zym20000325 / FINM32000 Private

Code Issues Pull requests Actions Projects Security Insights Se

Beta Try the new code view

master

FINM32000 / HW3 / Yumeng_Zhang_hw3.ipynb

Yumeng Zhang hw 1-3 History

0 contributors

2.01 MB

FINM 32000 - HW3

Yumeng Zhang (12372205)

I discussed Problem 1 and 2 with Yitong Li and Qianyu Pan.

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

Problem 1

(a)

According to Itô's rule, $df(t, X_t) = \frac{\partial f}{\partial t} dt + \frac{\partial f}{\partial X_t} dX_t + \frac{1}{2} \frac{\partial^2 f}{\partial X_t^2} d\langle X\rangle_t$

$f = C_t = C(r_t, t)$ $\quad \because dr_t = \alpha(r_t, t)\, dt + \beta(r_t, t)\, dW_t \quad \therefore d\langle r\rangle_t = \beta^2(r_t, t)\, dt$

$$\begin{aligned} \therefore dC_t &= \frac{\partial C}{\partial t} dt + \frac{\partial C}{\partial r_t} dr_t + \frac{1}{2} \frac{\partial^2 C}{\partial r_t^2} \beta^2(r_t, t)\, dt \\ &= \frac{\partial C}{\partial t} dt + \frac{\partial C}{\partial r_t} \alpha(r_t, t)\, dt + \frac{\partial C}{\partial r_t} \beta(r_t, t)\, dW_t + \frac{1}{2} \frac{\partial^2 C}{\partial r_t^2} \beta^2(r_t, t)\, dt \\ &= \left(\frac{\partial C}{\partial t} + \frac{\partial C}{\partial r_t} \alpha(r_t, t) + \frac{1}{2} \frac{\partial^2 C}{\partial r_t^2} \beta^2(r_t, t) \right) dt + \frac{\partial C}{\partial r_t} \beta(r_t, t)\, dW_t \end{aligned}$$

$\because$ drift $= rC$ $\quad \therefore$ PDE is $\frac{\partial C}{\partial t} + \frac{\partial C}{\partial r_t} \alpha(r_t, t) + \frac{1}{2} \frac{\partial^2 C}{\partial r_t^2} \beta^2(r_t, t) = rC(r_t, t) \quad C(r_T, T) = F(r_T)$

(b)

Apply a standard central-difference explicit finite difference scheme.

$\because dr_t = \kappa(\theta - r_t)\, dt + \sigma W_t \quad \therefore \alpha(r_t, t) = \kappa(\theta - r_t),\ \beta(r_t, t) = \sigma$

plug in PDE $\quad \frac{\partial C}{\partial t} + \frac{\partial C}{\partial r_t} \kappa(\theta - r_t) + \frac{1}{2} \frac{\partial^2 C}{\partial r_t^2} \sigma^2 = rC(r_t, t)$

according to lecture notes, $\quad \frac{\partial C}{\partial t} + \nu \frac{\partial C}{\partial r} + \frac{1}{2} \sigma^2 \frac{\partial^2 C}{\partial r^2} = rC$

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

$q_u = \frac{1}{2} \left[\frac{\sigma^2 \Delta t}{(\Delta r)^2} + \frac{\nu \Delta t}{\Delta r} \right] \quad q_m = 1 - \frac{\sigma^2 \Delta t}{(\Delta r)^2} \quad q_d = \frac{1}{2} \left[\frac{\sigma^2 \Delta t}{(\Delta r)^2} - \frac{\nu \Delta t}{\Delta r} \right]$ $\quad$ here $\nu = \kappa(\theta - r_t)$

let $\quad k_1 = \frac{\nu \Delta t}{\Delta r} \quad k_2 = \frac{\sigma^2 \Delta t}{(\Delta r)^2}$

In [17]:
```
class Vasicek:

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

In [18]:
```
hw3dynamics = Vasicek(kappa=3,theta=0.05,sigma=0.03)
```

In [19]:
```
class Bond:
```

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

In [20]:

```
hw3contract = Bond(T=5)
```

In [153…

```
class FDexplicit:

    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 mak
        bondprice=np.ones(np.size(r))

        k1 = (dynamics.kappa * (dynamics.theta - r) * self.deltat)/(self.delta
        k2 = (dynamics.sigma ** 2 * self.deltat) / self.deltar ** 2

        if self.useUpwind:

            qu = 0.5*(k2 + np.maximum(2*k1,0))    #fill this in with an array.
            qd = 0.5*(k2 - np.minimum(2*k1,0))    #fill this in with an array.
            qm = 1-qu-qd                          #fill this in with an array.

        else:

            qu = 0.5*(k2+k1)        #fill this in with an array.
            qd = 0.5*(k2-k1)        #fill this in with an array.
            qm = 1-k2               #fill this in with an array.

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

            # 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)
```

In [154…

```
hw3FD = FDexplicit(rMax=0.35,rMin=-0.25,deltar=0.01,deltat=0.01,useUpwind=Fals
```

In [155…

```
(r, bondprice) = hw3FD.price_bond_vasicek(hw3contract,hw3dynamics)
```

In [156…

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

In [157…

```
central_difference = np.stack((r, bondprice),1)[displayrows]
print(central_difference)
```

```
[[ 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]]
```

(c)

Use an explicit upwind approximation instead of the usual central difference.

If $\kappa(\theta - r_j) \geq 0$, $\quad \frac{\partial c}{\partial r}(r_j, t_{n+1}) \approx \frac{C_{n+1}^{j+1} - C_{n+1}^{j}}{\Delta r}$

$\Rightarrow \quad q_u = \frac{1}{2}\left[\frac{\sigma^2 \Delta t}{(\Delta r)^2} + 2\frac{v\Delta t}{\Delta r}\right] \qquad q_m = 1 - \frac{\sigma^2 \Delta t}{(\Delta r)^2} - \frac{v\Delta t}{\Delta r} \qquad q_d = \frac{\sigma^2 \Delta t}{2(\Delta r)^2}$

If $\kappa(\theta - r_j) < 0$, $\quad \frac{\partial c}{\partial r}(r_j, t_{n+1}) \approx \frac{C_{n+1}^{j} - C_{n+1}^{j-1}}{\Delta r}$

$\Rightarrow \quad q_u = \frac{\sigma^2 \Delta t}{2(\Delta r)^2} \qquad q_m = 1 - \frac{\sigma^2 \Delta t}{(\Delta r)^2} + \frac{v\Delta t}{\Delta r} \qquad q_d = \frac{1}{2}\left[\frac{\sigma^2 \Delta t}{(\Delta r)^2} - 2\frac{v\Delta t}{\Delta r}\right]$

In [158…

```
hw3FD2 = FDexplicit(rMax=0.35,rMin=-0.25,deltar=0.01,deltat=0.01,useUpwind=Tru

(r, bondprice) = hw3FD2.price_bond_vasicek(hw3contract,hw3dynamics)

np.set_printoptions(precision=4,suppress=True)
displayrows=(r<0.15+hw3FD2.deltar/2) & (r>0.0-hw3FD2.deltar/2)

upwind = np.stack((r, bondprice),1)[displayrows]
print(upwind)
```

```
[[ 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]]
```

(d)

Apply Taylor's theorem to f(x+h) and f(x-h) :

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

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

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

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

(e)

In [159…

```
r0 = 0.1
```

Central-difference calculation of the bond price for r0 = 0.10.

In [176…

```
index = np.where(abs(central_difference[:,0] - r0) <= 1e-8)[0][0]
central_difference[index][1]
```

Out[176…

```
3296.5929237489718
```

Upwind calculation of the bond price for r0 = 0.10.

In [177…

```
index = np.where(abs(upwind[:,0] - r0) <= 1e-8)[0][0]
upwind[index][1]
```

Out[177…

```
0.7662252882450523
```

Upwind calculation of the bond price is more accurate.

(f)

Ignoring stability issues and considering only consistency (i.e. “truncation error,” also known as “local discretization error”), the upwind explicit scheme, which uses one-sided spatial differences, discretizes the PDE with **less** accuracy than the standard explicit scheme, which uses central spatial differences.

However, to actually guarantee convergence, the grid spacing must satisfy certain stability constraints. In a PDE exhibiting strong drift, we have just seen that these constraints may allow the upwind scheme **greater** freedom in choosing grid spacing, compared to the standard scheme.

(g)

In [180…

```
P_T = 1
T = 5
t = 0
```

In [185…

```
r0 = 0.12
index = np.where(abs(upwind[:,0] - r0) <= 1e-8)[0][0]
P_t = upwind[index][1]

yield_ = np.log(abs(P_T/P_t))/(T-t)

print("The yield-to-maturity of a 5-year discount bond in the case that r0 = 0
```

```
The yield-to-maturity of a 5-year discount bond in the case that r0 = 0.12 is 0.
0546.
```

In [186…

```
r0 = 0.02
index = np.where(abs(upwind[:,0] - r0) <= 1e-8)[0][0]
P_t = upwind[index][1]

yield_ = np.log(abs(P_T/P_t))/(T-t)

print("The yield-to-maturity of a 5-year discount bond in the case that r0 = 0
```

```
The yield-to-maturity of a 5-year discount bond in the case that r0 = 0.02 is 0.
0479.
```

The reason why intuitively the yield for r0 = 0.12 is smaller than 0.12, whereas the yield for r0 = 0.02 is greater than 0.02 is that θ = 0.05. When interest rates are above their long-term average, they are likely to fall back towards the average, and when they are below their long-term average, they are likely to rise back towards the average.

As a result, when r0 = 0.12 > θ, the yield for r0 = 0.12 will be smaller than 0.12, reflecting the expectation that interest rates will decrease and bond prices will increase over time. Conversely, the yield for r0 = 0.02 < θ will be greater than 0.02.

Problem 2

(a)

According to L2.13, $C(K,T) = C^{BS}(\bar{\sigma}_T)$, $\sigma_{imp} = \bar{\sigma}_T = \sqrt{\frac{1}{T}\int_0^T \sigma^2(t)dt}$

The function is with respect to T, but not K. As a result, the dynamics are capable of generating a non-constant term-structure of implied volatility, but not capable of generating an implied volatility skew.

(b)

In [195…

```
from scipy.stats import norm
from scipy.optimize import bisect, brentq

class GBMdynamics:
```

```
    def __init__(self, S, r, rGrow, sigma=None):

        self.S = S
        self.r = r
        self.rGrow = rGrow
        self.sigma = sigma

    def update_sigma(self, sigma):

        self.sigma = sigma
        return self

class CallOption:

    def __init__(self, K, T, price=None):

        self.K = K
        self.T = T
        self.price = price

    def BSprice(self, dynamics):

        F = dynamics.S*np.exp(dynamics.rGrow*self.T)
        sd = dynamics.sigma*np.sqrt(self.T)
        d1 = np.log(F/self.K)/sd+sd/2
        d2 = d1-sd

        return np.exp(-dynamics.r*self.T)*(F*norm.cdf(d1)-self.K*norm.cdf(d2))

    def BSprice_f(self, S, K, T, r,rGrow, sigma):

        F = S*np.exp(rGrow*T)
        sd = sigma*np.sqrt(T)
        d1 = np.log(F/K)/sd+sd/2
        d2 = d1-sd

        return np.exp(-r*T)*(F*norm.cdf(d1)-K*norm.cdf(d2))

    def IV(self, dynamics):

        if self.price is None:
            raise ValueError('Contract price must be given')

        df = np.exp(-dynamics.r*self.T)  #discount factor
        F = dynamics.S / df
        lowerbound = np.max([0,(F-self.K)*df])

        C = self.price

        if C<lowerbound:
            return np.nan
        if C==lowerbound:
            return 0
        if C>=F*df:
            return np.nan

        dytry = dynamics
        dytry.sigma = 0.2

        while self.BSprice(dytry)>C:
            dytry.sigma /= 2
        while self.BSprice(dytry)<C:
            dytry.sigma *= 2
```

```
        hi = dytry.sigma
        lo = hi/2

        impliedVolatility = bisect(lambda x: self.BSprice_f(dynamics.S, self.K

        return impliedVolatility
```

In [205…

```
contract1 = CallOption(K = 100, T = 0.1, price = 5.25)
dynamics1 = GBMdynamics(S = 100, r = 0.05, rGrow = 0.05)
vol1 = contract1.IV(dynamics1)

print("Black-Scholes implied volatility of the 0.1-expiry call with 5.25 time-
```

```
Black-Scholes implied volatility of the 0.1-expiry call with 5.25 time-0 price i
s 0.3973204.
```

In [206…

```
contract2 = CallOption(K = 100, T = 0.2, price = 7.25)
dynamics2 = GBMdynamics(S = 100, r = 0.05, rGrow = 0.05)
vol2 = contract2.IV(dynamics2)

print("Black-Scholes implied volatility of the 0.2-expiry call with 7.25 time-
```

```
Black-Scholes implied volatility of the 0.2-expiry call with 7.25 time-0 price i
s 0.3801713.
```

In [207…

```
contract3 = CallOption(K = 100, T = 0.5, price = 9.5)
dynamics3 = GBMdynamics(S = 100, r = 0.05, rGrow = 0.05)
vol3 = contract3.IV(dynamics3)

print("Black-Scholes implied volatility of the 0.5-expiry call with 9.5 time-0
```

```
Black-Scholes implied volatility of the 0.5-expiry call with 9.5 time-0 price is
0.2950973.
```

Let σ_1, σ_2, σ_3 be the B-S implied volatilities of the three options

If $0 \le t \le 0.1$ $\quad \sigma_t = \sqrt{\sigma_1^2} = \sigma_1$

If $0.1 < t \le 0.2$ $\quad \sigma_t = \sqrt{2\sigma_2^2 - \sigma_1^2}$

If $0.2 < t \le 0.5$ $\quad \sigma_t = \sqrt{\frac{1}{3}(5\sigma_3^2 - 2\sigma_2^2)}$

$\therefore$ the time-varying local volatility function is

$$\sigma_t = \begin{cases} \sigma_1, & 0 \le t \le 0.1 \\ \sqrt{2\sigma_2^2 - \sigma_1^2}, & 0.1 < t \le 0.2 \\ \sqrt{\frac{1}{3}(5\sigma_3^2 - 2\sigma_2^2)}, & 0.2 < t \le 0.5 \end{cases}$$

(c)

$\bar{\sigma}_T = \sqrt{\frac{1}{T}\int_0^T \sigma^2(t)\,dt}$ when $T = 0.4$ $\quad \bar{\sigma}_{0.4} = \sqrt{\frac{1}{0.4}\int_0^{0.4} \sigma^2(t)\,dt}$

$$= \sqrt{\frac{1}{0.4}\left(0.1\sigma_1^2 + 0.1(2\sigma_2^2 - \sigma_1^2) + 0.2 \times \frac{1}{3}(5\sigma_3^2 - 2\sigma_2^2)\right)} = \sqrt{\frac{1}{6}\sigma_2^2 + \frac{5}{6}\sigma_3^2}$$

In [212…

```
vol4 = np.sqrt((1/6)*vol2**2 + (5/6)*vol3**2)

contract4 = CallOption(K = 100, T = 0.4)
dynamics4 = GBMdynamics(S = 100, r = 0.05, rGrow = 0.05,sigma = vol4)

price4 = contract4.BSprice(dynamics4)
```

```
print("The time-0 price of an at-the-money European call with expiry 0.4 is "
```

```
The time-0 price of an at-the-money European call with expiry 0.4 is 8.7842018.
```