European Option Pricing - Binomial Model Approach

In [83]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Parameters of the model

–

S_0 : initial underlying price

K : Strike price

N : Time to maturity

r : risk-free interest rate

σ : volatility

M : number of time steps

$\Delta_t = \frac{N}{M}$

option type : CALL/ PUT

In [84]:
```
__author__ = 'Faraujo'

## Binomial model function

def binomial_model(S_0,K,N,M,r,sigma,p_hat, option_type):

    q_hat = 1 - p_hat

    delta_t = N / M

    u = np.exp(sigma * np.sqrt(delta_t) + (r-0.5*sigma**2)*delta_t)

    d = np.exp(-1*sigma * np.sqrt(delta_t) + (r-0.5*sigma**2)*delta_t)

    ## Return a matrix of zeros with dimension {M+1} x {M+1}

    matrix_dimension = (M+1,M+1)

    price_matrix = np.zeros(matrix_dimension)

    ## Replace matrix values

    price_matrix[0,0] = S_0

    ## Loop through all nodes in order to compute the underlying asset price in all states of nature

    for index in range(1,M+1):
        price_matrix[0,index] = price_matrix[0,index - 1 ]*u
        for j in range(1,index+1):
            price_matrix[j,index] = price_matrix[j - 1,index-1]*d

    ## If condition based on the option type

    if option_type == 'CALL':

        alpha = 1

    if option_type == 'PUT':

        alpha = -1

    ## Determine the payoff at the maturity date
    value_matrix = np.zeros(matrix_dimension)

    for i in range(M+1):
        value_matrix[i,M] = max(alpha*(price_matrix[i,M] - K),0)

    ## Compute the present value of the derivative in each node
    for column in range(M-1,-1,-1):
        for row in range(0,M):
            value_matrix[row,column] = ((1/(1+r))**delta_t)*((p_hat)*(float(value_matrix[row,column+1])) + (q_hat)*(
value_matrix[row+1,column+1]))

    # return the price of the derivative at time t = 0
    return round(value_matrix[0,0], 7)
```

In a multiperiod binomial model, the no-arbitrage price of the derivative security that pays V_N at time N can be computed recursively as follows:

$$V_n(w_1w_2...w_n) = \frac{1}{(1+r)^{\Delta_t}} \cdot [\tilde{p} \cdot V_n(w_1w_2...w_nH) + \tilde{q} \cdot V_n(w_1w_2...w_nT)]$$

The two tables below show the price under our model for a given number of steps. Note that as M increases, the binomial tree price converges to the Black-Scholes price.

In [85]:
```
call_M10 = binomial_model(S_0 = 9,K = 10,N = 3,M = 10,r = 0.06,sigma = 0.3, p_hat = 0.5,option_type = 'CALL')
call_M50 = binomial_model(S_0 = 9,K = 10,N = 3,M = 50,r = 0.06,sigma = 0.3, p_hat = 0.5,option_type = 'CALL')
call_M100 = binomial_model(S_0 = 9,K = 10,N = 3,M = 100,r = 0.06,sigma = 0.3, p_hat = 0.5,option_type = 'CALL')
call_M200 = binomial_model(S_0 = 9,K = 10,N = 3,M = 200,r = 0.06,sigma = 0.3, p_hat = 0.5,option_type = 'CALL')
call_M400 = binomial_model(S_0 = 9,K = 10,N = 3,M = 400,r = 0.06,sigma = 0.3, p_hat = 0.5,option_type = 'CALL')
```

The call option is shown in the first table.

In [86]:
```
binomial_call = pd.DataFrame({'Binomial Tree': [call_M10, call_M50, call_M100,call_M200,call_M400],
                   'B&S': [ 2.120093831410867,  2.120093831410867,  2.120093831410867, 2.120093831410867, 2.12009383
1410867]},
                  index=['M=10', 'M=50', 'M=100','M=200','M=400'])

binomial_call['Error (Absolute Value)'] = abs(binomial_call['Binomial Tree'] - binomial_call['B&S'])

binomial_call
```

Out[86]:

	Binomial Tree	B&S	Error (Absolute Value)
M=10	2.132339	2.120094	0.012245
M=50	2.137571	2.120094	0.017478
M=100	2.134364	2.120094	0.014270
M=200	2.131069	2.120094	0.010975
M=400	2.131013	2.120094	0.010919

In [87]:
```
put_M10 = binomial_model(S_0 = 9,K = 10,N = 3,M = 10,r = 0.06,sigma = 0.3, p_hat = 0.5,option_type = 'PUT')
put_M50 = binomial_model(S_0 = 9,K = 10,N = 3,M = 50,r = 0.06,sigma = 0.3, p_hat = 0.5,option_type = 'PUT')
put_M100 = binomial_model(S_0 = 9,K = 10,N = 3,M = 100,r = 0.06,sigma = 0.3, p_hat = 0.5,option_type = 'PUT')
put_M200 = binomial_model(S_0 = 9,K = 10,N = 3,M = 200,r = 0.06,sigma = 0.3, p_hat = 0.5,option_type = 'PUT')
put_M400 = binomial_model(S_0 = 9,K = 10,N = 3,M = 400,r = 0.06,sigma = 0.3, p_hat = 0.5,option_type = 'PUT')
```

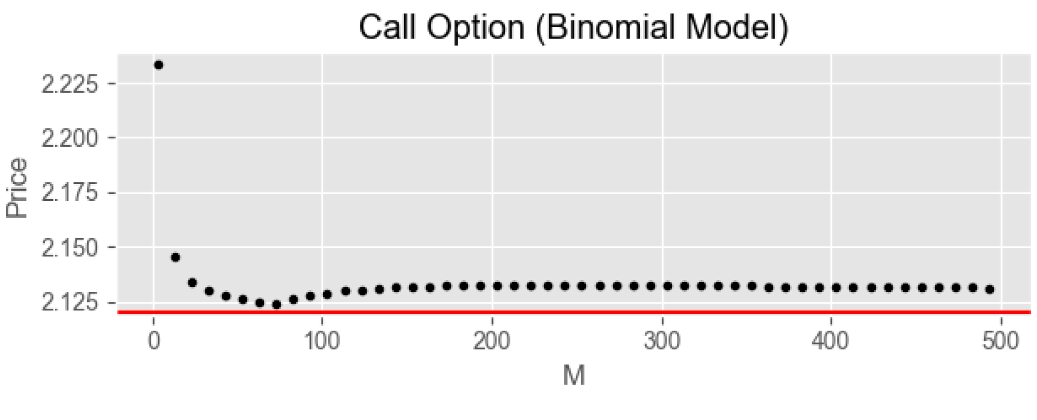
The put option is shown in the second table.

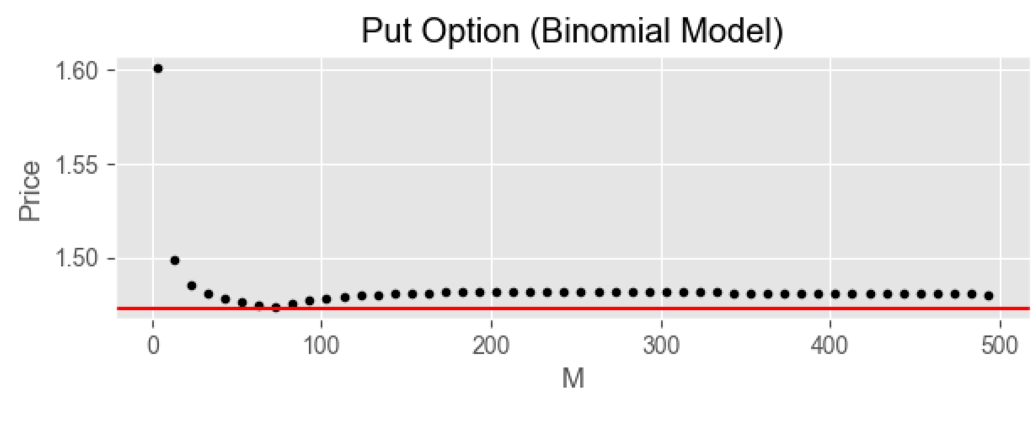
In [88]:
```
binomial_put = pd.DataFrame({'Binomial Tree': [put_M10, put_M50, put_M100,put_M200,put_M400],
                   'B&S': [  1.472795945523587,   1.472795945523587,   1.472795945523587,  1.472795945523587,  1.472
795945523587]},
                  index=['M=10', 'M=50', 'M=100','M=200','M=400'])

binomial_put['Error (Absolute Value)'] = abs(binomial_put['Binomial Tree'] - binomial_put['B&S'])
binomial_put
```

Out[88]:

	Binomial Tree	B&S	Error (Absolute Value)
M=10	1.487126	1.472796	0.014330
M=50	1.488001	1.472796	0.015205
M=100	1.484245	1.472796	0.011449
M=200	1.480675	1.472796	0.007879
M=400	1.480482	1.472796	0.007687

In [89]:
```
# Define the x-axis
Mx = np.arange(3,503,10)

# Set the binomial price array
call_array = []
put_array = []

for x in Mx:
    price_call = binomial_model(S_0 = 9,K = 10,N = 3,M = x,r = 0.06,sigma = 0.3, p_hat = 0.5,option_type = 'CALL')
    price_put = binomial_model(S_0 = 9,K = 10,N = 3,M = x,r = 0.06,sigma = 0.3, p_hat = 0.5,option_type = 'PUT')
    call_array.append(price_call)
    put_array.append(price_put)


#plot the function
plt.subplot(3, 1, 1)
plt.plot(Mx,call_array,'.',color='black')
plt.title('Call Option (Binomial Model)')
plt.xlabel('M')
plt.ylabel('Price')
plt.axhline(y=2.120093831410867, color='red', linestyle='-')

plt.subplot(3, 1, 3)
plt.plot(Mx,put_array,'.',color='black')
plt.title('Put Option (Binomial Model)')
plt.xlabel('M')
plt.ylabel('Price')
plt.axhline(y=1.472795945523587, color='red', linestyle='-')

#show the plot
plt.rcParams['figure.figsize'] = [7, 7]
plt.style.use('seaborn') #seaborn style
plt.show()
```





The red horizontal line on both graphs above represents the price under the Black-Scholes assumption. Analyzing the performance, our model converges quickly - it only takes around 80 steps to reach a stable price level.