IE 420 Project Report

Student Name: Shulu Chen

NetID: Shuluc2

Instructor: Liming Feng

04/22/2020

Content

I. Binomial Model

1.1 Summary of Binomial Model

Here, we will implement the CRR binomial model to price European and American puts and calls on a stock paying continuous dividend yield:

$$\boldsymbol{Binomial(Option, K, T, S_0, \sigma, r, q, N, Exercies)}$$

Where $Option = C$ for calls and $Option = P$ for puts, K is the strike price, T is the time to maturity, S_0 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 $Exercies = A$ for American options while $Exercies = E$ for European options.

For Binomial Model, firstly use Cox-Ross-Rubinstein binomial model to select u and d by volatility σ:

$$u = e^{\sigma\sqrt{\delta}}, \quad d = e^{-\sigma\sqrt{\delta}}$$

Where $\delta = \frac{T}{N}$.

Then, we need to compute the Risk Neutral Probability:

$$p^* = \frac{e^{(r-q)\delta} - d}{u - d}$$

And use Backward Introduction to get the option price:

Start with

$$S_{Nj} = S_0 u^j d^{N-j}$$

$$f_{N,j} = \max(0, S_{Nj} - K) \quad --- Call$$

$$f_{N,j} = \max(0, K - S_{Nj}) \quad --- Put$$

$$j = 0,1, \dots, N$$

For n=N-1, N-2, …, 0:

$$f_{n,j} = e^{-r\delta}\left(p^* f_{n+1,j+1} + (1 - p^*) f_{n+1,j}\right) \quad --- European\ Option$$

$$f_{n,j} = \max\left(e^{-r\delta}\left(p^* f_{n+1,j+1} + (1-p^*) f_{n+1,j}\right), \max\left(K - S_{n,j}, 0\right)\right)$$

$$---American\ put$$

$$f_{n,j} = \max\left(e^{-r\delta}\left(p^* f_{n+1,j+1} + (1-p^*) f_{n+1,j}\right), \max\left(S_{n,j} - K, 0\right)\right)$$

$$---American\ Call$$

$$j = 0,1,\ldots,n$$

Iterate $N-1$ times to get the option price f_0.

1.2 Compute Time

Select N from 1 to 1000 with step size 5, plot the Step-Time figures for European options and American options:

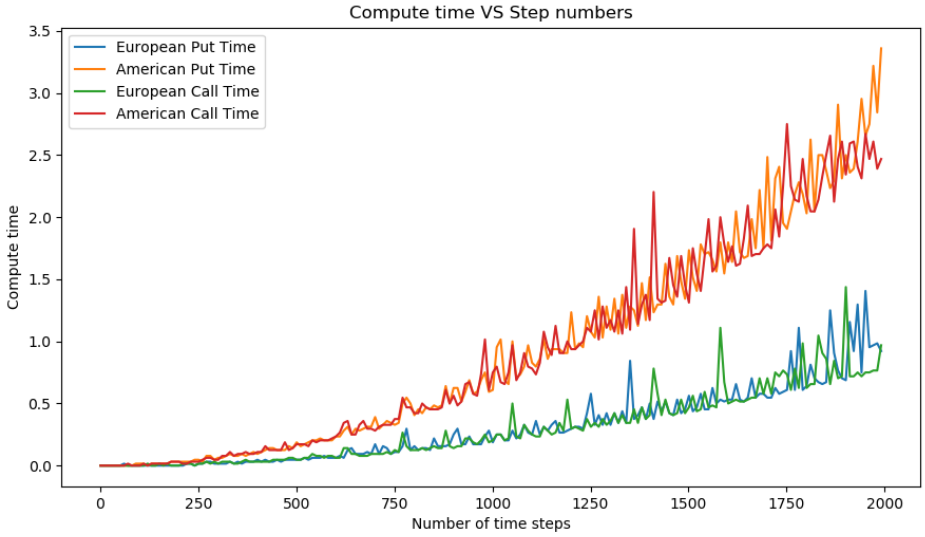

Figure1: Computing time V.S. Number of time steps

As the figure1 shows, the compute time growth greatly with the increase of N. We can also observed that the compute time for American options is larger than European options, one possible illustration is that the compute complexity is higher for American options because compared with European options, American options need to compute intrinsic value for each nodes in the tree, and it need to compare intrinsic value with option value at each nodes, so the time would be longer. We can also observe that compared Puts with Calls, because the compute complexity is the same. The vibration

may because of the CPU's dynamic frequency change.

II. Convergence of European Call

2.1 Summary of Black-Scholes Model

Here, we will use Black-Scholes Option Price as the convergence price, and observe whether the European Call would converge to that price as N increase.

For European Call price:

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

For European Put price:

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

Where $N(X)$ is the CDF of $N(0,1)$, and

$$d_1 = \frac{\ln\left(\frac{S_0}{K}\right) + \left(r - q + \frac{1}{2}\sigma^2\right)T}{\sigma\sqrt{T}}, \quad d_2 = d_1 - \sigma\sqrt{T}$$

2.2 Convergence period

For European Call option, select parameters as:

$$K = 100, \ S_0 = 100, \ r = 0.05, \ q = 0.04, \ \sigma = 0.2$$

N from 1 to 200, plot the option prices, compared with Black-Scholes price and the absolute errors:

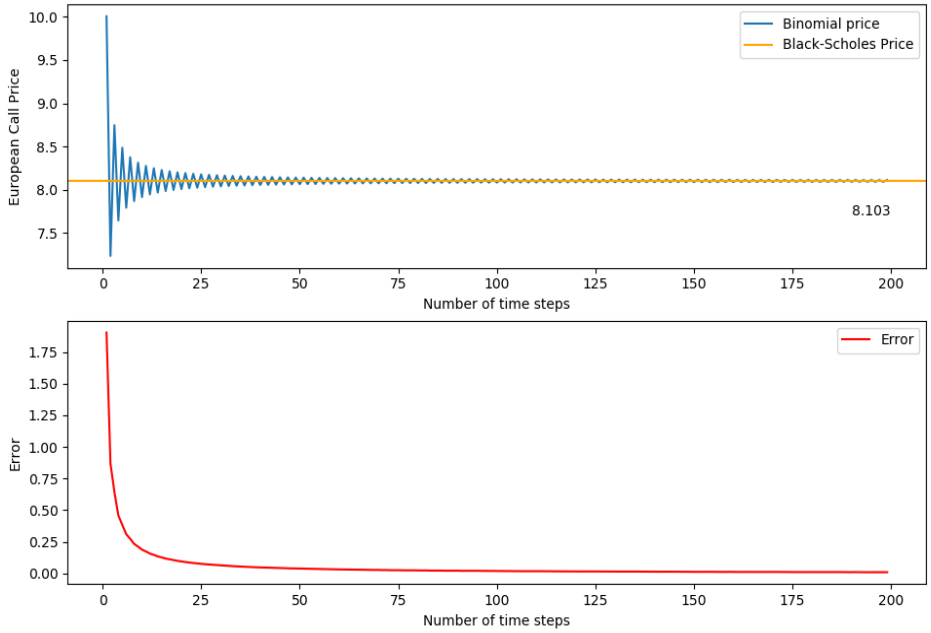

Figure2: Binomial and Black-Scholes prices V.S

Number of time steps and the absolute error.

As the figure shows, the Binomial prices oscillate around the Black-Scholes price, and they converge to Black-Scholes price in the end.

III. Boundary of American Put

Consider American Put with $K = 100,\ \sigma = 0.2,\ r = 0.05,\ q = 0/0.04.\ T = \frac{i}{12}$ for $i = 0,1,\dots,12.$

3.1 Number of time steps for accuracy.

3.1.1 Accuracy analysis

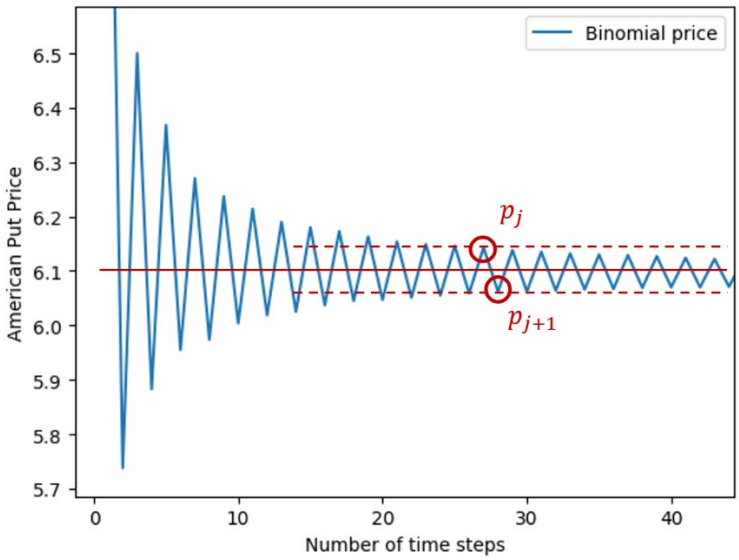

Figure3: American Put price V.S. Number of time steps

For American options, there is no determinate price value as a criterion to count accuracy. From the figure3, we can observe that the option price oscillates converge to a center line, and $(\frac{p_j+p_{j+1}}{2})$ is close enough to its convergence value. Thus, we can design a criterion for accuracy:

$$if\ (p_{j+1} - \left(\frac{p_j + p_{j+1}}{2}\right)) \leq 0.001$$

We got the 10^{-3} accuracy.

3.1.2 Programing Optimization

From experiment, the model is time consuming when N goes large and the option price converge very slow. Thus, we can choose a reasonable odd step size, e.g. 5, to search for the optimal solution. And with the increase of time T, the suitable time steps N growth as well. So, we could use the result N_i of time $i/12$ as the start to search for the N_{i+1}. The above two methods could save a lot of time.

3.1.3 Result for required N

The following table and plot show the number of time steps needed to achieve the required 10^{-3} accuracy.

Maturity Time (T)	Number of time steps when q= 0 (N_0^*)	Number of time steps when q= 0.04 ($N_{0.04}^*$)
1/12	388	397
2/12	583	694
3/12	607	700
4/12	610	919
5/12	640	925
6/12	697	931
7/12	901	2449
8/12	904	2452
9/12	910	2455
10/12	916	2458
11/12	934	2941
12/12	940	3043

Table1: Number of time steps required for accuracy for American Put

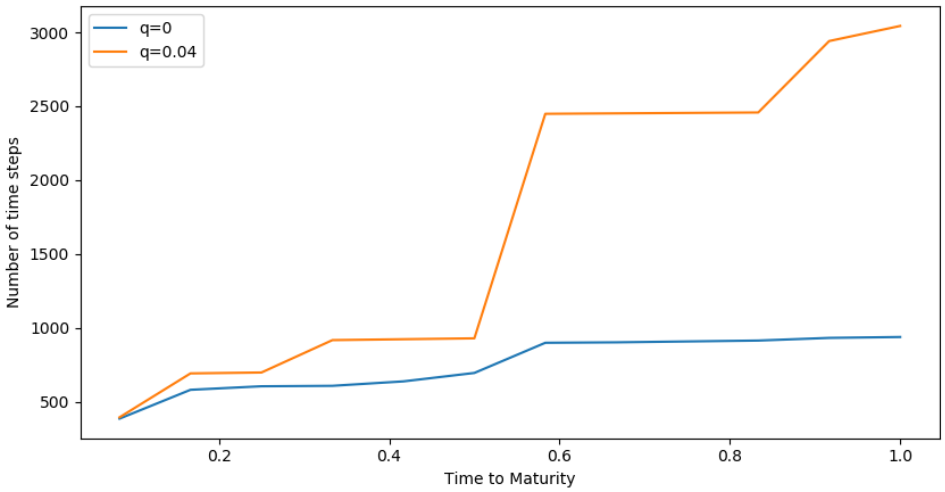

Figure4: Number of time steps required for accuracy for American Put

3.2 Price of 12-month put

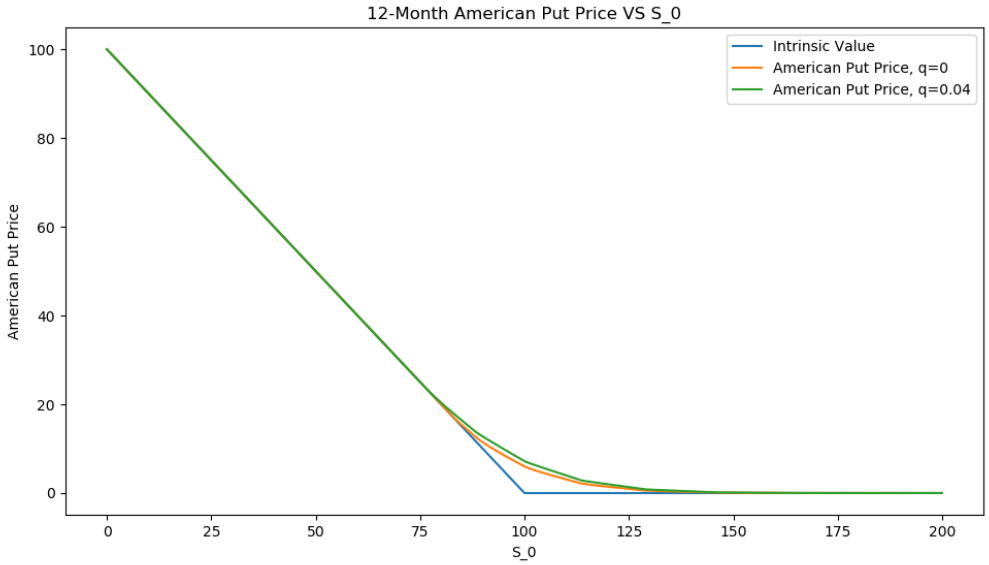

Figure5: 12-Month American Put Price V.S. S_0

3.3 Early exercise boundary

3.3.1 Summary of early exercise boundary

The Critical stock price $S^*(i)$ on the early exercise boundary is the largest stock price at which the difference between the option price and the intrinsic value is less than 0.005. It can be illustrated as:

$$\max_{S_0} \ (f_0 - \max(K - S_0, 0))$$

$$s.t. \quad f_0 - \max(K - S_0, 0) \leq 0.005$$

To find the suitable S^*, we can search back from the strike price K and select the first S_0 which satisfies the constrain that $f_0 - (\mathrm{K} - \mathrm{S}_0) \leq 0$.

3.3.2 Programming Optimization

The huge time consuming is a big issue in this problem, because if we search S_0 from 100 to 70 with the step-size 0.01, it would compute the model for nearly 3000 times for

each T. And in order to guarantee the accuracy, the number of time steps should be large enough. Thus, optimizing program is necessary.

The following figure is shows different maturity time T_i compared with intrinsic value. From this figure, we know that the early exercise boundary would monotonically increasing with the growth of maturity time. Thus, we can compute the boundary from the S_i^* which is the S^* for maturity time i/12.

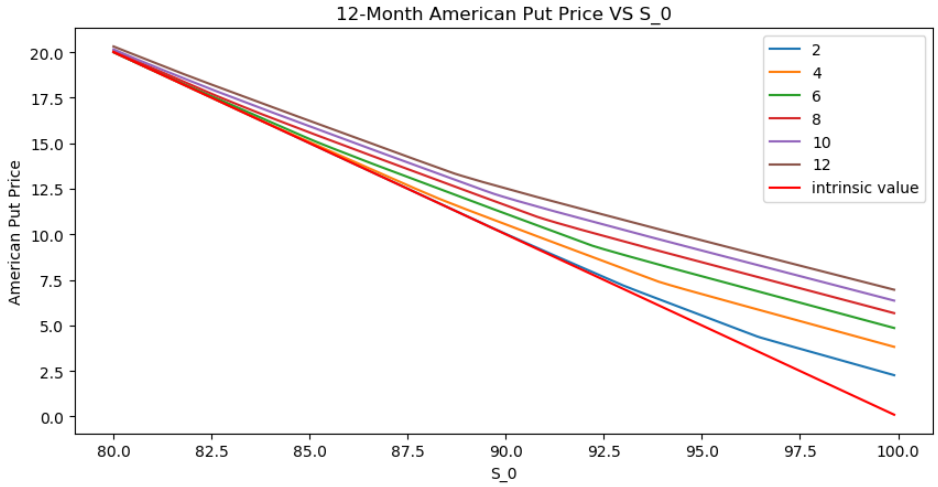

Figure6: Compare different maturity time

Also, we can observe that if N is small, the model would be fast, then we can design the program to roughly search for the possible S^* on a small N, e.g. 100 firstly, and then use the required time steps, e.g. 3000, to search around the possible S^* and find the optimal one.

3.3.3 Results of early exercise boundary

Maturity Time (T)	Early exercise boundary when q= 0 (S^*)	Early exercise boundary when q= 0.04 (S^*)
1/12	91.30	88.91
2/12	88.94	85.49
3/12	87.36	83.25
4/12	86.19	81.52
5/12	85.27	80.14
6/12	84.49	78.97
7/12	83.80	77.97
8/12	83.22	77.08
9/12	82.69	76.27
10/12	82.22	75.58
11/12	81.80	74.94
12/12	81.41	74.33

Table2: Early exercise boundary for American puts

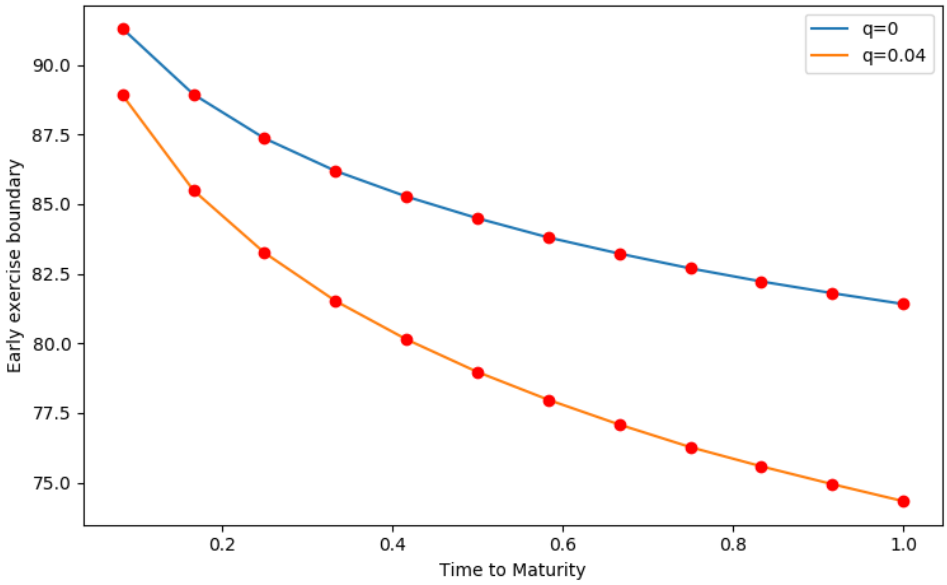

Figure7: Early exercise boundary V.S. Time to Maturity for American Put

3.4 Conclusion of dividend yield

From **figure5** and **figure7**, we can observe that when q goes from 0 to 0.04, the time value of put price would be higher, and for each maturity time, the early exercise boundary of q=0.04 is less than q=0. And we can conclude that with the dividend yield growth, the gap between intrinsic value and option price would increase.

From previous analysis, the dividend yield q only effects the Risk Neutral Probability:

$$p^* = \frac{e^{(r-q)\delta} - d}{u - d}$$

And p^* would decrease if q increase. Then, for American Put price:

$$f_{n,j} = \max\left(e^{-r\delta}\left(p^* f_{n+1,j+1} + (1-p^*) f_{n+1,j}\right), max\left(0, K - S_{n,j}\right)\right)$$

The chance that $e^{-r\delta}\left(p^* f_{n+1,j+1} + (1-p^*) f_{n+1,j}\right) > max\left(0, K - S_{n,j}\right)$ would increase, hence the gap would be wider.

IV. Boundary of American Call

The process of this part is similar with part 3. Consider American Call with K=100, σ=0.2, r=0.05, q=0.04/0.08. T=i/12 for i=0, 1, …, 12.

4.1 Number of time steps for accuracy

Maturity Time (T)	Number of time steps when q= 0.04 ($N^*_{0.04}$)	Number of time steps when q= 0.08 ($N^*_{0.08}$)
1/12	415	433
2/12	808	652
3/12	814	754
4/12	1861	757
5/12	1867	760
6/12	1870	811
7/12	1873	904
8/12	1876	1477
9/12	2293	1684
10/12	2296	1687
11/12	2302	1693
12/12	2305	1696

Table3: Number of time steps required for accuracy for American Call

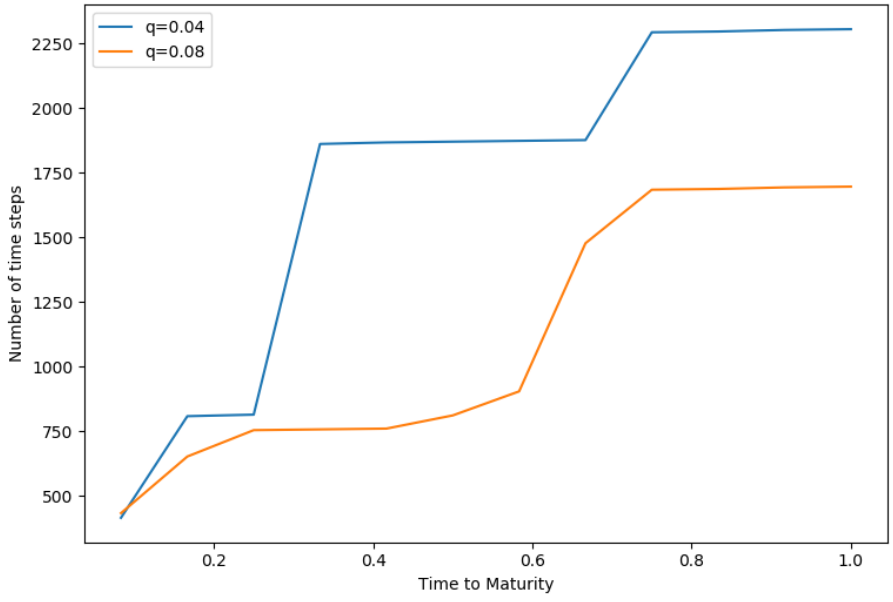

Figure8: Number of time steps required for accuracy for American Call

4.2 Price of 12-month call

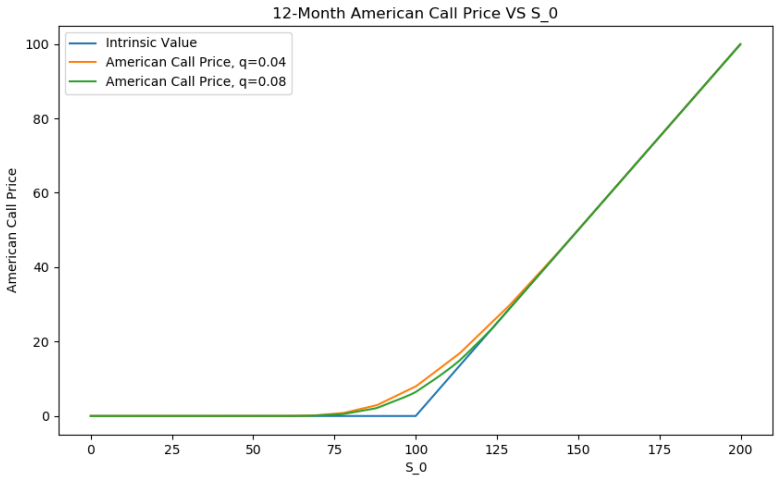

Figure9: 12-Month American Call Price V.S. S_0

4.3 Early exercise boundary

Maturity Time (T)	Early exercise boundary when q= 0.04 (S^*)	Early exercise boundary when q= 0.08 (S^*)
1/12	124.81	110.51
2/12	128.58	113.89
3/12	132.14	116.23
4/12	135.33	118.03
5/12	138.17	119.53
6/12	140.70	120.81
7/12	142.99	121.92
8/12	145.05	122.89
9/12	146.96	123.76
10/12	148.74	124.56
11/12	150.39	125.28
12/12	151.95	125.96

Table4: Early exercise boundary for American calls

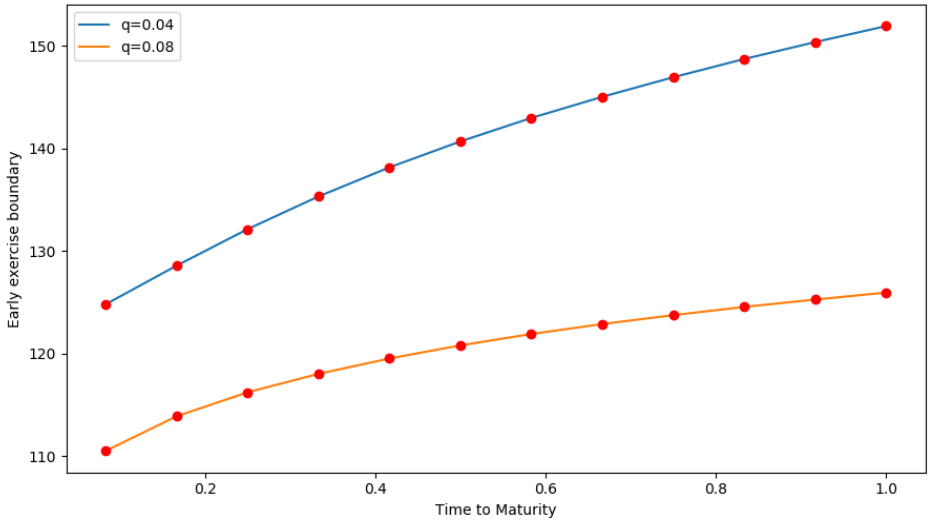

Figure10: Early exercise boundary V.S. Time to Maturity for American Put

4.4 Conclusion of dividend yield

From **figure9** and **figure10**, we can observe that when q goes from 0.04 to 0.08, the put price would be lower, and for each maturity time, the early exercise boundary of q=0.04 is less than q=0.08. And we can conclude that with the dividend yield growth, the gap between intrinsic value and option price would decrease.

Appendix (Python Codes)

April 23, 2020

```
import math
from scipy.stats import norm
import time
import matplotlib.pyplot as plt
import numpy as np
```

1 Binomial model function

```
def Binomial(Option,K,T,S,sigma,r,q,N,Exercise):
    start=time.process_time()
    delta=T/N
    u=math.exp(sigma*math.sqrt(delta))
    d=math.exp(-sigma*math.sqrt(delta))
    p = (math.exp((r - q) * delta) - d) / (u - d)
    if Exercise=="E":
        if Option=="C":
            #European Call
            f = []
            for j in range(N + 1):
                S_Nj = S * pow(u, j) * pow(d, N - j)
                f_Nj = max(0, S_Nj - K)
                f.append(f_Nj)
            for n in reversed(range(N)):
                f_n = []
                for j in range(n+1):
                    f_nj = math.exp(-r*delta)*(p*f[j+1]+(1-p)*f[j])
                    f_n.append(f_nj)
                f = f_n
            option_price = f[0]
        elif Option=="P":
            #European Put#
            f=[]
            for j in range(N+1):
                S_Nj=S*pow(u,j)*pow(d,N-j)
                f_Nj=max(0,K-S_Nj)
                f.append(f_Nj)
```

```
        for n in reversed(range(N)):
            f_n=[]
            for j in range(n+1):
                f_nj=math.exp(-r*delta)*(p*f[j+1]+(1-p)*f[j])
                f_n.append(f_nj)
            f=f_n
        option_price=f[0]
    else:
        print("wrong Option input")
elif Exercise=="A":
    if Option=="C":
        #American Call#
        f=[]
        for j in range(N+1):
            S_Nj=S*pow(u,j)*pow(d,N-j)
            f_Nj=max(0,S_Nj-K)
            f.append(f_Nj)
        for n in reversed(range(N)):
            f_n=[]
            for j in range(n+1):
                f_nj=max(max(0,S*pow(u,j)*pow(d,n-j)-K),math.exp(-r*delta)*/
                         (p*f[j+1]+(1-p)*f[j]))
                f_n.append(f_nj)
            f=f_n
        option_price=f[0]
    elif Option=="P":
        #American Put#
        f=[]
        for j in range(N+1):
            S_Nj=S*pow(u,j)*pow(d,N-j)
            f_Nj=max(0,K-S_Nj)
            f.append(f_Nj)
        for n in reversed(range(N)):
            f_n=[]
            for j in range(n+1):
                f_nj=max(max(0,K-S*pow(u,j)*pow(d,n-j)),math.exp(-r*delta)*/
                         (p*f[j+1]+(1-p)*f[j]))
                f_n.append(f_nj)
            f=f_n
        option_price=f[0]
    else:
        print("wrong Option input")
else:
    print("wrong Exercise input")
option_price=round(option_price,3)
end=time.process_time()
return option_price,start-end
```

2 Function to Count step needed for 10^3 accuracy

```
def count_step(op,qt):
    T3=np.arange(1/12,13/12,1/12)
    step_list=[]
    p_tmp=0
    n=1
    for i in T3:
        for j in np.arange(n,5000,3):
            p=Binomial(op,100,i,100,0.2,0.05,qt,j,"A")
            if abs(p-(p+p_tmp)/2)<0.001:
                step_list.append(j)
                print(i,j)
                n=j
                break
            p_tmp=p
    return step_list
```

3 Function to count early exercise boundary for puts

```
def count_put_s(q,N):
    n=10000
    s_star=[]
    for j in range(0,12):
        for i in reversed(range(n)):
            intrinsic_value_tmp=max((100 - S[i]), 0)
            p_tmp=Binomial("P",100,T[j],S[i],0.2,0.05,q,100,"A")
            dif_tmp=abs(intrinsic_value_tmp-p_tmp)
            if dif_tmp<0.005:
                print(S[i],"i")
                st=i
                for k in reversed(range(st)):
                    p=Binomial("P",100,T[j],S[k],0.2,0.05,q,N,"A")
                    print(S[k],"k")
                    intrinsic_value=max((100-S[k]),0)
                    dif=abs(intrinsic_value-p)
                    print(dif)
                    if dif<0.005:
                        s_star.append(S[k])
                        print(S[k])
                        n=k
                        break
                break
    return s_star
```

4 Function to count early exercise boundary for calls

```
def count_call_s(q,N):
    n=10000
    s_star=[]
    for j in range(0,12):
        for i in range(n,20000):
            intrinsic_value_tmp=max((S[i]-100), 0)
            p_tmp=Binomial("C",100,T[j],S[i],0.2,0.05,q,100,"A")
            dif_tmp=abs(intrinsic_value_tmp-p_tmp)
            if dif_tmp<0.005:
                print(S[i],"i")
                st=i
                for k in range(st,20000):
                    p=Binomial("C",100,T[j],S[k],0.2,0.05,q,N,"A")
                    print(S[k],"k")
                    intrinsic_value=max((S[k]-100),0)
                    dif=abs(intrinsic_value-p)
                    print(dif)
                    if dif<0.005:
                        s_star.append(S[k])
                        print(S[k])
                        n=k
                        break
                break
    return s_star
```

5 Compute Black-Scholes Option Price

```
Option2="C"
K2=100
T2=1
S2=100
sigma2=0.2
r2=0.05
q2=0.04
N2=range(1,200)
Exercise2="E"
d1 = (math.log(S2/K2)+(r2-q2+0.5*pow(sigma2,2))*T2)/(sigma2*math.sqrt(T2))
d2 = d1-sigma2*math.sqrt(T2)
BS_price = S2*math.exp(-q2*T2)*norm.cdf(d1)-K2*math.exp(-r2*T2)*norm.cdf(d2)
```

6 Compute Number of Time Steps needed for Accuracy

```
step_list1=count_step("P",0)
step_list2=count_step("P",0.04)
```

```
step_list3=count_step("C",0.04)
step_list4=count_step("C",0.08)
```

7 Compute Boundary S^*

```
s_star1=count_put_s(0,940)
s_star2=count_put_s(0.04,3043)
s_star3=count_call_s(0.04,2305)
s_star4=count_call_s(0.08,1696)
```

8 Plot Binomial Price compared with Black-Schole Price

```
p_list=[]
error_list=[]
for i in N2:
    p=Binomial(Option2,K2,T2,S2,sigma2,r2,q2,i,Exercise2)
    p_list.append(p)
    error_list.append(abs(p-BS_price))
plt.subplot(2, 1, 1)
plt.plot(N2, p_list,label='Binomial price')
plt.axhline(y=BS_price,label='Black-Scholes Price',c="orange")
plt.annotate(s=round(BS_price,3) ,xy=(0,8) ,xytext=(190,7.7))
plt.ylabel('European Call Price')
plt.xlabel("Number of time steps")
plt.legend()
plt.subplot(2, 1, 2)
plt.plot(N2, error_list,label='Error',c="red")
plt.ylabel('Error')
plt.xlabel("Number of time steps")
plt.legend()
plt.show()
```

9 Plot Number of time steps V.S running time figures

```
Option2="P"
K2=100
T2=1
S2=100
sigma2=0.2
r2=0.05
q2=0.04
N2=range(1,2000,10)
time_e=[]
time_a=[]
time_ec=[]
time_ac=[]
```

```
for i in N2:
    p1,t1=Binomial("P",K2,T2,S2,sigma2,r2,q2,i,"E")
    p2,t2=Binomial("P",K2,T2,S2,sigma2,r2,q2,i,"A")
    p3,t3=Binomial("C",K2,T2,S2,sigma2,r2,q2,i,"E")
    p4, t4 = Binomial("C", K2, T2, S2, sigma2, r2, q2, i, "A")
    time_e.append(t1)
    time_a.append(t2)
    time_ec.append(t3)
    time_ac.append(t4)
plt.plot(N2, time_e,label='European Put Time')
plt.plot(N2, time_a,label='American Put Time')
plt.plot(N2, time_ec,label='European Call Time')
plt.plot(N2, time_ac,label='American Call Time')
plt.title('Compute time VS Step numbers')
plt.ylabel('Compute time')
plt.xlabel("Number of time steps")
plt.legend()
plt.show()
```

10 Plot option price V.S. S_0

```
k3=100
S3=np.arange(0,200,0.1)
intrinsic_value=[]
p_list3=[]
p_list4=[]
gap=[]
for i in range(2000):
    v=max((S3[i]-k3),0)
    intrinsic_value.append(v)
    p=Binomial("C",k3,1,S3[i],0.2,0.05,0.04,10,"A")
    p2=Binomial("C",k3,1,S3[i],0.2,0.05,0.08,10,"A")
    p_list3.append(p)
    p_list4.append(p2)
plt.plot(S3, intrinsic_value,label="Intrinsic Value")
plt.plot(S3,p_list3,label="American Call Price, q=0.04")
plt.plot(S3,p_list4,label="American Call Price, q=0.08")
plt.title('12-Month American Call Price VS S_0')
plt.ylabel('American Call Price')
plt.xlabel("S_0")
plt.legend()
plt.show()
```