Li_Jiaqi_Project6

February 22, 2019

1 Problem 1

The followings are the funcntions for computing option prices:

```
In [1]: import numpy as np
        import time
        import matplotlib.pyplot as plt

        #1---------------------------------------------------
        def f_S_path(n,s0,r,dt,sigma,path):
            All = np.zeros(shape = (path,n+1))
            All[:,0] = [s0]*path
            for j in range(int(path/2)):
                Z1 = np.array(np.random.normal(0,1,n))
                Z2 = np.array([-z for z in Z1])
                dW1 = np.sqrt(dt)*Z1
                dW2 = np.sqrt(dt)*Z2
                for i in range(n):
                    x1 = np.exp((r-0.5*sigma**2)*dt+sigma*dW1[i])
                    All[2*j,i+1] = All[2*j,i]*x1
                    x2 = np.exp((r-0.5*sigma**2)*dt+sigma*dW2[i])
                    All[2*j+1,i+1] = All[2*j+1,i]*x2
            return All

        def f_FSLb_Euro(S0,r,T,K,steps,paths,sigma, option):
            dt = T/steps
            price = f_S_path(steps,S0,r,dt,sigma,paths)
            if option == "Call":
                Smax = np.zeros(paths)
                for i in range(paths):
                    Smax[i] = np.array(max(price[i,:]))
                result = np.mean(np.maximum(Smax - K, 0)*np.exp(-r*T))
            elif option == "Put":
                Smin = np.zeros(paths)
                for i in range(paths):
                    Smin[i] = np.array(min(price[i,:]))
```

```
            result = np.mean(np.maximum(K - Smin, 0)*np.exp(-r*T))
        return result
```

Followings are codes that draw plots:

```
In [3]: S01 = 98; K1 = 100; r1 = 0.03; T1 = 1; steps = 100; paths = 100000
        sigma1 = np.arange(0.12,0.52,0.04)

        Call = np.zeros(len(sigma1))
        for i in range(len(sigma1)):
            Call[i] = f_FSLb_Euro(S01,r1,T1,K1,steps,paths,sigma1[i], "Call")

        Put = np.zeros(len(sigma1))
        for i in range(len(sigma1)):
            Put[i] = f_FSLb_Euro(S01,r1,T1,K1,steps,paths,sigma1[i], "Put")

        plt.figure(1)
        plt.subplot(121)
        ax = plt.plot(sigma1,Call)
        plt.title("European Call Option Prices as function of Volatility")
        plt.xlabel("Volatility")
        plt.ylabel("Option Prices")

        plt.subplot(121)
        ax = plt.plot(sigma1,Put)
        plt.title("European Put Option Prices as function of Volatility")
        plt.xlabel("Volatility")
        plt.ylabel("Option Prices")


        ---------------------------------------------------------------------------

        KeyboardInterrupt                         Traceback (most recent call last)

        <ipython-input-3-99b44c79160c> in <module>
          8 Put = np.zeros(len(sigma1))
          9 for i in range(len(sigma1)):
    ---> 10     Put[i] = f_FSLb_Euro(S01,r1,T1,K1,steps,paths,sigma1[i], "Put")
         11
         12 plt.figure(1)


        <ipython-input-1-75751e327ef0> in f_FSLb_Euro(S0, r, T, K, steps, paths, sigma, option
         21 def f_FSLb_Euro(S0,r,T,K,steps,paths,sigma, option):
         22     dt = T/steps
    ---> 23     price = f_S_path(steps,S0,r,dt,sigma,paths)
         24     if option == "Call":
         25         Smax = np.zeros(paths)
```

```
<ipython-input-1-75751e327ef0> in f_S_path(n, s0, r, dt, sigma, path)
      8     All[:,0] = [s0]*path
      9     for j in range(int(path/2)):
---> 10         Z1 = np.array(np.random.normal(0,1,n))
     11         Z2 = np.array([-z for z in Z1])
     12         dW1 = np.sqrt(dt)*Z1


KeyboardInterrupt:
```

2 Problem 2

```
In [ ]: #2-----------------------------------------------------
        #V0 = 20000; L0 = 22000
        lambda1 = 0.2; lambda2 = 0.4; T2 = 5
        V0 = 20000; L0 = 22000

        #jump_diffusions function with default arguments
        def f_jump(V0 = 20000,L0 = 22000,lambda1 = 0.2,lambda2 = 0.4,T = 5):
            r0 = 0.02
            delta = 0.25
            alpha = 0.7
            epsilon = 0.95
            mu = -0.1
            gamma = -0.4
            sigma = 0.2
            paths = 100000
            steps = T*12

            dt = T/steps
            beta = (epsilon-alpha)/T
            R = r0 + delta*lambda2
            r = R/12
            n = T*12
            PMT = (L0*r)/(1-(1+r)**(-n))
            a = PMT/r
            b = PMT/(r*(1+r)**(n))
            c = 1+r
            t = np.arange(1/12,T+dt,dt)
            Lt = np.round(a - b*(c**(12*t)),4)
            qt = alpha + beta*t

            dt = 1/12
            Vt = np.zeros((paths,steps+1))
```

```
    Vt[:,0] = [V0]*paths

    for i in range(steps):
        Z = np.random.normal(0,1,paths)
        dWt = np.sqrt(dt)*Z
        dJt = np.random.poisson(dt*lambda1,paths)
        Vt[:,i+1] = Vt[:,i]*np.exp((mu-0.5*sigma**2)*dt+sigma*dWt)*(1+gamma*dJt)
    Vt = Vt[:,1:]

    res = np.tile(Lt*qt,paths).reshape((paths,steps))
    D = np.where(Vt-res<= 0, 1, 0)
    Q = np.argmax(D, axis = 1)*dt
    ND = np.where(np.sum(D, axis = 1) == 0)
    Q[ND] = 100

    Nt=np.clip(np.random.poisson(lambda2*dt,(paths,steps)),0,1)
    S=np.argmax(Nt,axis=1)*dt
    ND2 = np.where(np.sum(Nt, axis = 1) == 0)
    S[ND2] = 100

    count = 0
    out = np.zeros(paths)
    for i in range(paths):
        if Q[i] == 100 and S[i] == 100:
           out[i]=0
        elif Q[i] <= S[i]:
            out[i]=np.maximum((a-b*c**(12*Q[i]))-epsilon*Vt[i,int(Q[i]/dt)],0)*np.exp
            count += 1
        elif Q[i] > S[i]:
            out[i]=np.abs((a-b*c**(12*S[i]))-epsilon*Vt[i,int(S[i]/dt)])*np.exp(-r0*S
            count += 1

    tau = np.zeros(paths)
    for i in range(paths):
        tau[i] = min(min(S[i],Q[i]),T)
    i = 0
    N = paths
    while i < N:
        if tau[i] == T:
            tau = np.delete(tau,i)
            N = N - 1
        i += 1

    return [np.mean(out),count/paths, np.mean(tau)]

Payoff, DP, Etime= f_jump(V0 = V0, lambda1 = lambda1, lambda2 = lambda2, T = 5)
```

Plots:

In [ ]:
```
lambV1 = np.arange(0.05,0.45,0.05)
lambV2 = np.arange(0,0.9,0.1)
TT = np.arange(3,9,1)

#######################################################
A = np.zeros((8,6))
B = np.zeros((9,6))

for j in range(6):
    for i in range(8):
        A[i,j] = f_jump(V0 = V0, lambda1 = lambV1[i], T = TT[j])[2]

for j in range(6):
    for i in range(9):
        B[i,j] = f_jump(V0 = V0, lambda2 = lambV2[i], T = TT[j])[2]

plt.figure(2,figsize = (6,8))
plt.subplot(121)
for i in range(8):
    plt.plot(A[i,:])
plt.xlabel("Maturity Time")
plt.ylabel("Default Time")

plt.subplot(122)
for i in range(9):
    plt.plot(B[i,:])
plt.xlabel("Maturity Time")
plt.ylabel("Default Time")

#######################################################
PA = np.zeros((8,6))
PB = np.zeros((9,6))

for j in range(6):
    for i in range(8):
        PA[i,j] = f_jump(V0 = V0, lambda1 = lambV1[i], T = TT[j])[1]

for j in range(6):
    for i in range(9):
        PB[i,j] = f_jump(V0 = V0, lambda2 = lambV2[i], T = TT[j])[1]

plt.figure(3,figsize = (6,8))
plt.subplot(121)
for i in range(8):
    plt.plot(PA[i,:])
plt.xlabel("Maturity Time")
plt.ylabel("Probability")
```

```
plt.subplot(122)
for i in range(9):
    plt.plot(PB[i,:])
plt.xlabel("Maturity Time")
plt.ylabel("Probability")

#######################################################
TA = np.zeros((8,6))
TB = np.zeros((9,6))

for j in range(6):
    for i in range(8):
        TA[i,j] = f_jump(V0 = V0, lambda1 = lambV1[i], T = TT[j])[0]

for j in range(6):
    for i in range(9):
        TB[i,j] = f_jump(V0 = V0, lambda2 = lambV2[i], T = TT[j])[0]

plt.figure(4,figsize = (6,8))
plt.subplot(121)
for i in range(8):
    plt.plot(TA[i,:])
plt.xlabel("Maturity Time")
plt.ylabel("Payoff")

plt.subplot(122)
for i in range(9):
    plt.plot(TB[i,:])
plt.xlabel("Maturity Time")
plt.ylabel("Payoff")
```

3 I tried to run these code in Jupyter Notebooks so that I can plug in plots into the pdf. However, it takes forever for Jupyter Notebooks to generate the plots online. Thus, I used python on my local desktop environment to generate the plots, which only takes less than 10 minutes. The plots are included in the zipped file with corresponding names.