock price

In [3]:
```
simul_stock = np.zeros((n_simulation,n_step+1))    # create empty arrays
simul_stock[:,0] = S # current price in the beginning of every simulation

for i in range(1,n_step+1):
  simul_stock[:,i] = simul_stock[:,i-1] * np.exp(drift + a * z[:,i-1])
```

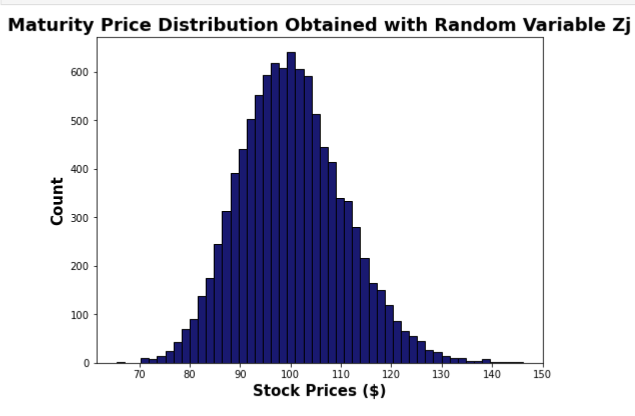
In [4]:
```
print(simul_stock.shape)
simul_stock[0] # printing example of 1 path
```

```
(10000, 9)
```

Out[4]:
```
array([100.        , 100.32791743,  96.14529095,  93.77521395,
        93.72023543,  98.53252511,  98.44954554,  98.34183755,
        93.05219797])
```

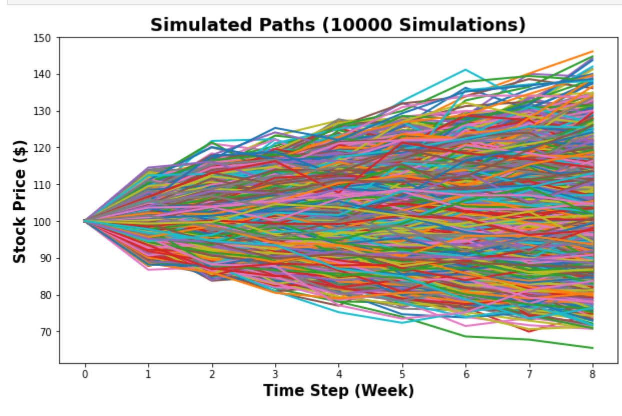
Histogram for Normal Distribution

In [5]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(8, 6))
plt.title('Maturity Price Distribution Obtained with Random Variable Zj', fontsize = 1
plt.ylabel('Count',fontsize = 15, fontweight = 'bold')
plt.xlabel('Stock Prices ($)',fontsize = 15, fontweight = 'bold')
plt.hist(simul_stock[:,8],bins=50, color = 'midnightblue', edgecolor = 'black')
plt.show()
```



Line Chart of Simulated Stock Prices

In [6]:
```
# Line plot for each scenario paths
plt.figure(figsize=(10, 6))
plt.title('Simulated Paths ('+ str(n_simulation) +' Simulations)', fontsize = 18, font
plt.ylabel('Stock Price ($)',fontsize = 15, fontweight = 'bold')
plt.xlabel('Time Step (Week)',fontsize = 15, fontweight = 'bold')
[plt.plot(simul_stock[i,:], linewidth=2) for i in range(n_simulation)]
plt.show()
```



Payoff Calculation

In [7]:
```
# calculate option payoffs
# taking the max of payoff. If it's negative, it won't be exercised (value becomes 0)
payoff_asian_call = np.maximum(simul_stock.mean(axis=1) - K, 0) # (mean of stock - K)
payoff_asian_put = np.maximum(K - simul_stock.mean(axis=1), 0) # (K - mean of stock)

payoff_lookback_call = np.maximum(simul_stock.max(axis=1) - K, 0) # (max of stock - K
payoff_lookback_put = np.maximum(K - simul_stock.min(axis=1), 0) # (K - min of stock)

payoff_fl_lookback_call =  np.maximum(simul_stock[:,-1] - simul_stock.min(axis=1), 0)
payoff_fl_lookback_put = np.maximum(simul_stock.max(axis=1) - simul_stock[:,-1], 0) #
```

Discount payoffs to present value

In [8]:
```
PV_payoff_asian_call = payoff_asian_call * np.exp(-r*T)
PV_payoff_asian_put = payoff_asian_put * np.exp(-r*T)

PV_payoff_lookback_call = payoff_lookback_call * np.exp(-r*T)
PV_payoff_lookback_put = payoff_lookback_put * np.exp(-r*T)

PV_payoff_fl_lookback_call = payoff_fl_lookback_call * np.exp(-r*T)
PV_payoff_fl_lookback_put = payoff_fl_lookback_put * np.exp(-r*T)
```

Sample Mean

In [9]:
```
samplemean_payoff_asian_call = np.mean(PV_payoff_asian_call)
samplemean_payoff_asian_put = np.mean(PV_payoff_asian_put)

samplemean_payoff_lookback_call = np.mean(PV_payoff_lookback_call)
samplemean_payoff_lookback_put = np.mean(PV_payoff_lookback_put)

samplemean_payoff_fl_lookback_call = np.mean(PV_payoff_fl_lookback_call)
samplemean_payoff_fl_lookback_put = np.mean(PV_payoff_fl_lookback_put)
```

Sample Variance

In [10]:
```
samplevar_payoff_asian_call = np.sum(np.square(PV_payoff_asian_call-np.mean(PV_payoff_
samplevar_payoff_asian_put = np.sum(np.square(PV_payoff_asian_put-np.mean(PV_payoff_a

samplevar_payoff_lookback_call = np.sum(np.square(PV_payoff_lookback_call-np.mean(PV_p
samplevar_payoff_lookback_put = np.sum(np.square(PV_payoff_lookback_put-np.mean(PV_pay

samplevar_payoff_fl_lookback_call = np.sum(np.square(PV_payoff_fl_lookback_call-np.mea
samplevar_payoff_fl_lookback_put = np.sum(np.square(PV_payoff_fl_lookback_put-np.mean
```

Result

In [11]:
```
col_1 = ["Asian Call", "Asian Put", "Lookback Call", "Lookback Put", "Floating Lookbac

monte_1 = {'Options': col_1, 'price (sample mean)': [samplemean_payoff_asian_call, sar
                                                     samplemean_payoff_fl_lookback_call, samplemear
           'sample variance': [samplevar_payoff_asian_call, samplevar_payoff_asian_put
                               samplevar_payoff_fl_lookback_call, samplevar_payoff_fl_lookbac
table_1_a = pd.DataFrame(data = monte_1)
table_1_a['95% confidence interval lower bound'] = table_1_a['price (sample mean)'] -
table_1_a['95% confidence interval upper bound'] = table_1_a['price (sample mean)'] +
table_1_a
```

Out[11]:

	Options	price (sample mean)	sample variance	95% confidence interval lower bound	95% confidence interval upper bound
0	Asian Call	0.712330	3.876655	0.673739	0.750921
1	Asian Put	5.538709	21.638668	5.447535	5.629883
2	Lookback Call	3.393235	29.841675	3.286165	3.500305
3	Lookback Put	11.035587	31.800712	10.925058	11.146115
4	Floating Lookback Call	6.361973	44.118722	6.231786	6.492160
5	Floating Lookback Put	6.296680	33.406887	6.183395	6.409965

1 b) Price American Put with Monte Carlo

In [12]:
```
#following the binomial tree model (Page11 of American option PDF)
from scipy.stats import norm
tao = np.zeros(n_simulation) #tao be the optimal exercise time step (or stopping time
for i in range(0,n_simulation):
  for j in range(1,n_step+1):

    d1 = (np.log(simul_stock[i,j]/K)+(r + 0.5 * sigma**2)*(T-j*dt))/(sigma*np.sqrt(T-j
    d2 = d1 - sigma*np.sqrt(T-j*dt)
    E_ST_St = K*np.exp(-r*(T-j*dt))*norm.cdf(-d2) - simul_stock[i,j]*norm.cdf(-d1)
    excercise_payoff = np.maximum(K - simul_stock[i,j], 0)
    if (excercise_payoff >= E_ST_St) and (tao[i]==0):
      tao[i] = j

#calculate present value of payoffs from exercise at time tao
PV_payoff_america_put = np.zeros(n_simulation)
for i in range(0,n_simulation):
  PV_payoff_america_put[i] = np.maximum(K - simul_stock[i,tao[i].astype(int)], 0)*np.e

#average optimal stopping time
average_stopping_time_american_put = np.mean(tao*dt)
#variance
variance_american_put = np.sum((tao*dt-average_stopping_time_american_put)**2)/(n_simu
#sample mean
sample_mean_american_put = np.mean(PV_payoff_america_put)
#sample variance
sample_variance_american_put = np.sum((PV_payoff_america_put-sample_mean_american_put
#95% confidence interval
lower_bound_american_put = sample_mean_american_put - 1.96*np.sqrt(sample_variance_ame
upper_bound_american_put = sample_mean_american_put + 1.96*np.sqrt(sample_variance_ame
```

```
<ipython-input-12-1edec7e5d6e3>:7: RuntimeWarning: divide by zero encountered in doubl
e_scalars
  d1 = (np.log(simul_stock[i,j]/K)+(r + 0.5 * sigma**2)*(T-j*dt))/(sigma*np.sqrt(T-j*d
t))
```

In [13]:
```
table_1_b = pd.DataFrame({'Option': 'American Put', 'Price Sample Mean)': [sample_mear
          'Sample Variance': [sample_variance_american_put], '95% Confidence Interval
          '95% Confidence Interval Upper Bound':[upper_bound_american_put]})
table_1_b
```

Out[13]:

	Option	Price Sample Mean)	Sample Variance	95% Confidence Interval Lower Bound	95% Confidence Interval Upper Bound
0	American Put	6.963289	35.595983	6.84635	7.080227

Lattice Approach

2a) Compute the price of the following options using Monte Carlo Simulation with the sample size 250 or larger.

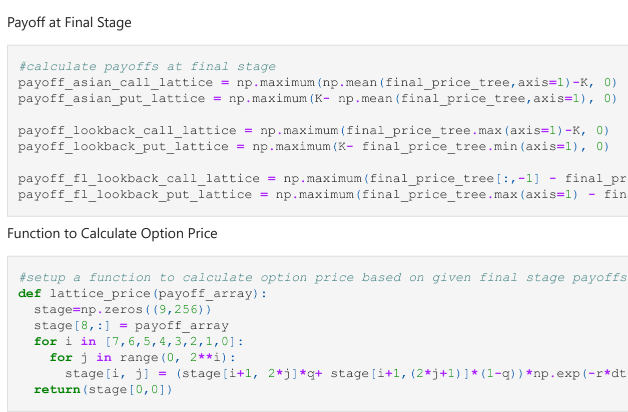
- Asian Call
- Asian Put
- Lookback Call
- Lookback Put
- Floating Lookback Call
- Floating Lookback Put

Creating Binomial Tree

In [14]:
```
from itertools import product
# please check the dt
u = np.exp(sigma*np.sqrt(dt)) # up multiplier
d = 1/u # down multiplier
# q = ((1+r*2/12/8-d)) / (u-d) # risk neutral probability (discrete)
q = (np.exp(r*dt)-d) / (u-d) # risk neutral probability (continuous)

# binomial tree
price_tree = np.zeros([n_step+1, n_step+1])
for i in range(n_step+1):
  for j in range(i+1):
    price_tree[j, i] = S * (d**j)*(u**(i-j))

price_tree.shape

# expanding binomial tree into every single path
n = 8
a = np.array(list(product((0, 1), repeat=n)))
a.shape
a = np.c_[[0] * 2 ** n, a]
a = a.cumsum(axis=1)
a.shape
final_price_tree = np.choose(a, price_tree)
final_price_tree
```

Out[14]:
```
array([[100.        , 103.67433355, 107.48367436, ..., 124.17309715,
        128.73563091, 133.46580739],
       [100.        , 103.67433355, 107.48367436, ..., 124.17309715,
        128.73563091, 124.17309715],
       [100.        , 103.67433355, 107.48367436, ..., 124.17309715,
        119.77226465, 124.17309715],
       ...,
       [100.        ,  96.45588892,  93.03738507, ...,  80.53274203,
         83.49178359,  80.53274203],
       [100.        ,  96.45588892,  93.03738507, ...,  80.53274203,
         77.6785722 ,  80.53274203],
       [100.        ,  96.45588892,  93.03738507, ...,  80.53274203,
         77.6785722 ,  74.92555731]])
```

In [15]:
```
# Line plot for each scenario paths
plt.figure(figsize=(10, 6))
plt.title('Lattice Stock Price ('+ str(2**n_step) +' Paths)', fontsize = 18, fontweigh
plt.ylabel('Stock Price ($)',fontsize = 15, fontweight = 'bold')
plt.xlabel('Time Step (Week)',fontsize = 15, fontweight = 'bold')
[plt.plot(final_price_tree[i,:], linewidth=2) for i in range(2**n_step)]
plt.show()
```



Payoff at Final Stage

In [16]:
```
#calculate payoffs at final stage
payoff_asian_call_lattice = np.maximum(np.mean(final_price_tree,axis=1)-K, 0)
payoff_asian_put_lattice = np.maximum(K- np.mean(final_price_tree,axis=1), 0)

payoff_lookback_call_lattice = np.maximum(final_price_tree.max(axis=1)-K, 0)
payoff_lookback_put_lattice = np.maximum(K- final_price_tree.min(axis=1), 0)

payoff_fl_lookback_call_lattice = np.maximum(final_price_tree[:,-1] - final_price_tree
payoff_fl_lookback_put_lattice = np.maximum(final_price_tree.max(axis=1) - final_price
```

Function to Calculate Option Price

In [17]:
```
#setup a function to calculate option price based on given final stage payoffs
def lattice_price(payoff_array):
  stage=np.zeros((9,256))
  stage[8,:] = payoff_array
  for i in [7,6,5,4,3,2,1,0]:
    for j in range(0, 2**i):
      stage[i, j] = (stage[i+1, 2*j]*q+ stage[i+1,(2*j+1)]*(1-q))*np.exp(-r*dt) # / (
  return(stage[0,0])
```

Result

In [18]:
```
lattice_1 = {'Options': col_1, 'Price ($)': [lattice_price(payoff_asian_call_lattice),
                                             lattice_price(payoff_lookback_p
table_2_a = pd.DataFrame(data = lattice_1)
table_2_a
```

Out[18]:

	Options	Price ($)
0	Asian Call	0.692490
1	Asian Put	5.509542
2	Lookback Call	3.574964
3	Lookback Put	11.248501
4	Floating Lookback Call	6.597918
5	Floating Lookback Put	6.547841

2b) Lattice Approach on American Put Option

In [19]:
```
decrementing_number = []
for i in range(-1, -1*price_tree.shape[0], -1):
  decrementing_number.append(i)
decrementing_number
```

Out[19]:
```
[-1, -2, -3, -4, -5, -6, -7, -8]
```

Last Tree

In [20]:
```
price_tree
```

Out[20]:
```
array([[100.        , 103.67433355, 107.48367436, 111.43298307,
        115.52740254, 119.77226465, 124.17309715, 128.73563091,
        133.46580739],
       [  0.        ,  96.45588892, 100.        , 103.67433355,
        107.48367436, 111.43298307, 115.52740254, 119.77226465,
        124.17309715],
       [  0.        ,   0.        ,  93.03738507,  96.45588892,
        100.        , 103.67433355, 107.48367436, 111.43298307,
        115.52740254],
       [  0.        ,   0.        ,   0.        ,  89.74003679,
         93.03738507,  96.45588892, 100.        , 103.67433355,
        107.48367436],
       [  0.        ,   0.        ,   0.        ,   0.        ,
         86.5595502 ,  89.74003679,  93.03738507,  96.45588892,
        100.        ],
       [  0.        ,   0.        ,   0.        ,   0.        ,
          0.        ,  83.49178359,  86.5595502 ,  89.74003679,
         93.03738507],
       [  0.        ,   0.        ,   0.        ,   0.        ,
          0.        ,   0.        ,  80.53274203,  83.49178359,
         86.5595502 ],
       [  0.        ,   0.        ,   0.        ,   0.        ,
          0.        ,   0.        ,   0.        ,  77.6785722 ,
         80.53274203],
       [  0.        ,   0.        ,   0.        ,   0.        ,
          0.        ,   0.        ,   0.        ,   0.        ,
         74.92555731]])
```

In [21]:
```
# setup empty array
option_tree = price_tree.copy()
option_tree = option_tree * 0

# setup last tree
for j in range(price_tree.shape[0]):
  option_tree[j][-1] = np.maximum(K - price_tree[:,[-1]][j], 0)
# print(option_tree)


for i in decrementing_number:
  # print(i)
  for k in range(price_tree.shape[0]):
    if k+1 < price_tree.shape[0] and i-1 > (n_step+1)*-1:
      option_tree[k][i-1] = (option_tree[k][i] * q + option_tree[k+1][i] * (1-q)) * np
      if option_tree[k][i-1] >= 0 and (K > price_tree[k][i-1]) and price_tree[k][i-1]
        # print("condition 1:", option_tree[k][i-1])
        option_tree[k][i-1] = np.maximum( option_tree[k][i-1], (K - price_tree[k][i-1]
        # print("condition 1 after calc:", option_tree[k][i-1])
      elif option_tree[k][i-1] >= 0 and (K < price_tree[k][i-1]) and price_tree[k][i-
        # print("condition 2:", option_tree[k][i-1])
        option_tree[k][i-1] = option_tree[k][i-1]*1
      else:
        option_tree[k][i-1] = 0
  # print(option_tree)


option_tree[0][0] = (option_tree[0][1] * q + option_tree[1][1] * (1-q)) * np.exp(-r*dt

if K != S:
  if option_tree[0][0] >= 0 and (K > price_tree[0][0]) and price_tree[0][0] != 0:
    option_tree[0][0] = np.maximum( option_tree[0][0], (K - price_tree[0][0]) ) # str
  elif option_tree[0][0] >= 0 and (K > price_tree[0][0]) and price_tree[0][0] != 0:
    option_tree[0][0] = option_tree[0][0]*1
  else:
    option_tree[0][0] = 0


# print("---------------------")
print("Payoff Tree")
print(option_tree)
print("American Put Option Price(Lattice):", option_tree[0][0]) # <<< american put pr
```

```
Payoff Tree
[[ 7.03216596  4.63139447  2.64202316  1.18300115  0.32016202  0.
   0.          0.          0.        ]
 [ 0.          9.40778412  6.59893256  4.08440751  2.03568685  0.63645816
   0.          0.          0.        ]
 [ 0.          0.         12.18818427  9.08647776  6.11007618  3.41854727
   1.26523124  0.          0.        ]
 [ 0.          0.          0.         15.25996321 12.03199996  8.77193655
   5.54691032  2.51518512  0.        ]
 [ 0.          0.          0.          0.         18.4404498  15.25996321
  11.96261493  8.54411108  5.        ]
 [ 0.          0.          0.          0.          0.         21.50821641
  18.4404498  15.25996321 11.96261493]
 [ 0.          0.          0.          0.          0.          0.
  24.46725797 21.50821641 18.4404498 ]
 [ 0.          0.          0.          0.          0.          0.
   0.         27.3214278  24.46725797]
 [ 0.          0.          0.          0.          0.          0.
   0.          0.         30.07444269]]
American Put Option Price(Lattice): 7.03216596414477
```

Result

In [22]:
```
table_2_b = pd.DataFrame({'Option': 'American Put', 'Price ($)': [option_tree[0][0]]})
table_2_b
```

Out[22]:

	Option	Price ($)
0	American Put	7.032166