Li_Jiaqi_Project8

March 8, 2019

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

1 Problem 1

```
In [3]: ################################Problem 1####################################
        r0 = 0.05
        sigma = 0.18
        cap = 0.82
        Er = 0.05
        paths = 1000

        #Vasicek mothod
        def f_Vasicek(paths,steps,r0,sigma,cap,dt,Er):
            r = np.zeros((paths,steps))
            r[:,0] = r0
            for i in range(1,steps):
                dWt = np.sqrt(dt)*np.random.normal(0,1,paths)
                r[:,i] = cap*(Er-r[:,i-1])*dt+sigma*dWt + r[:,i-1]
            return r

        #---------------------------------a--------------------------------------#
        # simulate 1000 interest rate paths
        FV1 = 1000
        T = 0.5
        steps = int(366/2)+1
        dt = T/steps

        #Pricing for Pure Discount Bond
        def f_PDB(steps,paths,r0,sigma,K,Er,FV,T):
            dt = T/steps
            r = f_Vasicek(paths,steps,r0,sigma,cap,dt,Er)

            Euler = np.zeros(paths)
```

```
    for i in range(paths):
        Euler[i] = -sum(r[i,1:]*dt)

    EP = np.mean(FV*np.exp(Euler))
    return EP

PDB = f_PDB(steps,paths,r0,sigma,cap,Er,FV1,T)
print(PDB)
```

```
975.3574437544914
```

The value of the pure discount bond is about $975

```
In [10]: #----------------------------------b----------------------------------------#
         FV1 = 1000
         steps1 = 366*4
         dt1 = T/steps
         C1 = np.array([30,30,30,30,30,30,30,1030])
         T1 = np.array([0.5,1,1.5,2,2.5,3,3.5,4])

         #Pricing for Coupon Payment Bond
         def f_CPB(steps,paths,r0,sigma,K,Er,FV,T,C):
             dt = T[len(T)-1]/steps
             r = f_Vasicek(paths,steps,r0,sigma,cap,dt,Er)
             n = len(T)
             T_steps = np.array([int(i*366) for i in T])

             Euler = np.zeros((paths,n))
             for i in range(paths):
                 for j in range(n):
                     Euler[i,j] = C[j]*np.exp(-sum(r[i,:(T_steps[j]+1)]*dt))

             EP = 0
             for i in range(paths):
                 EP += sum(Euler[i,:])
             EP = EP/paths
             return EP

         CPB = f_CPB(steps1,paths,r0,sigma,cap,Er,FV1,T1,C1)
         print(CPB)
```

```
1072.775307587957
```

The value of the coupon payment bond is about $1073

```
In [14]: #----------------------------------c----------------------------------------#
         K = 980
```

```
Toption = 3/12

#Pricing option with pure discount bond as underlying asset
def f_EuroCall_PDB(FV,steps,paths,r0,sigma,cap,Er,T,Toption,K):
    dt = T/steps
    r = f_Vasicek(paths,int(Toption/dt),r0,sigma,cap,dt,Er)
    rt = r[:,int(Toption/dt)-1]
    B = 1/cap*(1-np.exp(-cap*(T-Toption)))
    A = np.exp((Er-sigma**2/(2*cap**2))*(B-(T-Toption))-sigma**2/(4*cap)*B**2)
    PDB = A*np.exp(-B*rt)*FV
    discount = np.zeros(paths)
    for i in range(paths):
        discount[i] = -sum(dt*r[i,range(int(Toption/dt)-1)])
    call = np.mean(np.exp(discount)*np.maximum(PDB-K,0))
    return call

Call_on_PDB = f_EuroCall_PDB(FV1,steps,paths,r0,sigma,cap,Er,T,Toption,K)
print(Call_on_PDB)
```

```
11.837017887652385
```

The value of the option is about $11.74

In [26]:
```
#----------------------------------d----------------------------------------#
K = 980
Toption = 3/12

#Pricing option with coupon payment bond as underlying asset
def f_EuroCall_CPB(FV,steps,paths,r0,sigma,cap,Er,T,Toption,K):
    dt = 1/366
    r = f_Vasicek(paths,steps,r0,sigma,cap,dt,Er)
    rt = r[:,int(Toption/dt)-1]
    B = 1/cap*(1-np.exp(-cap*(T-Toption)))
    A = np.exp((Er-sigma**2/(2*cap**2))*(B-(T-Toption))-sigma**2/(4*cap)*B**2)

    r_star = 0.05
    for i in range(1000):
        if sum(A*np.exp(-B*r_star)*FV) - K > 0:
            r_star = r_star + 0.0001
        if sum(A*np.exp(-B*r_star)*FV) - K < 0:
            r_star = r_star - 0.0001
    r_star = np.round(r_star,4)

    Ki = A*np.exp(-B*r_star)*FV

    CPB = np.zeros(paths)
    for i in range(paths):
```

```
            for j in range(len(T)):
                CPB[i] += np.maximum(A[j]* \
                          np.exp(-B[j]*r[:,int(Toption/dt)-1][i])*FV[j] - Ki[j],0)

        discount = np.zeros(paths)
        for i in range(paths):
            discount[i] = -sum(dt*r[i,range(int(Toption/dt)-1)])

        call = np.mean(np.exp(discount)*CPB)
        return call

    FV_C = np.array([30,30,30,30,30,30,30,1030])
    T_C = np.arange(0.5,4.5,0.5)
    Call_on_PCB = f_EuroCall_CPB(FV_C,steps,paths,r0,sigma,cap,Er,T_C,Toption,K)
    print(Call_on_PCB)
```

```
116.87664022645829
```

The value of the option is about $116.9

2 Problem 2

First construct function for simulating interest rates by using CIR method

In [33]:
```
##################################Problem 2######################################
r02 = 0.05
sigma2 = 0.18
cap2 = 0.92
Er2 = 0.055
steps2 = 366
paths2 = 1000

def f_CIR(paths,steps,r0,sigma,cap,dt,Er):
    r = np.zeros((paths,steps+1))
    r[:,0] = r0
    for i in range(steps):
        dWt = np.sqrt(dt)*np.random.normal(0,1,paths)
        r[:,i+1] = np.maximum(cap*(Er-r[:,i])*dt+sigma*np.sqrt(r[:,i])*dWt \
                              + r[:,i],0)
    return r
```

In [34]:
```
#-----------------------------------a-------------------------------------------#
# simulate 1000 interest rate paths
FV2 = 1000
T2 = 1
K2 = 980
Toption2 = 0.5
```

```
def f_EuroCall_PDB_CIR(FV,steps,paths,r0,sigma,cap,Er,T,Toption,K):
    dt = 1/366
    r_path = f_CIR(paths,int(Toption/dt),r0,sigma,cap,dt,Er)
    r = r_path[:,int(Toption/dt)-1]
    PDB = np.zeros(paths)
    for i in range(paths):
        r_T = f_CIR(paths,steps-int(Toption/dt),r[i],sigma,cap,dt,Er)
        Euler = np.zeros(paths)
        for j in range(paths):
            Euler[j] = -sum(r_T[j,1:]*dt)
        PDB[i] = np.mean(FV*np.exp(Euler))
    discount = np.zeros(paths)
    for i in range(paths):
        discount[i] = -sum(dt*r_path[i,1:])
    call = np.mean(np.exp(discount)*np.maximum(PDB-K,0))
    return call

Call_on_PDB_CIR = f_EuroCall_PDB_CIR(FV2,steps2,paths2,r02, \
                                     sigma2,cap2,Er2,T2, \
                                     Toption2,K2)
print(Call_on_PDB_CIR)
```

```
1.1373878664753918
```

The value of the option computed by Monte Carlo simulation is \$1.14.

```
In [37]: #---------------------------------b----------------------------------------#
def f_EuroCall_PDB_CIR_expicit(FV,steps,paths,r0,sigma,cap,Er,T,Toption,K):
    h1 = np.sqrt(cap**2+2*sigma**2)
    h2 = (cap+h1)/2
    h3 = 2*cap*Er/sigma**2
    B_T = (np.exp(h1*(Toption))-1)/(h2*(np.exp(h1*(Toption))-1)+h1)
    A_T = ((h1*np.exp(h2*(Toption)))/(h2*(np.exp(h1*(Toption))-1)+h1))**h3
    B_S = (np.exp(h1*(T))-1)/(h2*(np.exp(h1*(T))-1)+h1)
    A_S = ((h1*np.exp(h2*(T)))/(h2*(np.exp(h1*(T))-1)+h1))**h3
    B_TS = (np.exp(h1*(T-Toption))-1)/(h2*(np.exp(h1*(T-Toption))-1)+h1)
    A_TS = ((h1*np.exp(h2*(T-Toption)))/(h2*(np.exp(h1*(T-Toption))-1)+h1))**h3

    PDB_T = A_T*np.exp(-B_T*r0)
    PDB_S = A_S*np.exp(-B_S*r0)

    theta = np.sqrt(cap**2+2*sigma**2)
    phi = 2*theta/(sigma**2*(np.exp(theta*Toption)-1))
    yucha = (cap+theta)/sigma**2
    r_star = np.log(A_TS/(K/FV))/B_TS
```

```
    x1 = 2*r_star*(phi+yucha+B_TS)
    p1 = 4*cap*Er/(sigma**2)
    q1 = (2*phi**2*r0*np.exp(theta*Toption))/(phi+yucha+B_TS)
    x2 = 2*r_star*(phi+yucha)
    p2 = 4*cap*Er/(sigma**2)
    q2 = (2*phi**2*r0*np.exp(theta*Toption))/(phi+yucha)

    call = FV*PDB_S*si.ncx2.cdf(x1,p1,q1)-K*PDB_T*si.ncx2.cdf(x2,p2,q2)

    return np.mean(call)

Call_on_PDB_CIR_explicit = f_EuroCall_PDB_CIR_expicit(FV2, steps2,paths2, \
                                                      r02,sigma2,cap2,Er2, \
                                                      T2,Toption2,K2)
print(Call_on_PDB_CIR_explicit)
```

```
1.1234212323333281
```

The value of the option computed by Explicit formula is $1.12.

Compare with the result generated by Monte Carlo Simulatoin, the value of the option generated by Explicit formula is slightly smaller.

3 Problem 3

In [38]:
```
#################################Problem 3####################################
x0 = 0
y0 = 0
phi = 0.03
r0 = 0.03
a = 0.1
b = 0.3
sigma = 0.03
ita = 0.08
rho = 0.7
S = 1
T = 0.5
paths = 1000
FV = 1000
K = 985

#Generate correlated brownian motions
def f_corr2W(n,var,rho):
    covM = np.array([[1,rho],[rho,1]])
    L = np.linalg.cholesky(covM)
    N1 = np.random.normal(0,1,n)
    N2 = np.random.normal(0,1,n)
    dWt1 = np.sqrt(var[0])*L[0,0]*N1
```

```python
    dWt2 = np.sqrt(var[1])*(L[1,0]*N1 + L[1,1]*N2)
    r = [dWt1,dWt2]
    return r

#Construct function for simulating interest rates by using G2++ method
def f_Gpp(steps,paths,x0,y0,r0,a,b,sigma,ita,phi,dt):
    var = [dt,dt]
    x = np.zeros((paths,steps+1))
    y = np.zeros((paths,steps+1))
    r = np.zeros((paths,steps+1))
    x[:,0] = x0
    y[:,0] = y0
    for i in range(1,steps+1):
        dWt = f_corr2W(paths,var,rho)
        x[:,i] = x[:,i-1]-a*x[:,i-1]*dt+sigma*dWt[0]
        y[:,i] = y[:,i-1]-b*y[:,i-1]*dt+ita*dWt[1]
    r = np.maximum(x+y+phi,0)
    return r,x[:,steps],y[:,steps]

def f_EuroP_PDB_Gpp(FV,paths,x0,y0,r0,a,b,sigma,ita,phi,T,S,K):
    dt = 1/366
    steps = int(S/dt)
    r_path,x,y = f_Gpp(int(T/dt),paths,x0,y0,r0,a,b,sigma,ita,phi,dt)
    r = r_path[:,int(T/dt)-1]
    PDB = np.zeros(paths)
    for i in range(paths):
        r_T = f_Gpp(steps-int(T/dt),paths,x[i],y[i],r[i],a,b, \
                    sigma,ita,phi,dt)[0]
        Euler = np.zeros(paths)
        for j in range(paths):
            Euler[j] = -sum(r_T[j,:]*dt)
        PDB[i] = np.mean(FV*np.exp(Euler))
    discount = np.zeros(paths)
    for i in range(paths):
        discount[i] = -sum(dt*r_path[i,:])
    put = np.mean(np.exp(discount)*np.maximum(K-PDB,0))
    return put

Put_on_PDB_Gpp = f_EuroP_PDB_Gpp(FV,paths,x0,y0,r0,a,b,sigma,ita,phi,T,S,K)
print(Put_on_PDB_Gpp)
```

```
13.608389168896446
```

The value of the European Put option computed by Monte Carlo Simulation is about $13.6