IE 420

Spring 2018

Final Project Report

Pricing American and European Options by Binomial Model

Feiyang Gu(fgu6)/Jinhao Hu(jhu41)

Liming Feng

Introduction:

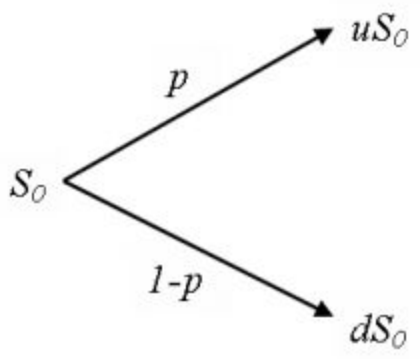
Option is a financial term relating to derivative contract. It gives a investor or holder the right, but not the obligation, to buy or sell the asset at certain time with agreed-upon price. There are two kinds of option. One is call option which gives a buyer the right but not obligation to buy an asset at future time with certain price. Another one is put price which gives the seller the right but obligation to sell an asset at future time with certain price. Usually, buyers who buy call option expect the strike price will be higher than the current asset price, and then they will gain profit from the price difference. Sellers who sell put option expect the strike price will be lower than the current asset price to gain the profit. Nowadays, with the drastically promoting finance, option becomes a vital part in our live. Moreover, it also becomes a way that people make a living if they are far-sighted. Thus, how to compute the option price naturally is being significant. There are two methods we can compute option price. They are binomial model method and black scholes model method. In the following report, we will mainly focus on binomial model method and mainly explore American option price.

Conception:

Binomial Model:

For binomial model, there are two possible outcomes after each iteration with same probability. And also, the sum of their probability is 1. We firstly consider one -step binomial and then evolve to multi-step binomial model.

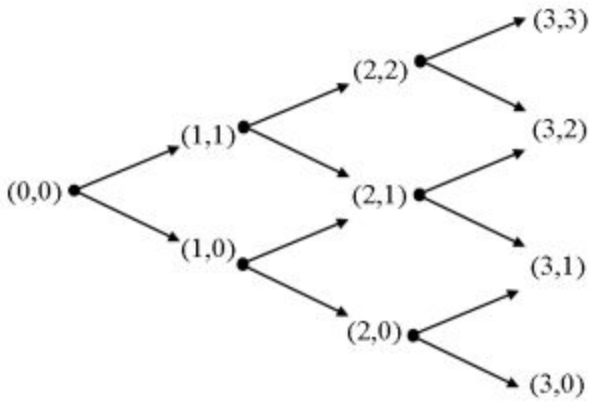
One-step binomial model: At time 0, the asset price is S0, and at time 1, the asset will have p



possibility to be uS0 and have (1-p) possibility to be dS0.

We define a node(n,j). N is time interval in a certain time. J is the number of times that S0 takes p possibility option. In one-step binomial model, we say uS0 is (1,1) and dS0 is (1,0).

Multi-Step binomial model: At time N, there are N+1 possible results. We can calculate any stock price at node(n,j) by expression S(n,j) = u^j*d(n-j)*S0



In order to calculate option price, we usually calculate from last step to get the former step till the first one. This is called backward induction.

Basic variable K, T, S0, σ, r, q, N are needed

K: Strike Price

T: Option Maturity

S0: Initial Stock Price

σ: Volatility (for Black-Scholes-Merton model)

r: Continuous Compounding Risk Free Interest Rate

q: Continuous Dividend Yield

N: Number of Time Steps

u, d: up and down value rate (for Binomial model) $u = e^{\sigma\sqrt{\delta}}, \quad d = e^{-\sigma\sqrt{\delta}}$

P* :Risk Neutral Probability can be expressed as $p^* = \frac{e^{(r-q)\delta} - d}{u - d}$

δ: time period δ=T/N

European Options: European options can be exercised only at maturity.

The intrinsic value of European options:

$$\text{call: } (S-K)^+, \quad \text{put: } (K-S)^+$$

American Options: American options can be exercised any time before or at maturity.

Payoff of American when exercised at nδ is

$$\text{put: } (K - S_{n,j})^+, \quad \text{call: } (S_{n,j} - K)^+$$

We start with

$$f_{N,j} = (K - S_{N,j})^+, \quad j = 0, 1, \cdots, N$$

to do backward induction for American options. For n = N-1, N-2,…,1

$$f_{n,j} = \max\left((K - S_{n,j})^+, e^{-r\delta}(p^* f_{n+1,j+1} + (1 - p^*) f_{n+1,j})\right)$$

Finally we can compute f0-option price by using this formula. For european option, we do not need to compare the second equation with first one because we do not do the early exercise. There is another formula to compute European options. It is called Black-Scholes .

Black-Scholes Model:

European call price:

$$c = S_0 e^{-qT} N(d_1) - K e^{-rT} N(d_2)$$

European put price: $$p = -S_0 e^{-qT} N(-d_1) + K e^{-rT} N(-d_2)$$

N(x) is the cdf of N(0,1)

$$d_1 = \frac{\ln(S_0/K) + (r - q + \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}}$$

$$d_2 = d_1 - \sigma\sqrt{T}$$

Answers for the questions:

1. Implement the CRR binomial model to price European and American puts and calls on a stock paying continuous dividend yield: Binomial(Option, K, T, S0, σ, r, q, N, Exercise) where Option = C for calls and P for puts, K is the strike, T is the time to maturity, S0 is the initial stock price, σ is the volatility, r is the continuous compounding risk free interest rate, q is the continuous dividend yield, N is the number of time steps, and Exercise = A for American options and E for European options. Your program should output both option price and computational time.

Code:

```
def Binomial(Option, K, T, S0, sigma, r, q, N, Exercise):
    delta = T/N
    u = np.e **(sigma *np.sqrt(delta))
    d = np.e **(-sigma*np.sqrt(delta))
    p =(np.e **((r-q)*delta)-d)/(u-d)
    if (Exercise=="E"):
        model = []
        if(Option=="C"):
            for i in range(N+1):
                model.append(max(S0 * (u**(N-i)) * (d**i)-K,0))
        else: # European put
            for i in range(N+1):
                model.append(max(-S0 * (u**(N-i)) * (d**i)+K,0))
        for i in range(N):
            temp =[]
            for x in range(N - i):
                temp.append(np.e**((-r) * delta) * (p * model[x] + (1-p) * model[x+1]))
            model = temp
    else:    # American
        node = []
        for i in range(N + 1):
            temp = []
            for j in range(i + 1):
                temp.append(S0 * (u ** (i - j)) * (d ** (j)))
            node.append(temp)
        if(Option=="P"):
            for i in range(len(node[-1])):
                model.append( K + (node[-1][i]))#np.asarray(node[-1])
                model[i]=max(model[i],0)
            for i in range(N):
                for j in range(N - i):
                    model[j] = max(K-node[N-i-1][j], np.e**((-r)*delta) * ((p) * model[j] + (1-p) * model[j+1]))
        else:   #Option=="C"
            model=[]
            for i in range(len(node[-1])):
                model.append( -K + (node[-1][i]))#np.asarray(node[-1])
                model[i]=max(model[i],0)
            for i in range(N):
                for j in range(N - i):
                    model[j] = max(-K+node[N-i-1][j], np.e**((-r)*delta) * ((p) * model[j] + (1-p) * model[j+1]))

    return model[0]
```

Explanation:

For European:

For European call: To begin with, we need to find the current stock prices at the last step given u and d. Meanwhile, we have to subtract K, and only keep the positive values, substituting the negative ones with 0. Then, starting from the list we have now, which is the last step, we have to move backward to find the initial option price. We put values in the formula $e^{-r\delta}(p * f_{n+1,j+1} + (1 - p*)f_{n+1}$. And we update the model in each iteration. And finally, there is actually one item in list, and this is the option price we need.

For European put: It is almost the same with european call, the only change here is we use K to subtract the item in the binomial model.

For American:

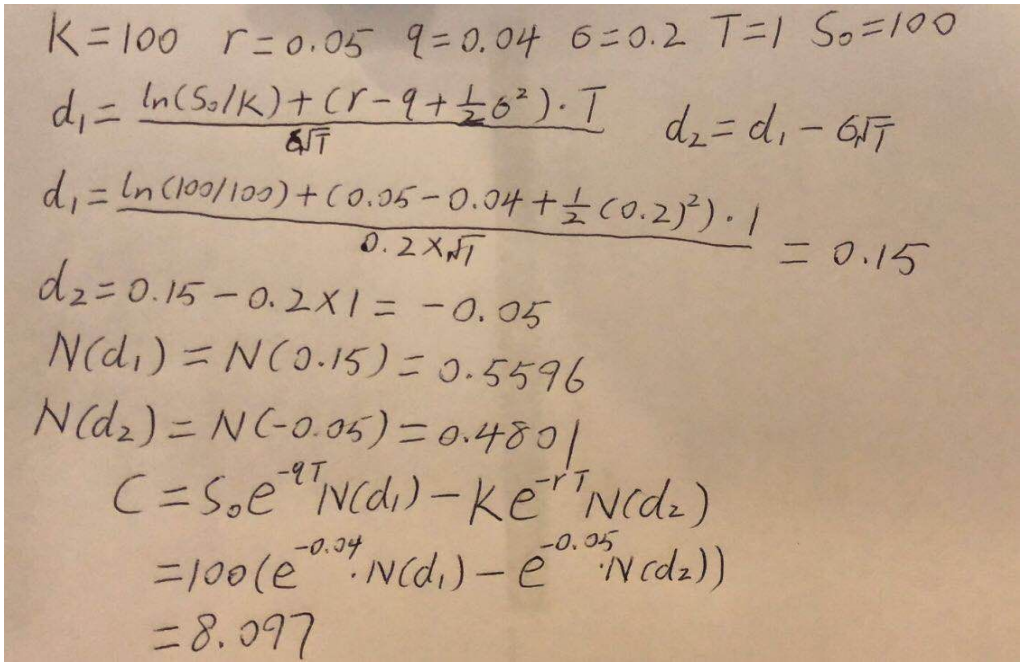
First we need to get the the multi-step binomial model over N steps. We build lists inside a bigger list. The first inner list has node for the first steps, the second inner list has nodes for the second steps, and so on. Hence, each step should have a corresponding inner list which contains all the nodes in that step. Then we have different cases for put and call.

For American put: payoff when exercised at $n\delta$ is $(K - S_{n,j})+$. We have to make sure all the items in the last inner list goes through this formula, since we need to start the computation from the last step, and we have to make sure it is greater than 0, or we have to choose 0 to substitube for that item instead. Then we have to go those the formula $f_{n,j} = \max(K - S_{n,j}) +, e^{-r\delta}(p * f_{n+1,j+1} + (1 - p*)f_{n+1,j})$. Then we update our model one step by one step forward. Each step we are going to have one item less, and in the final round, which is actually the first step we need to find, the first item in the list is going to be the option price we need. Instead of updating the

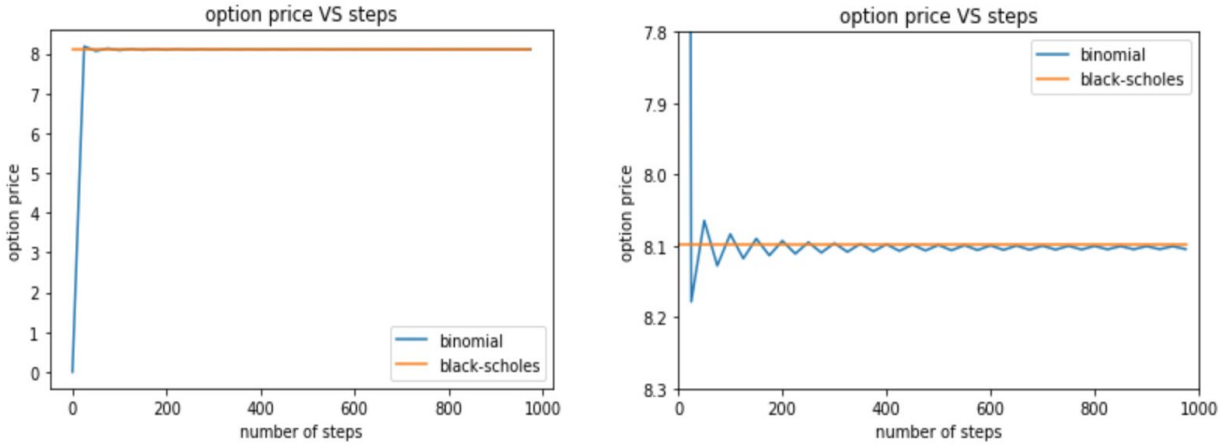
whole list of model like we did in European Exercise, one of us decide to only update the item we need in the list here.

For American call: It is pretty similar to American put. We also have to find out multi-step binomial model first. Then we have to build the same list with the last step in the binomial model and move backward to the first state. However, the differences between American call and American put are the formulas we used. We need to use (Sn, j − K) + in both formulas.

2. Consider a 1-year European call option with strike K = 100. The current stock price is 100. Other parameters are r = 0.05, q = 0.04, σ = 0.2. Use the Black-Scholes formula to compute the price of the call option. In the binomial model, take a sequence of increasing numbers of time steps N. Verify that the binomial option prices converge to the Black-Scholes option price as N increases. Construct a table/plot to visualize the convergence.



Graph for option price VS steps for European Call



Here is what we calculated european call price by using the Black-Scholes formula. we ran the binomial code and plot the data and we found the graph is gradually converging to the result we got from Black-Scholes when N is increasing in sequence.

3. Consider American puts with K = 100, σ = 0.2, r = 0.05, q = 0. For time to maturity varying from 1 month to 12 months, investigate the number of time steps needed to achieve the required 10−3 accuracy. Calculate and plot the price of a 12-month put as a function of S0. For time to maturity i/12, i = 0, 1, · · · , 12, find the critical stock price S ∗ (i) on the early exercise boundary (see notes below about how to determine S ∗ (i)). Report and plot the early exercise boundary {S ∗ (i), 0 ≤ i ≤ 12} as a function of the time to maturity. Repeat the above for q = 0.04. How do put prices and the early exercise boundary change when the

continuous dividend yield goes from 0 to 0.04? What is the intuition behind this dependence on the dividend yield?

Code for investigating the number of time steps for 1 to 12 month:

```
if __name__ == '__main__':

    accuracy = 10 ** -3
    number = np.zeros(12)
    temp=1500
    upper=3000
    lower=0
    for n in range(12):
        step = int(temp)
        temp=temp-lower
        while(True):
            temp=int(temp/2)
            print(step)
            pre=step
            value_now = Binomial("P", 100, (n + 1) / 12, 100, 0.2, 0.05, 0, step, "A")

            value_next = Binomial("P", 100, (n + 1) / 12, 100, 0.2, 0.05, 0, step+1, "A")
            if abs(value_next - value_now) > accuracy:
                step=step+temp
            else:
                step=step-temp

            if(step==pre):
                break

        number[n] = step
        lower=step
        temp=(upper-lower)/2+lower
        #print("month ")
    print(number)
```

Explanation: We first set the accuracy, the upper and lower bound. We actually tried the upper from much smaller number. Since we find there are consecutive numbers in the array equals our larger bound. We decide to increase it and finally, all the items in the array is lower than 3000. In the loop for each month, we need to find the current option price and the next-step option price, and we are doing this by binary search, which is guessing the middle value in the boundaries, and we keep updating the guess with the adding or subtracting half the current guess to the

boundary. Specifically, if next-step option price minus the current option price is greater than positive 0.001, we are going to make add current temp to step, if not, we are going to subtract temp to step. And temp is going to be cut to half in the beginning of each loop. Whenever, the current value equals the next-step value, which means the guess satisfy our condition, we are going to break out this small loop and go on with the next month. And since we know that with the increase of the time, the steps needed to achieve the accuracy should also increase, we can update each lower bound with the step for the current month.

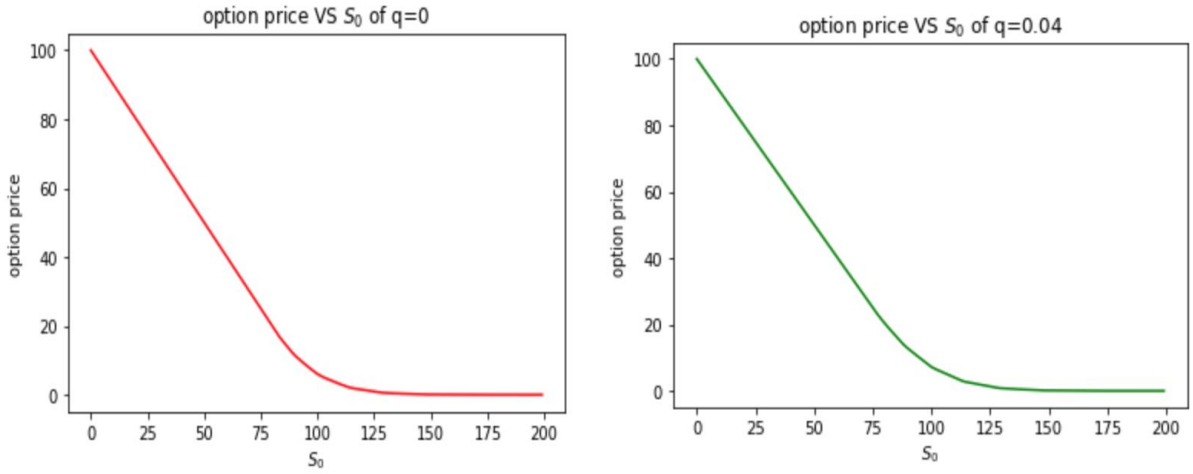
Code for early exercise boundary $\{S * (i), 0 \leq i \leq 12\}$ as a function of the time to maturity:

```
Sstar = np.zeros(12)
upper=100
lower=50
temp=25
for i in range(12):
    if i == 0:
        upper = 100 # Starting from strike price
    else:
        upper = Sstar[n-1]

    temp=(upper-lower)/2
    ss=lower+temp
    while (True):

        temp=temp/2
        pre=ss
        print(ss)
        option_price = Binomial("P", 100, (i+1)/12, ss, 0.2, 0.05, 0.04,int( number[i]), "A")
        if((option_price + ss -100) > 0.005):
            ss =ss-temp
        else:
            ss =ss+temp

        if(int(ss*100)==int(pre*100)):
            break

    Sstar[n]=int(ss*100)/100
    upper=ss
    temp=(upper-lower)/2

    print("month ", n, " :", Sstar[n])
print(Sstar)
```

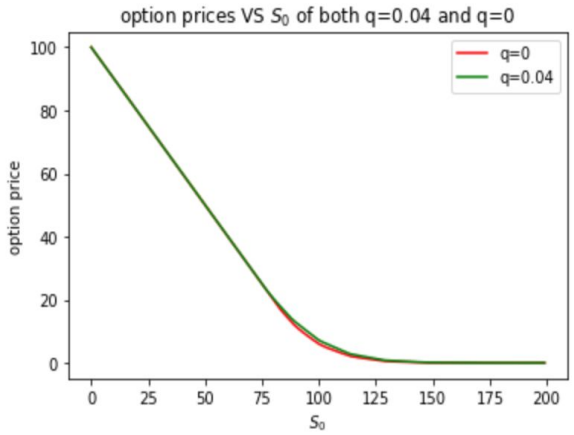
Explanation:

Firstly, we need to step the upper and lower bound, because we are using binary search again here. We have to update the upper boundary each time since we know that as time increase, the current stock price is going to decrease. Then the binary search we did here is pretty similar to the one above. Whenever option price minus (100-our guess) is greater than 0.005, we are going to lower the guess with temp, and raise the guess with temp when it is lower than 0.005. The temp is going to be divided by 2 each time. And finally when the current guess is the same as the one one-step-ahead of it, we are going to break out of the loop and we will have the stock price for the month.

Graph for option price VS S0 at q=0 and q=0.04



The graph above seems alike, we combine them together, and also zoomed in. And we can tell that when we have larger q, option price is going to decrease slowlier with the growth of S0. We can tell if we have large dividend yield in American put option, we expect to have more profit.



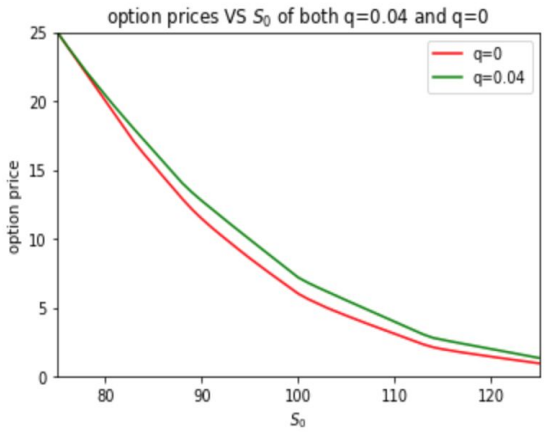


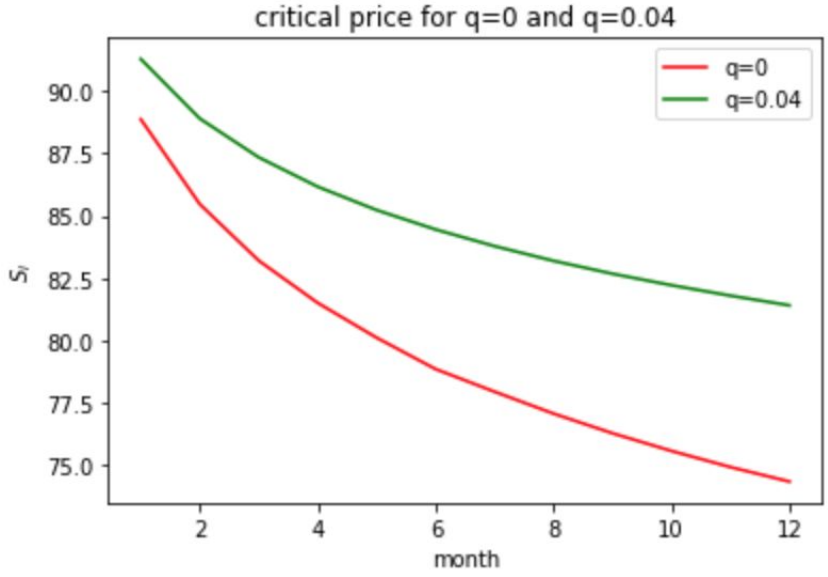
Table for results of number of steps needed to achieve accuracy for q=0 and q=0.04

American Put	N(1)	N(2)	N(3)	N(4)	N(5)	N(6)	N(7)	N(8)	N(9)	N(10)	N(11)	N(12)
q=0	979	1300	1507	1663	1786	1887	1971	2041	2099	2153	2199	2238
q=0.04	1035	1391	1637	1823	1974	2097	2202	2290	2369	2434	2489	2543

Table for results of critical stock price for q=0 and q=0.04

American Put	S*(1)	S*(2)	S*(3)	S*(4)	S*(5)	S*(6)	S*(7)	S*(8)	S*(9)	S*(10)	S*(11)	S*(12)
q=0	**91.3**	**88.91**	**87.35**	**86.17**	**85.23**	**84.45**	**83.78**	**83.19**	**82.67**	**82.21**	**81.79**	**81.4**
q=0.04	**88.88**	**85.47**	**83.2**	**81.49**	**80.11**	**78.94**	**77.93**	**77.04**	**76.25**	**75.54**	**74.9**	**74.32**

Graph for critical stock price for q=0 and q=0.04



4. Consider American calls with K = 100, σ = 0.2, r = 0.05, q = 0.04. Investigate the number of time steps needed to achieve 10−3 accuracy for 1-month to 12-month calls. Calculate and plot the price of a 12-month call as a function of S0. For time to maturity i/12, i = 0, 1, · · · , 12, find the critical stock price S ∗ (i) on the early exercise boundary. Report and plot the early exercise boundary {S ∗ (i), 0 ≤ i ≤ 12} as a function of the time to maturity. Repeat the above for q = 0.08. How do call prices and the early exercise boundary change when the continuous dividend yield goes from 0.04 to 0.08? What is the intuition behind this dependence on the dividend yield?

Code for finding the steps for q=0.08, steps for q=0.04 should be the same

```
if __name__ == '__main__':

    accuracy = 10 ** -3
    x1 = np.arange(12)
    y1 = np.zeros(12)
    temp=1500
    upper=6000
    lower=0
    for n in x1:
        step = int(temp)
        temp=temp-lower
        while(True):
            temp=int(temp/2)
            print(step)
            pre=step
            value_now = Binomial("C", 100, (n + 1) / 12, 100, 0.2, 0.05, 0.08, step, "A")
            value_next = Binomial("C", 100, (n + 1) / 12, 100, 0.2, 0.05, 0.08, step+1, "A")
            if abs(value_next - value_now) > accuracy:
                step=step+temp
            else:
                step=step-temp

            if(step==pre):
                break

        y1[n] = step
        lower=step
        temp=(upper-lower)/2+lower
```

Code for early exercise boundary $\{S * (i), 0 \leq i \leq 12\}$ as a function of the time to maturity:

```
guesstar = np.zeros(12)
upper=155
lower=0

for i in range(12):

    if(i!=0):
        lower = guesstar[i-1]

    temp=(upper-lower)/2
    guess=lower+temp
    while (True):

        temp=temp/2
        pre=guess
        print(guess)
        option_price = Binomial("C", 100, (i+1)/12, guess, 0.2, 0.05, 0.08,int( number[i]), "A")
        if((option_price - guess +100) < 0.005):
            guess =guess-temp
        else:
            guess =guess+temp

        if(int(guess*100)==int(pre*100)):
            break

    guesstar[i]=int(guess*100)/100
    lower=guess
    temp=(upper-lower)/2

    print("month ", i, " :", guesstar[i])
print("0.08call")
print(guesstar)
```

Explanation: Almost same with the third question, we calculated steps by setting the accuracy to be 0.001 and then used the result to compute option price and plot the relationship between the option price with the current stock price. At last, we derived the critical stock price by making the difference between the option price and the intrinsic value less than 0.005. The only different computing procedure is that we switch the position of Stock price at n,j and strike price rather than (K-Sn,j)+ when calculating payoff. We firstly ran q = 0.04 and then changed q to 0.08 and did the same thing with the above. There is also a tiny difference when we do the binary search, which is we have to update the lower bound instead of the upper bound when we try to find the critical strike price.

Graph for option prices VS S0 for both q=0.04 an q=0.08, call option

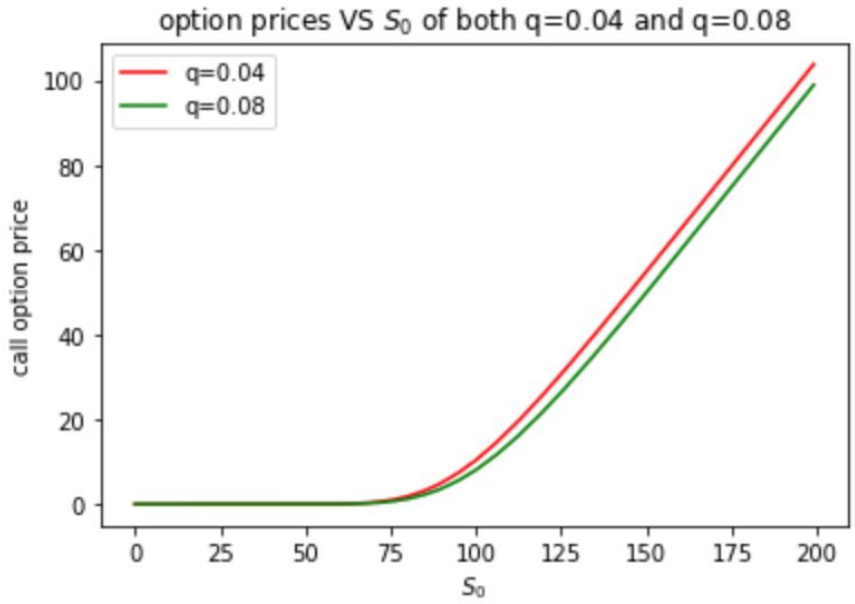


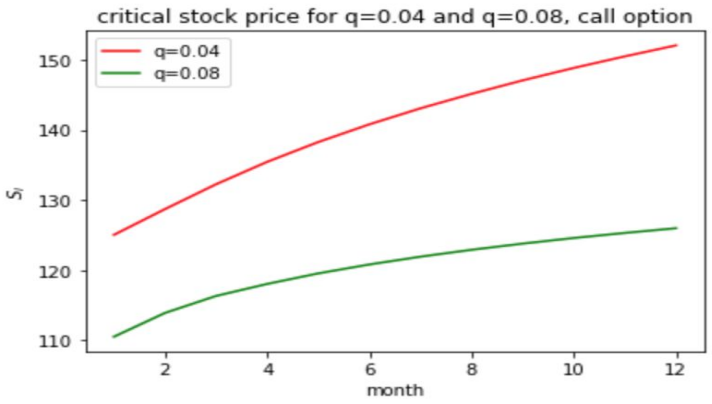
Table for results of number of steps needed to achieve accuracy for q=0.04 and q=0.08

American Call	N(1)	N(2)	N(3)	N(4)	N(5)	N(6)	N(7)	N(8)	N(9)	N(10)	N(11)	N(12)
q=0.04	1147	1614	1968	2264	2522	2747	2955	3140	3324	3484	3636	3778
q=0.08	1035	1391	1637	1823	1974	2097	2202	2290	2369	2434	2489	2543

Table for results of critical stock price for q=0.04 and q=0.08

American Call	S*(1)	S*(2)	S*(3)	S*(4)	S*(5)	S*(6)	S*(7)	S*(8)	S*(9)	S*(10)	S*(11)	S*(12)
q=0.04	125.05	128.72	132.26	135.45	138.27	140.79	143.07	145.13	147.05	148.82	150.47	152.03
q=0.08	110.54	113.93	116.25	118.06	119.56	120.83	121.94	122.92	123.8	124.6	125.32	125.99

Graph for critical stock price for q=0.04 and q=0.08. When q is greater, the critical strike price is going to be smaller as the time increase. We can tell if the dividend yield increase, we can get more profit, because dividend yield is how much we can gain back.



Efforts on reducing computational time

1) We originally use the solver function in the python library. This is also the first reason we choose python instead of C++, since python has lots of useful and convenient libraries. However, we found it too slow. Then we decide to find the value manually by doing a linear search here. For both finding the steps to maturity and finding the critical stock price, we set either a lower bound or upper bound, to increase or decrease each guess by 0.01. And break the loops whenever it is within the accuracy. And each loop starts with the same bound.
2) We still find it too slow. Then we decide to update the boundary for each time we goes in the loop. Take finding the steps with the relation to each month as an example. We know that with increase of the time, it takes more steps to achieve the required accuracy. Then, we update the lower bound with the previous month before we start to find the step for the current month. The loops do not start to search at the beginning, which saves us some time.
3) However, still slow. We decide to use binary search, which is providing both lower and upper bound, and guess the middle value. If it is greater than our expecting value, we guess the new middle value, which the middle point between the upper bound and the current guess. And if it is smaller than our expecting value, the new middle value is going to be the middle point between the lower bound and the current guess. Still take finding the steps to maturity as an example. We give it a arbitrary upper and lower bound. And we set our temp value to half the difference between upper and lower bound. Every time when we compare the guess with the expected value, we have to either increase or

decrease the current guess by temp. Until finally, the previous round is having the same value as this round, and this is when we should break out of the loop. Hence, binary search speed up the program exponentially!

Work Distribution

Jinhao: American option code, graph, question 1, and 3

Feiyang: European option code, table, conception question 2 and 4.

Code Appendix:

```
def Binomial(Option, K, T, S0, sigma, r, q, N, Exercise):
    delta = T/N
    u = np.e **(sigma *np.sqrt(delta))
    d = np.e **(-sigma*np.sqrt(delta))
    p =(np.e **((r-q)*delta)-d)/(u-d)
    if (Exercise=="E"):
        model = []
        if(Option=="C"):
            for i in range(N+1):
                model.append(max(S0 * (u**(N-i)) * (d**i)-K,0))
        else: # European put
            for i in range(N+1):
                model.append(max(-S0 * (u**(N-i)) * (d**i)+K,0))
        for i in range(N):
            temp =[]
            for x in range(N - i):
                temp.append(np.e**((-r) * delta) * (p * model[x] + (1-p) * model[x+1]))
            model = temp
    else:     # American
        node = []
        for i in range(N + 1):
            temp = []
            for j in range(i + 1):
                temp.append(S0 * (u ** (i - j)) * (d ** (j)))
            node.append(temp)
        if(Option=="P"):
            for i in range(len(node[-1])):
                model.append( K + (node[-1][i]))#np.asarray(node[-1])
                model[i]=max(model[i],0)
            for i in range(N):
                for j in range(N - i):
                    model[j] = max(K-node[N-i-1][j], np.e**((-r)*delta) * ((p) * model[j] + (1-p) * model[j+1]))
        else:   #Option=="C"
            model=[]
            for i in range(len(node[-1])):
                model.append( -K + (node[-1][i]))#np.asarray(node[-1])
                model[i]=max(model[i],0)
            for i in range(N):
                for j in range(N - i):
                    model[j] = max(-K+node[N-i-1][j], np.e**((-r)*delta) * ((p) * model[j] + (1-p) * model[j+1]))

    return model[0]
```

```
if __name__ == '__main__':

    accuracy = 10 ** -3
    number = np.zeros(12)
    temp=1500
    upper=3000
    lower=0
    for n in range(12):
        step = int(temp)
        temp=temp-lower
        while(True):
            temp=int(temp/2)
            print(step)
            pre=step
            value_now = Binomial("P", 100, (n + 1) / 12, 100, 0.2, 0.05, 0, step, "A")

            value_next = Binomial("P", 100, (n + 1) / 12, 100, 0.2, 0.05, 0, step+1, "A")
            if abs(value_next - value_now) > accuracy:
                step=step+temp
            else:
                step=step-temp

            if(step==pre):
                break

        number[n] = step
        lower=step
        temp=(upper-lower)/2+lower
        #print("month ")
    print(number)
```

```
guesstar = np.zeros(12)
upper=155
lower=0

for i in range(12):

    if(i!=0):
        lower = guesstar[i-1]

    temp=(upper-lower)/2
    guess=lower+temp
    while (True):

        temp=temp/2
        pre=guess
        print(guess)
        option_price = Binomial("C", 100, (i+1)/12, guess, 0.2, 0.05, 0.08,int( number[i]), "A")
        if((option_price - guess +100) < 0.005):
            guess =guess-temp
        else:
            guess =guess+temp

        if(int(guess*100)==int(pre*100)):
            break

    guesstar[i]=int(guess*100)/100
    lower=guess
    temp=(upper-lower)/2

    print("month ", i, " :", guesstar[i])
print("0.08call")
print(guesstar)
```

```
if __name__ == '__main__':

    accuracy = 10 ** -3
    x1 = np.arange(12)
    y1 = np.zeros(12)
    temp=1500
    upper=6000
    lower=0
    for n in x1:
        step = int(temp)
        temp=temp-lower
        while(True):
            temp=int(temp/2)
            print(step)
            pre=step
            value_now = Binomial("C", 100, (n + 1) / 12, 100, 0.2, 0.05, 0.08, step, "A")
            value_next = Binomial("C", 100, (n + 1) / 12, 100, 0.2, 0.05, 0.08, step+1, "A")
            if abs(value_next - value_now) > accuracy:
                step=step+temp
            else:
                step=step-temp

            if(step==pre):
                break

        y1[n] = step
        lower=step
        temp=(upper-lower)/2+lower
```

```
guesstar = np.zeros(12)
upper=155
lower=0

for i in range(12):

    if(i!=0):
        lower = guesstar[i-1]

    temp=(upper-lower)/2
    guess=lower+temp
    while (True):

        temp=temp/2
        pre=guess
        print(guess)
        option_price = Binomial("C", 100, (i+1)/12, guess, 0.2, 0.05, 0.08,int( number[i]), "A")
        if((option_price - guess +100) < 0.005):
            guess =guess-temp
        else:
            guess =guess+temp

        if(int(guess*100)==int(pre*100)):
            break

    guesstar[i]=int(guess*100)/100
    lower=guess
    temp=(upper-lower)/2

    print("month ", i, " :", guesstar[i])
print("0.08call")
print(guesstar)
```