Computational HW4_ Xiangui Mei

February 7, 2019

```
In [1]: # python set up
        import matplotlib.pyplot as plt
        import math as m
        import numpy as np
        import pandas_datareader as web
        import pandas as pd

In [ ]: Q1

In [ ]: Compare the convergence rates of the four methods below by doing the following:

In [3]: # Q1
        # define parameters
        rf=0.05
        sigma=0.24
        S0=32.0
        K=30.0
        T=0.5
        steps=np.array([10,20,40,80,100,200,500])
        dt=T/steps
        s=7

        def Europeancall(steps,S0,K,rf,dt,u,d,p_up,p_down):
        # get the stock price
            R=np.exp(rf*dt)
            Rinv=1.0/R
            uu=u/d
            prices=np.zeros(steps+1)
            prices[0]=S0*(d**steps)
            for i in range(1,steps+1):
                prices[i]=uu*prices[i-1]
        # get the call price
            call_value=np.zeros(steps+1)
            for i in range (steps+1):
                call_value[i]=max(0.0,prices[i]-K)
            n=steps-1
            while n>=0:
                for i in range(n+1):
```

```
                call_value[i]=(p_up*call_value[i+1]+p_down*call_value[i])*Rinv
            n=n-1
        return call_value[0]
```

In [ ]: (a)

```
In [4]: # Q1(a)
        c = 0.5*(np.exp(-rf*dt)+np.exp((rf+sigma**2)*dt))
        d = c - np.sqrt(c**2-1)
        u = 1.0/d
        p_up = (np.exp(rf*dt)-d)/(u-d)
        p_down = 1.0-p_up

        callA = [0]*len(steps)
        for i in range(len(steps)):
            callA[i] = Europeancall(steps[i],S0,K,rf,dt[i],u[i],d[i],p_up[i],p_down[i])
```

In [ ]: (b)

```
In [5]: # Q1(b)
        u_b=np.exp(rf*dt)*(1.0+np.sqrt(np.exp((sigma**2)*dt)-1))
        d_b=np.exp(rf*dt)*(1.0-np.sqrt(np.exp((sigma**2)*dt)-1))
        p_upb = 0.5
        p_downb=1.0-p_upb

        callB = [0]*len(steps)
        for i in range(len(steps)):
            callB[i] = Europeancall(steps[i],S0,K,rf,dt[i],u_b[i],d_b[i],p_upb,p_downb)
```

In [ ]: (c)

```
In [7]: # Q1(c)
        u_c = np.exp((rf-(sigma**2)/2)*dt+sigma*np.sqrt(dt))
        d_c= np.exp((rf-(sigma**2)/2)*dt-sigma*np.sqrt(dt))
        p_upc = 0.5
        p_downc=1.0-p_upb

        callC = [0]*len(steps)
        for i in range(len(steps)):
            callC[i] = Europeancall(steps[i],S0,K,rf,dt[i],u_c[i],d_c[i],p_upc,p_downc)
```

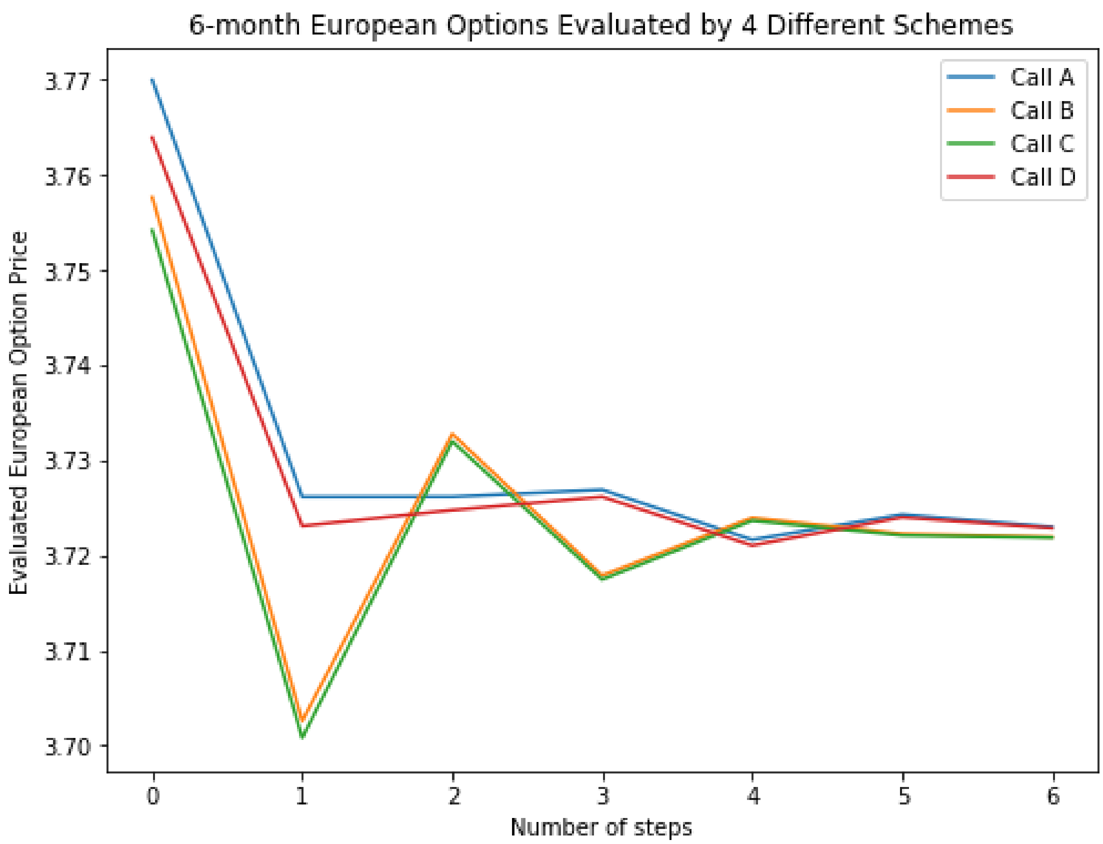
In [ ]: (d)

```
In [9]: # Q1(d)
        u_d = np.exp(sigma*np.sqrt(dt))
        d_d = np.exp(-sigma*np.sqrt(dt))
        p_upd=0.5+0.5*((rf-(0.5*sigma**2))*np.sqrt(dt)/sigma)
        p_downd=1.0-p_upd
```

```
callD = [0]*len(steps)
for i in range(len(steps)):
    callD[i] = Europeancall(steps[i],S0,K,rf,dt[i],u_d[i],d_d[i],p_upd[i],p_downd[i])
```

In [10]:
```
plt.figure(figsize=(8,6))

plt.plot(callA, label="Call A")
plt.plot(callB, label="Call B")
plt.plot(callC, label="Call C")
plt.plot(callD, label="Call D")
plt.title("6-month European Options Evaluated by 4 Different Schemes")
plt.xlabel("Number of steps")
plt.ylabel("Evaluated European Option Price")
plt.legend()
plt.show()
```



In [ ]:
```
Comment:
by using differnet method to caculate u, d and p, the covergence rate is different.
But after the number of steps reaching 6, evaluated european call option prices
converged to the same level.
```

In [ ]: Q2

In [ ]: Take the current price of GOOG, use binomial tree to caculate the option price.

In [ ]: (a) Compare your estimated option price with the one you can get from Bloomberg
        or finance.yahoo.com and comment

In [19]:
```
# Set the date range to be Google's last 60 months price
start_date = '2014-02-05'
end_date = '2019-02-05'

# User pandas_reader.data.DataReader to load Google's data (remove empty prices)
gp=web.DataReader('GOOG', 'yahoo', start_date, end_date).dropna(axis=1,how='all')
gr=gp[['Adj Close']]/ gp[['Adj Close']].shift(1) - 1.0
std_g=np.array(np.std(gr)*np.sqrt(252))

# get the call price for GOOG
# Define parameters
r = 0.02
Tg = 1.0
n = 365
Sg = np.array(gp[['Adj Close']].iloc[len(gp)-1])
Kg = (1.1*Sg)// 10 * 10

dt = Tg/n
pg = 0.5
ug = np.exp((r-(std_g**2)/2)*dt+std_g*np.sqrt(dt))
dg = np.exp((r-(std_g**2)/2)*dt-std_g*np.sqrt(dt))

call_g = Europeancall(n,Sg,Kg,r,dt,ug,dg,pg,1.0-pg)

print ("By using the binomial tree, the call option price for GOOG is")
print(round(call_g,4))
```

```
By using the binomial tree, the call option price for GOOG is
72.5352
```

In [ ]: Comment:
        by searching on Yahoo Finance, the true call option price for Google is 56.90.
        The reason of this discrepancy is probably due to the errors inccured in the
        estimation of annualized historical volitality.The historical volitality has
        very little predicting power over of the future volitality of Google returns,
        therefore should not be used to build the binomial model.

In [ ]: (b) If the two are different in part (a), find the volatility that would make your
        estimated price equal to the market price and comment.

In [20]:
```
# To find the implied volatility
print ("By using the binomial tree, the volatility for call option is")
```

```
print (round(std_g,4))
# start with the volatility from binomial model that is close to the true immplied
#volitality
imvol = std_g
ug_start = np.exp((r-(std_g**2)/2)*dt+std_g*np.sqrt(dt))
dg_start = np.exp((r-(std_g**2)/2)*dt-std_g*np.sqrt(dt))
C =  Europeancall(n,Sg,Kg,r,dt,ug_start,dg_start,pg,1.0-pg)
C0 = 56.9
while (C-C0) > 0.1:
    imvol -= 0.001
    ug_imvol = np.exp((r-(imvol**2)/2)*dt+imvol*np.sqrt(dt))
    dg_imvol = np.exp((r-(imvol**2)/2)*dt-imvol*np.sqrt(dt))
    C = Europeancall(n,Sg,Kg,r,dt,ug_imvol,dg_imvol,pg,1.0-pg)
print ("After price adjusting, the implied volatility for call option is")
print(round(imvol,4))
```

```
By using the binomial tree, the volatility for call option is
0.235
After price adjusting, the implied volatility for call option is
0.199
```

In [ ]:
```
Comment:
after price adjusting, now the implied volatility for GOOG call option is
0.1999, which is slightly smaller than the std I caculated by binomial tree (0.235).
Potential reasons are that volatilty derived from history data is higher, but
currently, the market is as volatile as before.
```

In [ ]:
```
Q3
```

In [ ]:
```
Using the Binomial Method (any one of them) estimate greeks.
```

In [21]:
```
# set the parameters
K_3 = 50.0
sd_3 = 0.25
r_3 = 0.03
T_3 = 0.3846
n_3= 200
S0_3 = list(range(20, 81, 2))
p_3=0.5
```

In [ ]:
```
(a)  Delta of the call option as a function of S0
```

In [22]:
```
h_a=0.5
delt=np.zeros(len(S0_3))
dt_3=T_3/n_3
u_3 = np.exp((r_3-(sd_3**2)/2)*dt_3+sd_3*np.sqrt(dt_3))
d_3 = np.exp((r_3-(sd_3**2)/2)*dt_3-sd_3*np.sqrt(dt_3))
```

```
    for j in range(0,len(S0_3)):
        delt[j]=(Europeancall(n_3,S0_3[j]+h_a,K_3,r_3,dt_3,u_3,d_3,p_3,p_3)- \
            Europeancall(n_3,S0_3[j],K_3,r_3,dt_3,u_3,d_3,p_3,p_3))/h_a
```

In [ ]: (b) Delta of the call option, as a function of T

In [23]:
```
h_b=1.5
S_03=49.0
T_3range=list(np.arange(0.0000,0.3846,0.0100))
d_T3=np.array(T_3range)/n_3
u_3b = np.exp((r_3-(sd_3**2)/2)*d_T3+sd_3*np.sqrt(d_T3))
d_3b = np.exp((r_3-(sd_3**2)/2)*d_T3-sd_3*np.sqrt(d_T3))

delt_T=np.zeros(len(T_3range))

for j in range (0,len(T_3range)):
    delt_T[j]=(Europeancall(n_3,S_03+h_b,K_3,r_3,d_T3[j],u_3b[j],d_3b[j],p_3,p_3)- \
        Europeancall(n_3,S_03,K_3,r_3,d_T3[j],u_3b[j],d_3b[j],p_3,1.0-p_3))/h_b
```

In [ ]: (c) Theta of the call option, as a function of S0

In [24]:
```
dt_c=0.1
dt_n=(T_3+dt_c)/n_3
# up and down for T+dt
u_3n = np.exp((r_3-(sd_3**2)/2)*dt_n+sd_3*np.sqrt(dt_n))
d_3n = np.exp((r_3-(sd_3**2)/2)*dt_n-sd_3*np.sqrt(dt_n))
# up and down for T
u_3c = np.exp((r_3-(sd_3**2)/2)*dt_3+sd_3*np.sqrt(dt_3))
d_3c = np.exp((r_3-(sd_3**2)/2)*dt_3-sd_3*np.sqrt(dt_3))

theta=np.zeros(len(S0_3))
for j in range(0,len(S0_3)):
    theta[j]=(Europeancall(n_3,S0_3[j],K_3,r_3,dt_n,u_3n,d_3n,p_3,p_3)- \
        Europeancall(n_3,S0_3[j],K_3,r_3,dt_3,u_3c,d_3c,p_3,p_3))/(-dt_c)
```

In [ ]: (d) Gamma of the call option, as a function of S0

In [25]:
```
h_d=1.5
u_3d = np.exp((r_3-(sd_3**2)/2)*dt_3+sd_3*np.sqrt(dt_3))
d_3d = np.exp((r_3-(sd_3**2)/2)*dt_3-sd_3*np.sqrt(dt_3))

gamma=np.zeros(len(S0_3))
for j in range(0,len(S0_3)):
    gamma[j]=(Europeancall(n_3,S0_3[j]+h_d,K_3,r_3,dt_3,u_3d,d_3d,p_3,p_3)- \
        2*Europeancall(n_3,S0_3[j],K_3,r_3,dt_3,u_3d,d_3d,p_3,p_3)+ \
        Europeancall(n_3,S0_3[j]-h_d,K_3,r_3,dt_3,u_3d,d_3d,p_3,p_3))/(h_d**2)
```

In [ ]: (e) Vega of the call option, as a function of S0

In [26]:
```
h_e=0.01
sd_n= sd_3+h_e
# up and down for sd
u_3e = np.exp((r_3-(sd_3**2)/2)*dt_3+sd_3*np.sqrt(dt_3))
d_3e = np.exp((r_3-(sd_3**2)/2)*dt_3-sd_3*np.sqrt(dt_3))
# up and down for new sd
u_3ne = np.exp((r_3-(sd_n**2)/2)*dt_3+sd_n*np.sqrt(dt_3))
d_3ne = np.exp((r_3-(sd_n**2)/2)*dt_3-sd_n*np.sqrt(dt_3))

vega=np.zeros(len(S0_3))
for j in range(0,len(S0_3)):
    vega[j]=(Europeancall(n_3,S0_3[j],K_3,r_3,dt_3,u_3ne,d_3ne,p_3,p_3)- \
        Europeancall(n_3,S0_3[j],K_3,r_3,dt_3,u_3e,d_3e,p_3,p_3))/h_e
```

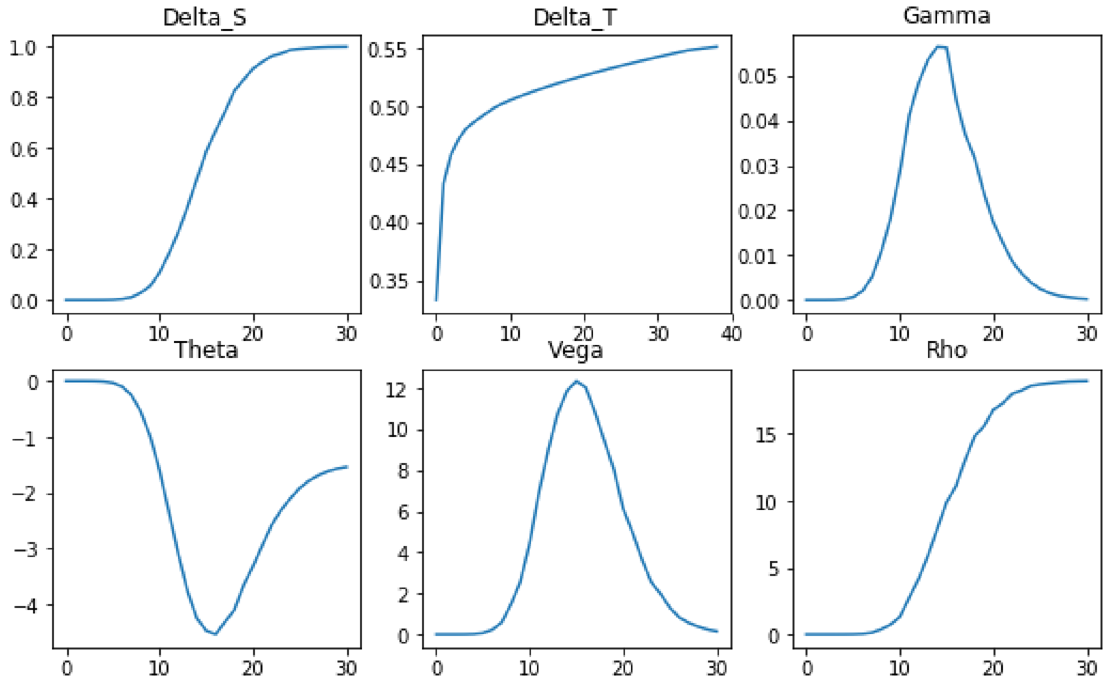
In [ ]:
```
(f) Rho of the call option, as a function of S0
```

In [27]:
```
h_f= 0.01
r_n= r_3+h_f
# up and down for r
u_3f = np.exp((r_3-(sd_3**2)/2)*dt_3+sd_3*np.sqrt(dt_3))
d_3f = np.exp((r_3-(sd_3**2)/2)*dt_3-sd_3*np.sqrt(dt_3))
# up and down for r_n
u_3nf = np.exp((r_n-(sd_3**2)/2)*dt_3+sd_3*np.sqrt(dt_3))
d_3nf = np.exp((r_n-(sd_3**2)/2)*dt_3-sd_3*np.sqrt(dt_3))

rho=np.zeros(len(S0_3))
for j in range(0,len(S0_3)):
    rho[j]=(Europeancall(n_3,S0_3[j],K_3,r_n,dt_3,u_3nf,d_3nf,p_3,p_3)- \
        Europeancall(n_3,S0_3[j],K_3,r_3,dt_3,u_3f,d_3f,p_3,p_3))/h_f
```

In [37]:
```
plt.figure(2, figsize=(10, 6))
plt.subplot(231)
ax1 = plt.plot(delt)
plt.title("Delta_S")
plt.subplot(232)
ax1 = plt.plot(delt_T)
plt.title("Delta_T")
plt.subplot(233)
ax2 = plt.plot(gamma)
plt.title("Gamma")
plt.subplot(234)
ax3 = plt.plot(theta)
plt.title("Theta")
plt.subplot(235)
ax4 = plt.plot(vega)
plt.title("Vega")
plt.subplot(236)
ax5 = plt.plot(rho)
plt.title("Rho")
```

Out[37]: Text(0.5,1,'Rho')



In [ ]: Q4

In [ ]: Use a Binomial Method to estimate the prices of European and American Put option, draw them all in one graph, compare and comment.

In [30]:
```
# get parameters:
T_4=1.0
r_4=0.05
sd_4=0.3
K_4=100.0
n_4=200
dt_4=T_4/n_4
u_4 = np.exp((r_4+(sd_4**2)/2)*dt_4+sd_4*np.sqrt(dt_4))
d_4 = np.exp((r_4-(sd_4**2)/2)*dt_4-sd_4*np.sqrt(dt_4))
p_4=0.5
S0_4=list(range(80, 121, 4))

def Europeanput(steps,S0,K,rf,dt,u,d,p_up,p_down):
# get the stock price
    R=np.exp(rf*dt)
    Rinv=1.0/R
    uu=u/d
    prices=np.zeros(steps+1)
```

```python
    prices[0]=S0*(d**steps)
    for i in range(1,steps+1):
        prices[i]=uu*prices[i-1]
# get the put price
    put_value=np.zeros(steps+1)
    for i in range (steps+1):
        put_value[i]=max(0.0,K-prices[i])
    n=steps-1
    while n>=0:
        for i in range(n+1):
            put_value[i]=(p_up*put_value[i+1]+p_down*put_value[i])*Rinv
        n=n-1
    return put_value[0]

PE=np.zeros(len(range(80, 121, 4)))
for j in range(len(range(80, 121, 4))):
    PE[j]=Europeanput(n_4,S0_4[j],K_4,r_4,dt_4,u_4,d_4,p_4,p_4)

# define the American Option:
def Americanput(steps,S0,K,rf,dt,u,d,p_up,p_down):
# get the stock price
    R=np.exp(rf*dt)
    Rinv=1.0/R
    uu=u/d
    prices=np.zeros(steps+1)
    prices[0]=S0*(d**steps)
    for i in range(1,steps+1):
        prices[i]=uu*prices[i-1]
# get the put price
    put_value=np.zeros(steps+1)
    for i in range (steps+1):
        put_value[i]=max(0.0,K-prices[i])
    n=steps-1
    while n>=0:
        # get the stock price for previous node
        prices_a=np.zeros(n+1)
        prices_a[0]=S0*(d**n)
        for i in range(1,n+1):
            prices_a[i]=uu*prices_a[i-1]
        put_a=np.zeros(n+1)
        for i in range (n+1):
            put_a[i]=max(0.0,K-prices_a[i])
            put_value[i]=max(put_a[i],(p_up*put_value[i+1] \
                                        +p_down*put_value[i])*Rinv)
        n=n-1
    return put_value[0]

PA=np.zeros(len(range(80, 121, 4)))
```

```
for j in range(len(range(80, 121, 4))):
    PA[j]=Americanput(n_4,S0_4[j],K_4,r_4,dt_4,u_4,d_4,p_4,p_4)

plt.plot(PE,color="red",label="PE")
plt.plot(PA,color="blue",label="PA")
plt.title("Put Option Price")
plt.legend()
plt.show()
```


```
In [ ]: In general, the price of American put option is larger than the price of European
        put option, since with the right to exercise early, the value of American Put is higher.
        Moreover, the convergence rate for European put option is also higher than American
        put option.

In [ ]: Q5

In [ ]: Use the Trinomial Method to price a 6-month European Call option,
        use n = 10, 15, 20, 40, 70, 80, 100, 200 and 500 to compute the
        approximate price and draw them in one graph

In [ ]: (a) Use the trinomial method applied to the stock price-process

In [31]: def TriEurocall(steps,S0,K,rf,dt,u,d,p_up,p_mid,p_down):
         # get the stock price
             R=np.exp(rf*dt)
```

```
    Rinv=1.0/R
    prices=np.zeros(2*steps+1)
    prices[0]=S0*(d**steps)
    for i in range(1,2*steps+1):
        prices[i]=u*prices[i-1]
# get the put price
    call_value=np.zeros(2*steps+1)
    for i in range (2*steps+1):
        call_value[i]=max(0.0,prices[i]-K)
    n=steps-1

    while n>=0:
        for i in range(2*steps-1):
            call_value[i]=(p_up*call_value[i+2]+p_mid*call_value[i+1]+p_down* \
                               call_value[i])*Rinv
        n=n-1
    return call_value[0]

T_5 = 0.5
sd_5 = 0.24
r_5 = 0.05
S0_5 = 32.0
K_5 =30.0
N_5 = [10, 15, 20, 40, 70, 80, 100, 200, 500]
dt_5= T_5/np.array(N_5)

d_5=np.array(np.exp(-sd_5*np.sqrt(3*dt_5)))
u_5=1.0/d_5
pd_5=(r_5*dt_5*(1-u_5)+(r_5*dt_5)**2+sd_5**2*dt_5)/((u_5-d_5)*(1.0-d_5))
pu_5=(r_5*dt_5*(1-d_5)+(r_5*dt_5)**2+sd_5**2*dt_5)/((u_5-d_5)*(u_5-1.0))
pm_5=1-pu_5-pd_5

tricall=np.zeros(len(N_5))
for i in range(len(N_5)):
    tricall[i]=TriEurocall(N_5[i],S0_5,K_5,r_5,dt_5[i],\
           u_5[i],d_5[i],pu_5[i],pm_5[i],pd_5[i])
```

In [ ]: (b) Use the trinomial method applied to the Log-stock price-process

In [34]:
```
# Q5(b)
def TriEurocall_p(steps,X0,K,rf,dt,u,d,p_up,p_mid,p_down):
# get the stock price
    R=np.exp(rf*dt)
    Rinv=1.0/R
    prices=np.zeros(2*steps+1)
    prices[0]=np.exp(X0+steps*d)
    for i in range(1,2*steps+1):
        prices[i]=np.exp(np.log(prices[i-1])+u)
```

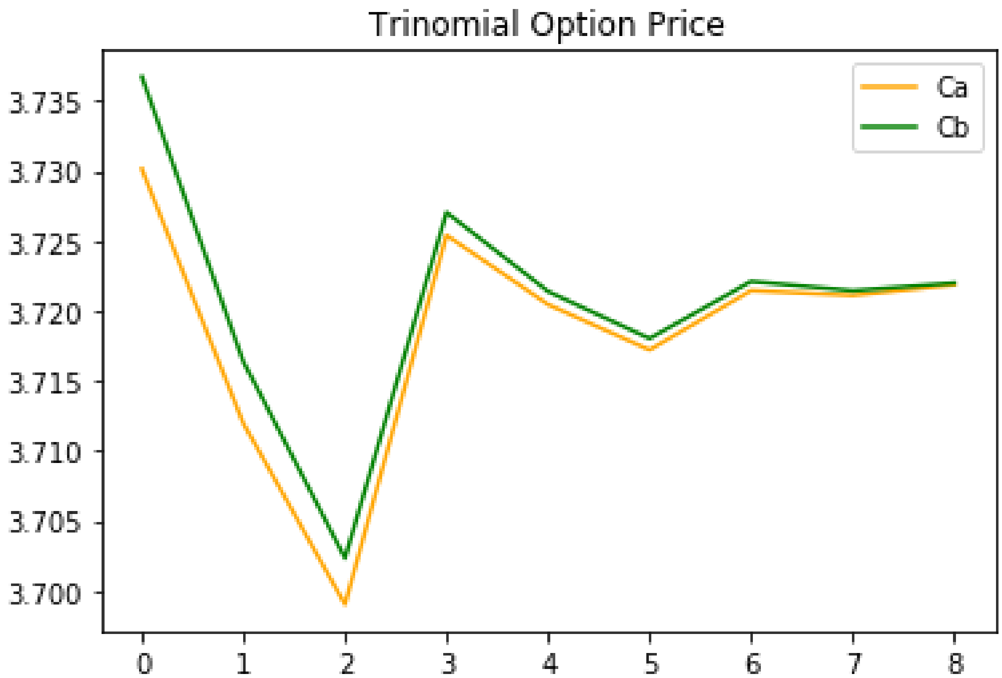
```python
# get the put price
    call_value=np.zeros(2*steps+1)
    for i in range (2*steps+1):
        call_value[i]=max(0.0,prices[i]-K)
    n=steps-1

    while n>=0:
        for i in range(2*steps-1):
            call_value[i]=(p_up*call_value[i+2]+p_mid*call_value[i+1]+ \
                           p_down*call_value[i])*Rinv
        n=n-1
    return call_value[0]

# up, down and middle
X0_5=m.log(S0_5)
dXu = sd_5*np.sqrt(3.0*dt_5)
dXd = -sd_5*np.sqrt(3.0*dt_5)
# find risk-neutral probability, pu, pd, pm
p5_u =0.5*(((r_5-0.5*sd_5**2)**2*(dt_5)**2+sd_5**2*dt_5) \
           /(dXu)**2+((r_5-0.5*sd_5**2)*dt_5)/dXu)
p5_d =0.5*(((r_5-0.5*sd_5**2)**2*(dt_5)**2+sd_5**2*dt_5) \
           /(dXu)**2-((r_5-0.5*sd_5**2)*dt_5)/dXu)
p5_m = 1 - p5_u - p5_d

tricall_p=np.zeros(len(N_5))
for i in range(len(N_5)):
    tricall_p[i]=TriEurocall_p(N_5[i],X0_5,K_5,r_5,dt_5[i],\
           dXu[i],dXd[i],p5_u[i],p5_m[i],p5_d[i])

plt.plot(tricall,color="orange",label="Ca")
plt.plot(tricall_p,color="green",label="Cb")
plt.title("Trinomial Option Price")
plt.legend()
plt.show()
```



In [ ]: Q6

In [ ]: Use Haltons Low-Discrepancy Sequences to price European call options.

In [36]:
```
def  get_Halton(base,n):
         seq=np.zeros(n)
         numbits=1+m.ceil(np.log(n)/np.log(base))
         d=np.zeros(int(numbits))
         a=np.array([i+1 for i in range(int(numbits))])
         b=1.0/base**(a)
         for i in range(n):
             j=0 ; ok =0
             while ok==0:
                 d[j]=d[j]+1
                 if d[j]<base:
                    ok=1
                 else:
                    d[j]=0 ; j = j+1
             seq[i]=np.dot(d,b)
         return seq

# Box-Muller Method
def calloption(b1,b2,n,T,S_0,sigma,K,r):
    U1=get_Halton(b1,n)
    U2=get_Halton(b2,n)
```

```python
    Z6=np.zeros(n)
    for i in range(n):
        Z6[i]=np.sqrt(-2*np.log(U1[i]))*m.cos(2*m.pi*U2[i])

    WT=np.array(m.sqrt(T)* Z6)
    S=S_0*np.exp((r-0.5*(sigma**2))*T+sigma*WT)
    C=np.exp(-r*T)*np.mean([max(0.0,i-K) for i in S])

    return C
# Check the result
print(round(calloption(2,5,1000,0.5,32.0,0.24,30.0,0.05),4))
```

```
3.7168
```