finm320_24_hw3

April 19, 2024

0.1 Problem 1

0.1.1 (a)

The short rate (the instantaneous spot rate of interest) follows the process

$dr_t = \alpha(r_t, t)dt + \beta(r_t, t)dW_t$.

Applying Ito's rule, we can get:

$$dC_t = \frac{\partial C}{\partial t}dt + \frac{\partial C}{\partial r_t}dr_t + \frac{1}{2}\frac{\partial^2 C}{\partial r_t^2}(dr_t)^2$$

$$= \frac{\partial C}{\partial t}dt + \frac{\partial C}{\partial r_t}(\alpha(r_t, t)dt + \beta(r_t, t)dW_t) + \frac{1}{2}\frac{\partial^2 C}{\partial r_t^2}(\beta^2(r_t, t)dt)$$

$$= (\frac{\partial C}{\partial t} + \frac{\partial C}{\partial r_t}\alpha(r_t, t) + \frac{1}{2}\frac{\partial^2 C}{\partial r_t^2}\beta^2(r_t, t))dt + \frac{\partial C}{\partial r_t}\beta(r_t, t)dW_t$$

Setting drift equal to rC, we obtain the PDE:

$$\frac{\partial C}{\partial t} + \frac{\partial C}{\partial r_t}\alpha(r_t, t) + \frac{1}{2}\frac{\partial^2 C}{\partial r_t^2}\beta^2(r_t, t) = rC$$

$\alpha(r_t, t) = \kappa(\theta - r_t), \beta(r_t, t) = \sigma$. We have

$$C_n^j = \frac{1}{1 + r_t\Delta t}(q_u C_{n+1}^{j+1} + q_m C_{n+1}^j + q_d C_{n+1}^{j-1})$$

where

$$q_u = \frac{1}{2}[\frac{\sigma^2\Delta t}{(\Delta r_t)^2} + \frac{\nu\Delta t}{\Delta r_t}]$$

$$q_m = 1 - \frac{\sigma^2\Delta t}{(\Delta r_t)^2}$$

$$q_d = \frac{1}{2}[\frac{\sigma^2\Delta t}{(\Delta r_t)^2} - \frac{\nu\Delta t}{\Delta r_t}]$$

[1]:
```
import numpy as np
```

[2]:
```
class Vasicek:

    def __init__(self,kappa,theta,sigma):
        self.kappa=kappa
        self.theta=theta
        self.sigma=sigma
```

[3]:
```
hw3dynamics = Vasicek(kappa=3,theta=0.05,sigma=0.03)
```

[4]:
```
class Bond:

    def __init__(self, T):
        self.T=T
```

[5]:
```
hw3contract = Bond(T=5)
```

[6]:
```
class FDexplicitEngine:

    def __init__(self, rMax, rMin, deltar, deltat, useUpwind):
        self.rMax=rMax
        self.rMin=rMin
        self.deltar=deltar
        self.deltat=deltat
        self.useUpwind=useUpwind

    def price_bond_vasicek(self,contract,dynamics):
    # You complete the coding of this function
    #
    # Returns array of all initial short rates,
    # and the corresponding array of zero-coupon
    # T-maturity bond prices

        T = contract.T
        N=round(T/self.deltat)
        if abs(N-T/self.deltat) > 1e-12:
            raise ValueError("Bad delta t")

        r=np.arange(self.rMax,self.rMin-self.deltar/2,-self.deltar)    #I'm␣
 ↪making the FIRST indices of the array correspond to HIGH levels of r
        bondprice=np.ones(np.size(r))

        nu = dynamics.kappa*(dynamics.theta - r)
        sigma = dynamics.sigma
        a = sigma**2 * self.deltat / (self.deltar**2)
        b = nu * self.deltat / self.deltar
        if self.useUpwind:
            qu = (1/2)*(a+b+abs(b))
            qd = (1/2)*(a-b+abs(b))
            qm = 1-qu-qd
        else:
            qu= (1/2)*(a+b)
            qd= (1/2)*(a-b)
            qm= 1-a

        for t in np.arange(N-1,-1,-1)*self.deltat:
            bondprice=1/(1+r*self.deltat)*(qd*np.
↪roll(bondprice,-1)+qm*bondprice+qu*np.roll(bondprice,1))

            # It is not obvious in this case,
            # what boundary conditions to use at the top and bottom
            # so let us assume "linearity" boundary conditions
            bondprice[0]=2*bondprice[1]-bondprice[2]
            bondprice[-1]=2*bondprice[-2]-bondprice[-3]

        return (r, bondprice)
```

[7]:
```
hw3FD = FDexplicitEngine(rMax=0.35,rMin=-0.25,deltar=0.01,deltat=0.
↪01,useUpwind=False)
```

[8]:
```
(r1, bondprice1) = hw3FD.price_bond_vasicek(hw3contract,hw3dynamics)
```

[9]:
```
np.set_printoptions(precision=4,suppress=True)
displayrows=(r1<0.15+hw3FD.deltar/2) & (r1>0.0-hw3FD.deltar/2)
```

[10]:
```
print(np.stack((r1, bondprice1),axis=1)[displayrows])
```

```
[[ 1.5000e-01 -1.4273e+09]
 [ 1.4000e-01  1.6361e+08]
 [ 1.3000e-01  2.2294e+07]
 [ 1.2000e-01 -1.3724e+06]
 [ 1.1000e-01 -1.3361e+05]
 [ 1.0000e-01  3.2966e+03]
 [ 9.0000e-02  1.3021e+02]
 [ 8.0000e-02  7.7128e-01]
 [ 7.0000e-02  7.7385e-01]
 [ 6.0000e-02  7.7643e-01]
 [ 5.0000e-02  7.7902e-01]
 [ 4.0000e-02  7.8162e-01]
 [ 3.0000e-02  7.8423e-01]
 [ 2.0000e-02  7.8685e-01]
 [ 1.0000e-02  1.4165e+03]
 [-3.3307e-16  5.1498e+04]]
```

0.2 (c)

```
[11]: hw3FD2 = FDexplicitEngine(rMax=0.35,rMin=-0.25,deltar=0.01,deltat=0.
      ↪01,useUpwind=True)
```

```
[12]: (r2, bondprice2) = hw3FD2.price_bond_vasicek(hw3contract,hw3dynamics)
```

```
[13]: np.set_printoptions(precision=4,suppress=True)
      displayrows=(r2<0.15+hw3FD2.deltar/2) & (r2>0.0-hw3FD2.deltar/2)
```

```
[14]: print(np.stack((r2, bondprice2),axis=1)[displayrows])
```

```
[[ 0.15     0.7536]
 [ 0.14     0.7561]
 [ 0.13     0.7586]
 [ 0.12     0.7611]
 [ 0.11     0.7637]
 [ 0.1      0.7662]
 [ 0.09     0.7688]
 [ 0.08     0.7713]
 [ 0.07     0.7739]
 [ 0.06     0.7765]
 [ 0.05     0.7791]
 [ 0.04     0.7817]
 [ 0.03     0.7843]
 [ 0.02     0.7869]
 [ 0.01     0.7895]
 [-0.       0.7922]]
```

0.3 (d)

By Taylor's theorem, we have:

$f(x+h) = f(x) + hf'(x) + O(h^2)$

$f(x+h) - f(x) - hf'(x) = O(h^2)$

$\left|\frac{f(x+h)-f(x)}{h} - f'(x)\right| = \left|\frac{O(h^2)}{h}\right| = O(h)$

By Taylor's theorem, we have:

$f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2f''(x) + O(h^3)$

$f(x-h) = f(x) - hf'(x) + \frac{1}{2}h^2f''(x) + O(h^3)$

By subtracting these two functions we can get:

$f(x+h) - f(x-h) = 2hf'(x) + O(h^3)$

$f(x+h) - f(x-h) - 2hf'(x) = O(h^3)$

$\left|\frac{f(x+h)-f(x-h)}{2h} - f'(x)\right| = \left|\frac{O(h^3)}{2h}\right| = O(h^2)$

0.4 (e)

[15]:
```python
# dr = 0.01, dt = 0.01, rmax = 0.35, rmin = -0.25
# find where r is 0.10
displayrows=(abs(r1-0.10)<1e-8)

# central difference
print(np.stack((r1, bondprice1),axis=1)[displayrows])
# upwind calculation
print(np.stack((r2, bondprice2),axis=1)[displayrows])
```

```
[[   0.1     3296.5929]]
[[0.1    0.7662]]
```

The upwind calculation is deemed to be more accurate.

From the outputs displayed above, it is evident that the central difference method yields a highly unusual result 3296.5929, which appears to be implausible. In contrast, the upwind calculation produces a similar value 0.7662 to the exact bond price 0.7661.

0.5 (f)

"less"

"greater"

0.6 (g)

[16]:
```python
T = 5
displayrows = (abs(r1 - 0.12) < 1e-8)
bondprice_12 = bondprice2[displayrows][0]
print('The price of the bond when r0 equals 0.12:', bondprice_12)
ytm_12 = np.log(1/bondprice_12) / T
print('Yield to maturity (YTM) of the bond for r0 at 0.12:', ytm_12)
```

```
The price of the bond when r0 equals 0.12: 0.761143773472767
Yield to maturity (YTM) of the bond for r0 at 0.12: 0.05458660238621129
```

[17]:
```python
T = 5
displayrows = (abs(r1 - 0.02) < 1e-8)
bondprice_02 = bondprice2[displayrows][0]
print('The price of the bond when r0 equals 0.02:', bondprice_02)
ytm_02 = np.log(1/bondprice_02) / T
print('Yield to maturity (YTM) of the bond for r0 at 0.02:', ytm_02)
```

```
The price of the bond when r0 equals 0.02: 0.7869156415550577
Yield to maturity (YTM) of the bond for r0 at 0.02: 0.04792684524038049
```

Understanding yield to maturity (YTM) as the anticipated average of the instantaneous spot rates from the present until time T can help explain its behavior. A lower initial spot rate, represented as r, usually shifts upward to align with the long-term average as time progresses. On the other hand, a higher initial spot rate tends to gravitate downwards towards this average.

In the context of the Vasicek model $dr_t = 3(0.05 - r_t)dt + 0.03dW_t$, where the average level is 0.05, the starting point of r at time 0 influences its subsequent movement. For instance, if $r_0 = 0.02$, the drift is positive, indicating that r is likely to rise over time, leading to a YTM higher than 0.02. Conversely, if $r_0 = 0.12$, the drift is negative, suggesting that r is likely to decline over time, resulting in a YTM lower than 0.12. do not change the meaning and rephrase it

1 Problem 2

1.1 (a)

B-S call option (time-0)

$$\Delta = N(d1)$$

where $d1 = \frac{log(F/K)}{\sigma\sqrt{T}} + \frac{\sigma\sqrt{T}}{2}, F = S_0 e^{r(T)}$.

Therefore, we have

$$d1 = N^{-1}(\Delta) = \frac{log(S_0 e^{r(T)}/K)}{\sigma\sqrt{T}} + \frac{\sigma\sqrt{T}}{2}$$

$$K = S_0 e^{-(r+\frac{1}{2}\sigma^2)T + \sigma\sqrt{T}N^{-1}(\Delta)}$$

[18]:
```python
from scipy.stats import norm
from scipy.optimize import brentq
import numpy as np

# Define the Black-Scholes function to calculate d1, d2 and call option price
def black_scholes(S0, K, T, r, sigma):
    d1 = (np.log(S0 / K) + (r + 0.5 * sigma**2) * T) / (sigma * np.sqrt(T))
    d2 = d1 - sigma * np.sqrt(T)
    return d1, d2

# Part (a): Function to find the strike price K for a given delta Δ
def find_strike(S0, delta, T, r, sigma):
    def f(K):
        d1, _ = black_scholes(S0, K, T, r, sigma)
        return norm.cdf(d1) - delta
    # Use brentq root finding method to solve for K
    return brentq(f, S0 / 2, S0 * 2)
```

```
/opt/anaconda3/lib/python3.8/site-packages/scipy/__init__.py:146: UserWarning: A
NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy
(detected version 1.24.4
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

1.2 (b)

[19]:
```python
# Define parameters given in the problem
S0 = 300  # Initial stock price
T = 1/12  # Time to expiry in years
sigma = 0.4  # Volatility
r = 0.01  # Interest rate

# Part (b): Find strike prices for 25-delta and 75-delta calls
delta_25 = 0.25
delta_75 = 0.75

K_25_delta = find_strike(S0, delta_25, T, r, sigma)
K_75_delta = find_strike(S0, delta_75, T, r, sigma)

# Calculate the premiums for these options
def option_premium(S0, K, T, r, sigma):
    d1, d2 = black_scholes(S0, K, T, r, sigma)
    C = S0 * norm.cdf(d1) - K * np.exp(-r * T) * norm.cdf(d2)
    return C

premium_25_delta = option_premium(S0, K_25_delta, T, r, sigma)
premium_75_delta = option_premium(S0, K_75_delta, T, r, sigma)
print("The strike of a 25-delta call is: " + str(round(K_25_delta,3)))
print("The strike of a 75-delta call is: " + str(round(K_75_delta,3)))
print("The premium of a 25-delta call is: " + str(round(premium_25_delta,3)))
print("The premium of a 75-delta call is: " + str(round(premium_75_delta,3)))
```

```
The strike of a 25-delta call is: 326.74
The strike of a 75-delta call is: 279.611
The premium of a 25-delta call is: 4.883
The premium of a 75-delta call is: 26.104
```

1.3 (c)

The 25-delta option gives you more leverage compared to the 75-delta option. Leverage here is understood as the ratio of the percentage change in the option price to the percentage change in the underlying asset price. The higher the leverage, the more sensitive the option price is to changes in the underlying asset price.

[20]:
```python
# Part (c): Calculate the lambdas of the two options
lambda_25_delta = delta_25 * S0 / premium_25_delta
lambda_75_delta = delta_75 * S0 / premium_75_delta

print("The leverage (lambda) for the 25-delta option is: " +␣
 ↪str(round(lambda_25_delta,3)))
print("The leverage (lambda) for the 25-delta option is: " +␣
 ↪str(round(lambda_75_delta,3)))
```

```
The leverage (lambda) for the 25-delta option is: 15.361
The leverage (lambda) for the 25-delta option is: 8.62
```