Li_Jiaqi_Project4

February 7, 2019

Followings are functions from previous project:

In [1]:
```
import numpy as np
import math
import matplotlib.pyplot as plt
import scipy.stats as bs

a = 7**5
b = 0
m = 2**31-1

#This function generates uniform distribution
def f_unif(n,x_0):
    U = [None] * (n+1)
    U[0] = x_0
    for i in range(1,(n+1)):
        U[i] = np.mod(a*U[i-1]+b,m)
    del U[0]
    U = [x/m for x in U]
    return U

#This function generates normal distribution
def f_norm(n,U):
    Z_1 = [None]*n
    Z_2 = [None]*n
    for i in range(n):
        Z_1[i] = np.sqrt(-2*np.log(U[2*i]))*np.cos(2*math.pi*U[2*i+1])
        Z_2[i] = np.sqrt(-2*np.log(U[2*i]))*np.sin(2*math.pi*U[2*i+1])
        i = i + 1
    return Z_1+Z_2

#This function generates a brownian motion
def f_w(T,Z):
    W = [np.sqrt(T)*x for x in Z]
    return W

#this function generates any size of Halton Sequence with any base
```

```
def f_HaltonS(base,n):
    seq = np.zeros(n)
    bits = 1+math.ceil(np.log(n)/np.log(base))
    bs = np.array([i+1 for i in range(bits)])
    b = 1/(base**bs)
    d = np.zeros(bits)
    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
```

1 Problem 1

The following function is the Binomial Pricing Model for European Call Option

```
In [2]: rf = 0.05
        sigma = 0.24
        S = 32
        K = 30
        n = np.array([10, 20, 40, 80, 100, 200, 500])

        T = 0.5
        dt = T/n
        s = len(n)

        def f_Euro_Call(n,S,K,rf,dt,u,d,p_up,p_down):
            R = np.exp(rf*dt)
            Rinv = 1/R
            if u == 1/d:
                uu = u*u
            else:
                uu = u/d
            prices = [0]*(n+1)
            prices[0] = S*(d**n)
            for i in range(1,n+1):
                prices[i] = uu*prices[i-1]
            call = [0]*(n+1)
            for i in range(n+1):
                call[i] = max(0, prices[i]-K)
            step = n-1
            while step >= 0:
```

```
            for i in range(step+1):
                call[i] = (p_up*call[i+1]+p_down*call[i])*Rinv
            step = step - 1
        return call[0]
```

(a)

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

        callA = [0]*s
        for i in range(s):
            callA[i] = f_Euro_Call(n[i],S,K,rf,dt[i],u[i],d[i],p_up[i],p_down[i])
```

(b)

```
In [5]: p2 = 0.5
        d2 = np.exp(rf*dt)*(1-np.sqrt(np.exp(sigma**2*dt)-1))
        u2 = np.exp(rf*dt)*(1+np.sqrt(np.exp(sigma**2*dt)-1))

        callB = [0]*s
        for i in range(s):
            callB[i] = f_Euro_Call(n[i],S,K,rf,dt[i],u2[i],d2[i],p2,p2)
```

(c)

```
In [6]: p3 = 0.5
        u3 = np.exp((rf-(sigma**2)/2)*dt+sigma*np.sqrt(dt))
        d3 = np.exp((rf-(sigma**2)/2)*dt-sigma*np.sqrt(dt))

        callC = [0]*s
        for i in range(s):
            callC[i] = f_Euro_Call(n[i],S,K,rf,dt[i],u3[i],d3[i],p3,p3)
```

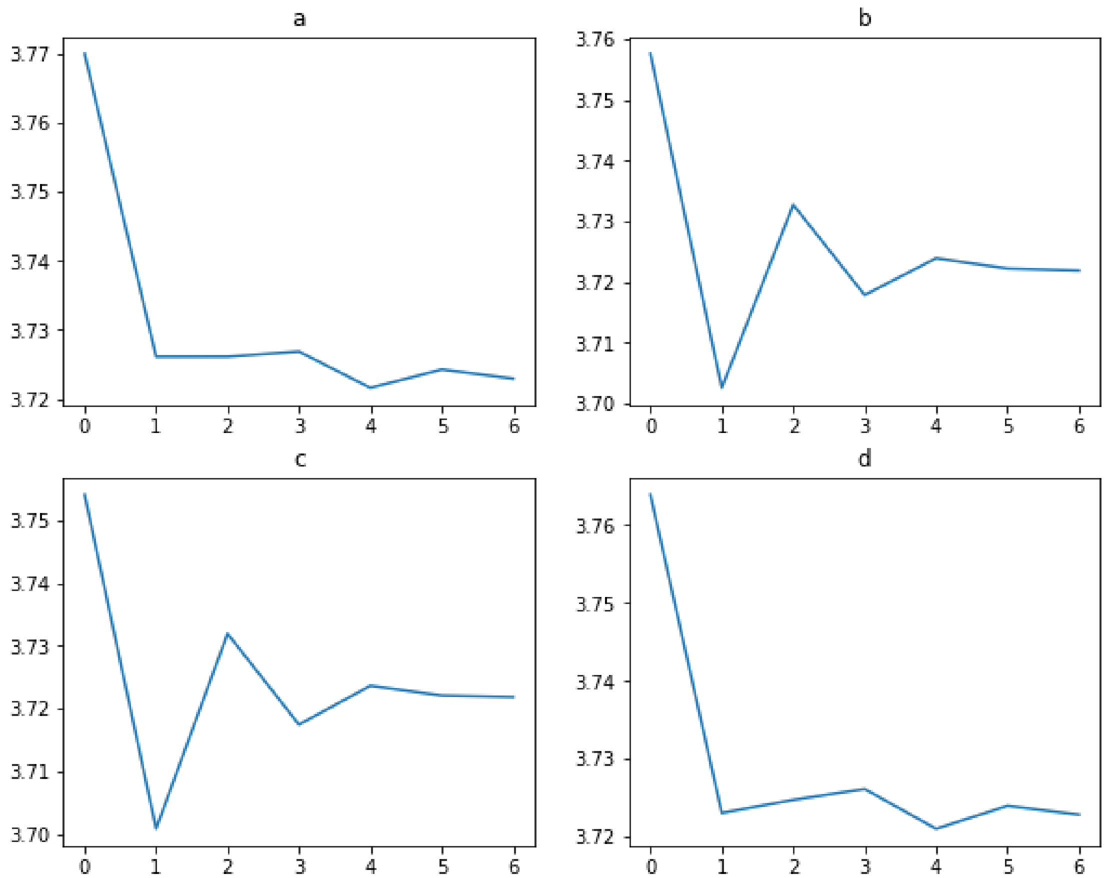
(d)

```
In [7]: p4_up = 0.5+0.5*(rf-0.5*(sigma**2))*np.sqrt(dt)/sigma
        p4_down = 1 - p4_up
        u4 = np.exp(sigma*np.sqrt(dt))
        d4 = np.exp(-sigma*np.sqrt(dt))

        callD = [0]*s
        for i in range(s):
            callD[i] = f_Euro_Call(n[i],S,K,rf,dt[i],u4[i],d4[i],p4_up[i],p4_down[i])
```

plot

In [8]:
```
plt.figure(1, figsize = (10,8))
plt.subplot(221)
ax1 = plt.plot(callA)
plt.title("a")
plt.subplot(222)
ax2 = plt.plot(callB)
plt.title("b")
plt.subplot(223)
ax3 = plt.plot(callC)
plt.title("c")
plt.subplot(224)
ax4 = plt.plot(callD)
plt.title("d")
```

Out[8]: Text(0.5, 1.0, 'd')



In [38]:
```
print("part a method",np.round(np.array(callA),3))
print("part b method",np.round(np.array(callB),3))
print("part c method",np.round(np.array(callC),3))
print("part d method",np.round(np.array(callD),3))
```

```
part a method [3.77  3.726 3.726 3.727 3.722 3.724 3.723]
part b method [3.758 3.703 3.733 3.718 3.724 3.722 3.722]
part c method [3.754 3.701 3.732 3.717 3.724 3.722 3.722]
part d method [3.764 3.723 3.725 3.726 3.721 3.724 3.723]
```

All of these 4 methods finally converge to about 3.72. Comparing the convergence rates, a and d have similar rates while b and c have similar rates. In general, method in part c has the highest convergence rate.

2 Problem 2

(a)

```
In [14]: import pandas_datareader as web
         import datetime

         start = datetime.datetime(2014,2,4)
         end = datetime.datetime(2019,2,4)
         ticker = "GOOG"
         f = web.DataReader(ticker,"yahoo",start,end)

         import pandas
         rf2 = 0.02
         T2 = 1
         t = 500
         dt2 = T2/t

         price = f[['Close']]
         N = len(price)
         ret = f[['Close']]/ f[['Close']].shift(1) - 1.0
         ret = pandas.DataFrame.as_matrix(ret.iloc[1:len(ret)])
         sigma2 = np.std(ret)*np.sqrt(252)
         curP = pandas.DataFrame.as_matrix(price.iloc[N-1])[0]
         K2 = round(curP*1.1/10)*10

         g2 = 0.5*(np.exp(-rf2*dt2)+np.exp((rf2+sigma2**2)*dt2))
         d_2 = g2 - np.sqrt(g2**2-1)
         u_2 = 1/d_2
         p_up_2 = (np.exp(rf2*dt2)-d_2)/(u_2-d_2)
         p_down_2 = 1 - p_up_2
         callGOOG = f_Euro_Call(t,curP,K2,rf2,dt2,u_2,d_2,p_up_2,p_down_2)
         print("option price =",callGOOG)
         print("strike =",K2)
         print("current price =",curP)
         print("volatility =",sigma2)
```

```
option price = 72.29678157146034
strike = 1260.0
```

```
current price = 1145.989990234375
volatility = 0.23465627965146882
```

```
D:\anaconda_distribution\Anaconda\lib\site-packages\ipykernel_launcher.py:18: FutureWarning: Me
D:\anaconda_distribution\Anaconda\lib\site-packages\ipykernel_launcher.py:20: FutureWarning: Me
```

The actual price of a option with strike price = 1260 is $56.9.(based on data from yahho finance on 2/5/2019) This implies that the actual market volatility of google is much more smaller than the estimated volatility based on the 60 month historical daily returns of google stock.

(b)

```
In [15]: test = sigma2
         i = 0
         while np.abs(round(callGOOG,3)-56.90) > 0.01:
             if round(callGOOG,3) > 56.90:
                 test = test - 0.0001
             else:
                 test = test + 0.0001
             g2 = 0.5*(np.exp(-rf2*dt2)+np.exp((rf2+test**2)*dt2))
             d_2 = g2 - np.sqrt(g2**2-1)
             u_2 = 1/d_2
             p_up_2 = (np.exp(rf2*dt2)-d_2)/(u_2-d_2)
             p_down_2 = 1 - p_up_2
             callGOOG = f_Euro_Call(t,curP,K2,rf2,dt2,u_2,d_2,p_up_2,p_down_2)
             i = i+1
         #this will take some time to get the result.
         print(test)
```

```
0.20005627965147263
```

The volatility should be about 20% to make my estimated price equal to the market price.

3 Problem 3

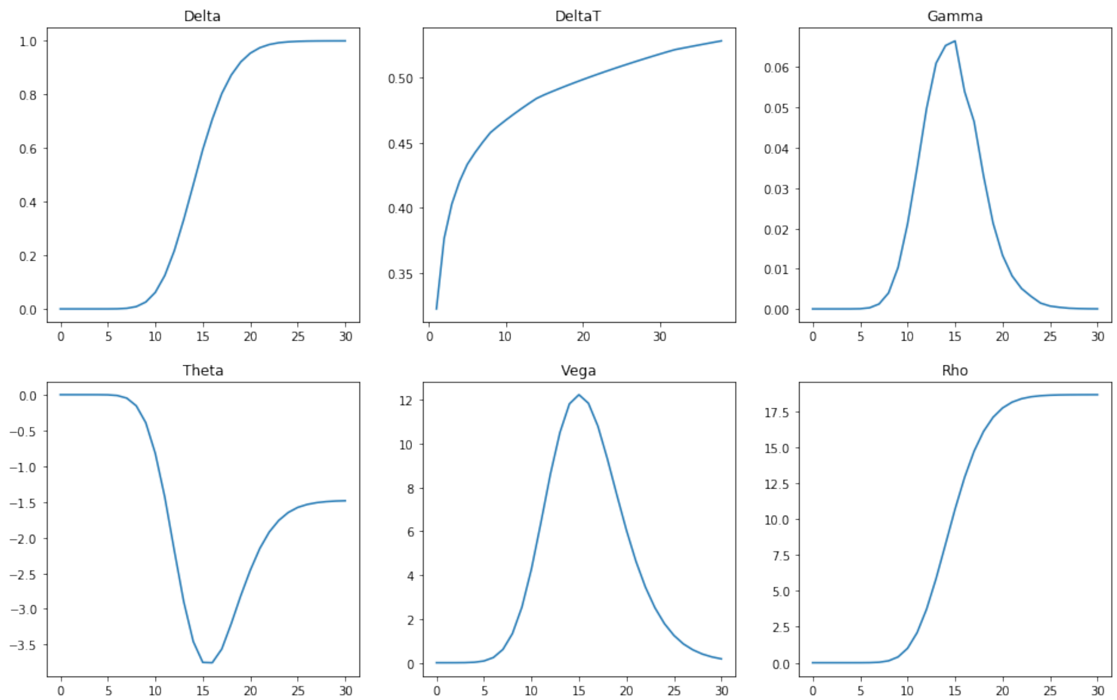
```
In [40]: S0 = 49
         K3 = 50
         rf3 = 0.03
         sigma3 = 0.2
         T3 = 0.3846
         mu = 0.14
         t3 = 500


         def F_Euro_Call(n,S,K,rf,T,sigma):
             dt = T/n
```

```
    g = 0.5*(np.exp(-rf*dt)+np.exp((rf+sigma**2)*dt))
    d = g - np.sqrt(g**2-1)
    u = 1/d
    p_up = (np.exp(rf*dt)-d)/(u-d)
    p_down = 1 - p_up
    R = np.exp(rf*dt)
    Rinv = 1/R
    if u == 1/d:
        uu = u*u
    else:
        uu = u/d
    prices = [0]*(n+1)
    prices[0] = S*(d**n)
    for i in range(1,n+1):
        prices[i] = uu*prices[i-1]
    call = [0]*(n+1)
    for i in range(n+1):
        call[i] = max(0, prices[i]-K)
    step = n-1
    while step >= 0:
        for i in range(step+1):
            call[i] = (p_up*call[i+1]+p_down*call[i])*Rinv
        step = step - 1
    return call[0]

Srange = np.array([i*2 for i in range(10,41)])
N3 = len(Srange)
Trange = np.array([i*0.01 for i in range(39)])

Delta = [0]*N3; Gamma = [0]*N3; Theta = [0]*N3; Vega = [0]*N3;
Rho = [0]*N3; ECp = [0]*N3; ECn = [0]*N3; EC = [0]*N3;
C1 = [0]*N3; DeltaT = [0]*len(Trange); C2 = [0]*len(Trange)
for i in range(N3):
    C1[i] = F_Euro_Call(t3,Srange[i],K3,rf3,T3,sigma3)
    Delta[i] = (F_Euro_Call(t3,Srange[i]+1,K3,rf3,T3,sigma3) - C1[i])/1
    ECp[i] = F_Euro_Call(t3,Srange[i]+1,K3,rf3,T3,sigma3)
    ECn[i] = F_Euro_Call(t3,Srange[i]-1,K3,rf3,T3,sigma3)
    Gamma[i] = (ECp[i] + ECn[i] - 2*np.array(C1[i]))/(1**2)
    Theta[i] = (C1[i] - F_Euro_Call(t3,Srange[i],K3,rf3,T3+0.1,sigma3))/0.1
    Vega[i] = (F_Euro_Call(t3,Srange[i],K3,rf3,T3,sigma3+0.1) - C1[i])/0.1
    Rho[i] = (F_Euro_Call(t3,Srange[i],K3,rf3+0.1,T3,sigma3) - C1[i])/0.1

for i in range(len(Trange)):
    C2[i] = F_Euro_Call(t3,S0,K3,rf3,Trange[i],sigma3)
    DeltaT[i] = (F_Euro_Call(t3,S0+1,K3,rf3,Trange[i],sigma3) - C2[i])/1

plt.figure(2, figsize=(16, 10))
plt.subplot(231)
```

```
ax1 = plt.plot(Delta)
plt.title("Delta")
plt.subplot(232)
ax1 = plt.plot(DeltaT)
plt.title("DeltaT")
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")
```

```
D:\anaconda_distribution\Anaconda\lib\site-packages\ipykernel_launcher.py:15: RuntimeWarning: r
  from ipykernel import kernelapp as app
```

Out[40]: `Text(0.5, 1.0, 'Rho')`

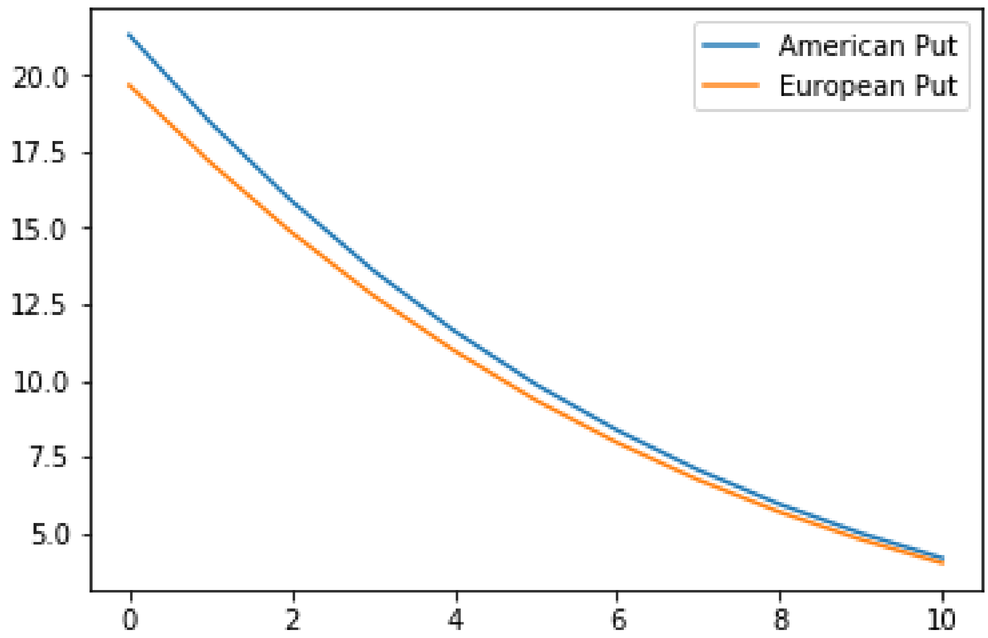


4 Problem 4

```
In [21]: def F_Euro_Put(n,S,K,rf,T,sigma):
             dt = T/n
             g = 0.5*(np.exp(-rf*dt)+np.exp((rf+sigma**2)*dt))
             d = g - np.sqrt(g**2-1)
             u = 1/d
             p_up = (np.exp(rf*dt)-d)/(u-d)
             p_down = 1 - p_up
             R = np.exp(rf*dt)
             Rinv = 1/R
             if u == 1/d:
                 uu = u*u
             else:
                 uu = u/d
             prices = [0]*(n+1)
             prices[0] = S*(d**n)
             for i in range(1,n+1):
                 prices[i] = uu*prices[i-1]
             put = [0]*(n+1)
             for i in range(n+1):
                 put[i] = max(0, K-prices[i])
             step = n-1
             while step >= 0:
                 for i in range(step+1):
                     put[i] = (p_up*put[i+1]+p_down*put[i])*Rinv
                 step = step - 1
             return put[0]

         T4 = 1
         t4 = 500
         rf4 = 0.05
         sigma4 = 0.3
         K4 = 100
         S0_4 = [i*4 for i in range(20,31)]
         N4 = len(S0_4)

         Eput = [0]*N4
         for i in range(N4):
             Eput[i] = F_Euro_Put(t4,S0_4[i],K4,rf4,T4,sigma4)

         n=t4
         S=S0_4[0]
         K=K4
         rf=rf4
         T=T4
         sigma=sigma4
```

```python
def F_Ameri_Put(n,S,K,rf,T,sigma):
    dt = T/n
    g = 0.5*(np.exp(-rf*dt)+np.exp((rf+sigma**2)*dt))
    d = g - np.sqrt(g**2-1)
    u = 1/d
    p_up = (np.exp(rf*dt)-d)/(u-d)
    p_down = 1 - p_up
    R = np.exp(rf*dt)
    Rinv = 1/R
    uu = u*u

    prices = [0]*(n+1)
    prices[0] = S*(d**n)
    for i in range(1,n+1):
        prices[i] = uu*prices[i-1]
    put = [0]*(n+1)
    for i in range(n+1):
        put[i] = max(0, K-prices[i])

    step = n-1
    while step >= 0:
        prices_F = [0]*(n+1)
        prices_F[0] = S*(d**step)
        for i in range(1,step+1):
            prices_F[i] = uu*prices_F[i-1]
        put_F = [0]*(step+1)
        for i in range(step+1):
            put_F[i] = max(0, K-prices_F[i])
        for i in range(step+1):
            put[i] = max((p_up*put[i+1]+p_down*put[i])*Rinv, put_F[i])
        step = step - 1
    return put[0]

Aput = [0]*N4
for i in range(N4):
    Aput[i] = F_Ameri_Put(t4,S0_4[i],K4,rf4,T4,sigma4)

plt.figure(4)
ax1, = plt.plot(Aput, label = "American Put")
ax2, = plt.plot(Eput, label = "European Put")
plt.legend(handles = [ax1,ax2])
```

```
Out[21]: <matplotlib.legend.Legend at 0x2ae4f29fdd8>
```



Based on the graph, we can tell that the American Put Option prices are always higher than the European Put Option Prices.

5 Problem 5

(a)

In [22]:
```
rf5 = 0.05
T5 = 0.5
sigma5 = 0.24
S0_5 = 32
K5 = 30
n5 = np.array([10, 15, 20, 40, 70, 80, 100, 200, 500])
dt5 = T5/n5
N5 = len(n5)

d5 = np.exp(-sigma5*np.sqrt(3*dt5))
u5 = 1/d5
p_up5 = (rf5*dt5*(1-d5)+(rf5*dt5)**2+sigma5**2*dt5)/((u5-d5)*(u5-1))
p_down5 = (rf5*dt5*(1-u5)+(rf5*dt5)**2+sigma5**2*dt5)/((u5-d5)*(1-d5))
p_mid5 = 1 - p_up5 - p_down5

def f_Euro_Call_Tri(n,S,K,rf,dt,u,d,p_up,p_mid,p_down):
    R = np.exp(rf*dt)
    Rinv = 1/R
```

```
    prices = [0]*(2*n+1)
    prices[0] = S*(d**n)
    for i in range(1,2*n+1):
        prices[i] = u*prices[i-1]

    call = [0]*(2*n+1)
    for i in range(2*n+1):
        call[i] = max(0, prices[i]-K)
    step = n-1
    while step >= 0:
        for i in range(2*n-1):
            call[i] = (p_up*call[i+2]+p_mid*call[i+1]+p_down*call[i])*Rinv
        step = step - 1
    return call[0]

Ecall1 = [0]*N5
for i in range(N5):
    Ecall1[i] = f_Euro_Call_Tri(n5[i],S0_5,K5,rf5,dt5[i],u5[i], \
                                d5[i],p_up5[i],p_mid5[i],p_down5[i])
```

(b)

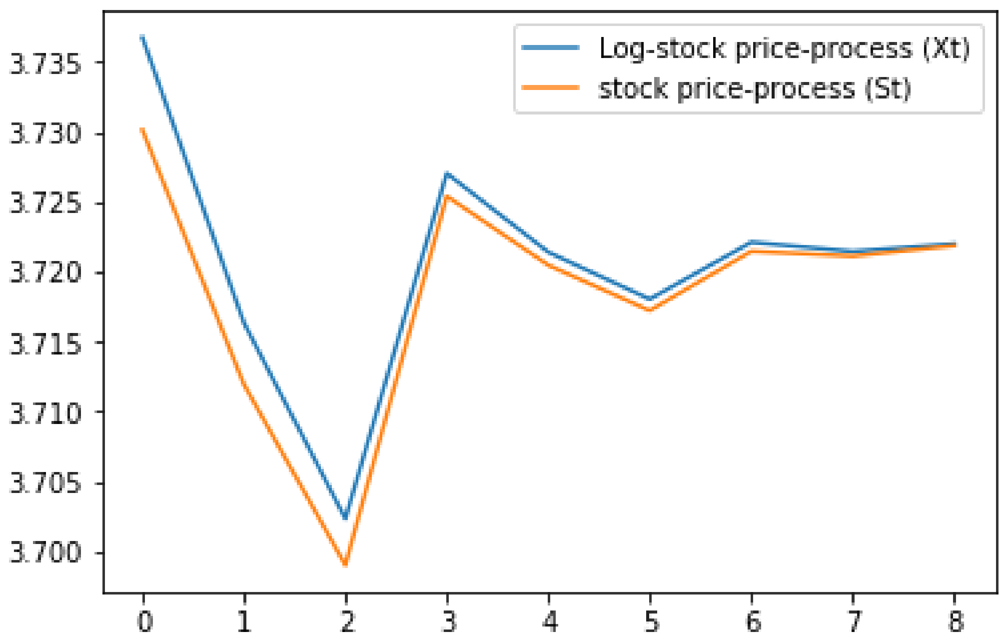
```
In [24]: dXu = sigma5*np.sqrt(3*dt5)
dXd = -sigma5*np.sqrt(3*dt5)

p_up52 =0.5*(((rf5-0.5*sigma5**2)**2*(dt5)**2+sigma5**2*dt5)/(dXu)**2+ \
             ((rf5-0.5*sigma5**2)*dt5)/dXu)
p_down52 =0.5*(((rf5-0.5*sigma5**2)**2*(dt5)**2+sigma5**2*dt5)/(dXu)**2- \
               ((rf5-0.5*sigma5**2)*dt5)/dXu)
p_mid52 = 1 - p_up52 - p_down52

def f_Euro_Call_Tri2(n,S,K,rf,dt,u,d,p_up,p_mid,p_down):
    R = np.exp(rf*dt)
    Rinv = 1/R

    prices = [0]*(2*n+1)
    prices[0] = np.exp(np.log(S)+n*d)
    for i in range(1,2*n+1):
        prices[i] = np.exp(np.log(prices[i-1]) + u)

    call = [0]*(2*n+1)
    for i in range(2*n+1):
        call[i] = max(0, prices[i]-K)
    step = n-1
    while step >= 0:
        for i in range(2*n-1):
            call[i] = (p_up*call[i+2]+p_mid*call[i+1]+p_down*call[i])*Rinv
        step = step - 1
```

```
    return call[0]

Ecall2 = [0]*N5
for i in range(N5):
    Ecall2[i] = f_Euro_Call_Tri2(n5[i],S0_5,K5,rf5,dt5[i], \
                                 dXu[i],dXd[i],p_up52[i],p_mid52[i],p_down52[i])
```

Plot

```
In [25]: plt.figure(5)
         ax1, = plt.plot(Ecall2, label = "Log-stock price-process (Xt)")
         ax2, = plt.plot(Ecall1, label = "stock price-process (St)")
         plt.legend(handles = [ax1,ax2])
```

```
Out[25]: <matplotlib.legend.Legend at 0x2ae4f3142b0>
```




```
In [39]: print("stock price-process converges to", Ecall1[len(Ecall1)-1])
         print("Log-stock price-process converges to", Ecall2[len(Ecall2)-1])
```

```
stock price-process converges to 3.721867597520623
Log-stock price-process converges to 3.721997935265941
```

For both methods, European call option price finally converge to about $3.72.

6 Problem 6

```
In [33]: def LDS_EuroCall(S0,K,T,r,sigma,N,b1,b2):
             H1 = f_HaltonS(b1,N)
             H2 = f_HaltonS(b2,N)

             Z_1 = [None]*N
             Z_2 = [None]*N

             #Here we used Box-Muller Method
             for i in range(N):
                 Z_1[i] = np.sqrt(-2*np.log(H1[i]))*np.cos(2*math.pi*H2[i])
                 Z_2[i] = np.sqrt(-2*np.log(H1[i]))*np.sin(2*math.pi*H2[i])
                 i = i + 1

             W = np.array(f_w(T,Z_1))
             C = np.exp(-r*T)*np.mean([max(0,S0*np.exp((r-0.5*sigma**2)*T+ \
                                                  sigma*i)-K) for i in W])
             return C
```

Using argument values given from problem 1, we will test how this function works as following:

```
In [27]: b1 = 2
         b2 = 5
         T6 = 0.5
         K6 = 30
         S0_6 = 32
         sigma6 = 0.24
         rf6 = 0.05
         N6 = 1000

         C = LDS_EuroCall(S0_6,K6,T6,rf6,sigma6,N6,b1,b2)
         print(C)

3.7168172671172104
```

This function gives a European call option price approximately $3.72, which is consistent to the result estimated in problem 1.