Ki_Hyun_12125881_FINM 32000_HW4

April 28, 2023

FINM 32000 - Numerical Methods Spring 2023

Homework 4

Due - 23:59 [CST] April 28th, 2023

Ki Hyun

Student ID: 12125881

0.0.1 Imports

[1]:
```
import numpy as np
from scipy.sparse import diags
from scipy.sparse.linalg import spsolve
from scipy.stats import norm
```

0.0.2 Helper Functions

[2]:
```
class CEV:

    def __init__(self,volcoeff,alpha,rGrow,r,S0):
        self.volcoeff = volcoeff
        self.alpha = alpha
        self.rGrow = rGrow
        self.r = r
        self.S0 = S0
```

[3]:
```
class Put:

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

[4]:
```
class FD_CrankNicolson:

    def __init__(self,SMax,SMin,deltaS,deltat):
        self.SMax=SMax
        self.SMin=SMin
```

```
        self.deltaS=deltaS
        self.deltat=deltat


    #You complete the coding of this function:

    def price_put_CEV(self,contract,dynamics):
    # returns array of all initial spots,
    # and the corresponding array of put prices

        alpha, r, rGrow, volcoeff = dynamics.alpha, dynamics.r, dynamics.rGrow,␣
↪dynamics.volcoeff

        # SMin and SMax denote the smallest and largest S in the _interior_.
        # The boundary conditions are imposed one level _beyond_,
        # e.g. at S_lowboundary=SMin-deltaS, not at SMin.
        # To relate to lecture notation, S_lowboundary is S_{-J}
        # whereas SMin is S_{-J+1}

        N=round(contract.T/self.deltat)
        if abs(N-contract.T/self.deltat)>1e-12:
            raise ValueError('Bad time step')
        numS=round((self.SMax-self.SMin)/self.deltaS)+1
        if abs(numS-(self.SMax-self.SMin)/self.deltaS-1)>1e-12:
            raise ValueError('Bad time step')
        S=np.linspace(self.SMax,self.SMin,numS)    #The FIRST indices in this␣
↪array are for HIGH levels of S
        S_lowboundary=self.SMin-self.deltaS

        putprice = np.maximum(contract.K-S,0)

        ratio1 = self.deltat/self.deltaS
        ratio2 = self.deltat/self.deltaS**2
        f =  (volcoeff**2 * S**(2 + 2*alpha))/2  # You fill in with an array of␣
↪the same size as S.
        g =  rGrow * S  # You fill in with an array of the same size as S.
        h =  np.full(numS, -r) # You fill in with an array of the same size as␣
↪S (or a scalar is acceptable here)
        F = 0.5*ratio2 * f + 0.25*ratio1 * g
        G =     ratio2 * f - 0.50*self.deltat * h
        H = 0.5*ratio2 * f - 0.25*ratio1 * g

        #Right-hand-side matrix
        RHSmatrix = diags([H[:-1], 1-G, F[1:]], [1,0,-1], shape=(numS,numS),␣
↪format="csr")

        #Left-hand-side matrix
```

```
        LHSmatrix = diags([-H[:-1], 1+G, -F[1:]], [1,0,-1], shape=(numS,numS), format="csr")
        # diags creates SPARSE matrices

        for t in np.arange(N-1,-1,-1)*self.deltat:

            rhs = RHSmatrix * putprice

            #Now let's add the boundary condition vectors.
            #They are nonzero only in the last component:
            rhs[-1] = rhs[-1] + 2*H[-1] * (contract.K-S_lowboundary)

            putprice = spsolve(LHSmatrix, rhs)   #You code this.  Hint...
            # numpy.linalg.solve, which expects arrays as inputs,
            # is fine for small matrix equations, and for matrix equations without special structure.
            # But for large matrix equations in which the matrix has special structure,
            # we may want a more intelligent solver that can run faster
            # by taking advantage of the special structure of the matrix.
            # Specifically, in this case, let's try to use a solver that recognizes the SPARSE MATRIX structure.
            # Try spsolve, imported from scipy.sparse.linalg

            putprice = np.maximum(putprice, contract.K-S)

        return(S, putprice)
```

[5]:

```
def strike_from_delta(Delta, S_0, r, sigma, T):
    return S_0 * np.exp((r + sigma**2/2) * T) / np.exp(sigma * np.sqrt(T) * norm.ppf(Delta))
```

[6]:

```
def price_from_delta(Delta, S_0, sigma, T):
    return S_0 * (Delta - norm.cdf(norm.ppf(Delta) - sigma * np.sqrt(T)) * np.exp(sigma**2/2 * T)/
                  np.exp(sigma * np.sqrt(T) * norm.ppf(Delta)))
```

1 Problem 1.

1.1 (a)

Using Ito's formula:

$$dC(S_t,t) = \dot{C}(S_t,t)dt + C'(S_t,t)dS_t + \frac{1}{2}C''(S_t,t)d < S >_t$$

Here, it was given that:

$$dS_t = \sigma S_t^{1+\alpha} dW_t$$

Therefore, the SDE for $C(S_t, t)$ becomes:

$$\begin{aligned}
dC(S_t,t) &= \dot{C}(S_t,t)dt + C'(S_t,t)(\sigma S_t^{1+\alpha} dW_t) + \frac{1}{2}C''(S_t,t)(\sigma S_t^{1+\alpha})^2 dt \\
&= \left(\dot{C}(S_t,t) + \frac{1}{2}C''(S_t,t)(\sigma S_t^{1+\alpha})^2\right) dt + C'(S_t,t)(\sigma S_t^{1+\alpha})dW_t \\
&= \left(\dot{C}(S_t,t) + \frac{1}{2}\sigma^2 S_t^{2(1+\alpha)} C''(S_t,t)\right) dt + \sigma S_t^{1+\alpha} C'(S_t,t) dW_t
\end{aligned}$$

We were given that the drift coefficient of $C(S_t, t)$ is r. From the SDE and the information above, we may conclude:

$$rC(S_t,t) = \dot{C}(S_t,t) + \frac{1}{2}\sigma^2 S_t^{2(1+\alpha)} C''(S_t,t)$$

Therefore, the PDE for C becomes:

$$rC = \frac{\delta C}{\delta t} + \frac{1}{2}\sigma^2 S^{2(1+\alpha)} \frac{\delta^2 C}{\delta S^2}$$

With the terminal condition:

$$C(S_T, T) = (K - S_T)^+$$

1.2 (b)

[7]:
```
hw4dynamics = CEV(volcoeff=3, alpha=-0.5, rGrow=0, r=0.05, S0=100)
```

[8]:
```
hw4contract = Put(T=0.25, K=100)
```

[9]:
```
hw4FD = FD_CrankNicolson(SMax=200,SMin=50,deltaS=0.1,deltat=0.0005)
```

[10]:
```
(S0_all, putprice) = hw4FD.price_put_CEV(hw4contract,hw4dynamics)
```

[11]:
```
# pricer_put_CEV_CrankNicolson gives us option prices for ALL S0 from SMin to
 ↪SMax
# But let's display only for a few S0 near 100:

displayStart = hw4dynamics.S0 - hw4FD.deltaS*1.5
displayEnd   = hw4dynamics.S0 + hw4FD.deltaS*1.5
displayrows  = (S0_all > displayStart) & (S0_all < displayEnd)
np.set_printoptions(precision=4, suppress=True)
print(np.stack((S0_all, putprice),axis=1)[displayrows])
```

```
[[100.1       5.8704]
 [100.        5.9183]
 [ 99.9       5.9665]]
```

The time-0 price of an American put on S with strike $K = 100$ and expiry $T = 0.25$ using Crank-Nicolson is ≈ 5.9183

1.3 (c)

We may estimate the time-0 delta of the put as:

$$\Delta \approx \frac{C(S_0 + \Delta S, 0) - C(S_0 - \Delta S, 0)}{2\Delta S}$$

Similarly, we may estimate the time-0 gamma of the put as:

$$\begin{aligned}\Gamma \approx& \frac{\frac{C(S_0+\Delta S,0)-C(S_0,0)}{\Delta S} - \frac{C(S_0,0)-C(S_0-\Delta S,0)}{\Delta S}}{\Delta S} \\ =& \frac{C(S_0 + \Delta S, 0) - 2C(S_0, 0) + C(S_0 - \Delta S, 0)}{(\Delta S)^2}\end{aligned}$$

The $C(S_0 + \Delta S, 0)$, $C(S_0, 0)$, and $C(S_0 - \Delta S, 0)$ values are provided in the solution from (b).

The ΔS value was set as 0.1

[12]:
```
time_0_delta = (putprice[S0_all > hw4dynamics.S0][-1] -
                putprice[S0_all < hw4dynamics.S0][0])/(2 * hw4FD.deltaS)

print("The time-0 delta numerically computed from (b) is approximately:",␣
 ↪round(time_0_delta, 3))
```

```
The time-0 delta numerically computed from (b) is approximately: -0.481
```

[13]:
```
time_0_gamma = (putprice[S0_all > hw4dynamics.S0][-1] -
                2 * putprice[S0_all == hw4dynamics.S0][0] +
                putprice[S0_all < hw4dynamics.S0][0])/(hw4FD.deltaS**2)

print("The time-0 gamma numerically computed from (b) is approximately:",␣
 ↪round(time_0_gamma, 4))
```

```
The time-0 gamma numerically computed from (b) is approximately: 0.0264
```

1.4 (d)

From the CEV dynamics:

$$dS_t = \sigma S_t^{1+\alpha} dW_t$$

If we set $\alpha = 0$ and $R_{grow} = r$ then it becomes a Black-Scholes GBM dynamics:

$$dS_t = rS_t dt + \sigma S_t dW_t$$

Therefore, creating a new dynamics with the adjusted values ($\sigma = 0.30$, $\alpha = 0$, \$R_{grow} $= 0.05$) and passing it on as an input for the same **FD_CrankNicolson.price_put_CEV** function would give the time -0 American put price of the desired Black-Scholes dynamics.

[14]:
```
hw4_p1d_dynamics = CEV(volcoeff=0.3, alpha=0, rGrow=0.05, r=0.05, S0=100)
```

[15]:
```
(p1d_S0_all, p1d_putprice) = hw4FD.price_put_CEV(hw4contract,hw4_p1d_dynamics)
```

[16]:
```
print("The time-0 American put price for Black-Scholes dynamics:",
      p1d_putprice[p1d_S0_all == hw4_p1d_dynamics.S0][0])
```

```
The time-0 American put price for Black-Scholes dynamics: 5.441979712760295
```

2 Problem 2.

2.1 (a)

First, under the Black-Scholes dynamics, with interest rate r and volatility σ, the price of a call option can be expressed as:

$$C(S_0, 0) = N(d_1)S_0 - N(d_2)Ke^{-rT}$$

where

$$d_1 = \frac{\log \frac{S_0}{K} + \left(r + \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}}$$

and

$$d_2 = d_1 - \sigma\sqrt{T}$$

We also know from the Black-Scholes formula that

$$\Delta_C = N(d_1)$$

The given Δ is $0 < \Delta < 1$ and the given $T > 0$. Therefore there is a solution.

First, from the definition:

$$d_1 = N^{-1}(\Delta_C)$$

Additionally, from the definition of d_1:

$$
\begin{aligned}
&\frac{\log \frac{S_0}{K} + \left(r + \frac{\sigma^2}{2}\right) T}{\sigma\sqrt{T}} = N^{-1}(\Delta_C) \\
\Leftrightarrow &\log \frac{S_0}{K} + \left(r + \frac{\sigma^2}{2}\right) T = \sigma\sqrt{T} N^{-1}(\Delta_C) \\
\Leftrightarrow &\log \frac{S_0}{K} = \sigma\sqrt{T} N^{-1}(\Delta_C) - \left(r + \frac{\sigma^2}{2}\right) T \\
\Leftrightarrow &\log S_0 - \log K = \sigma\sqrt{T} N^{-1}(\Delta_C) - \left(r + \frac{\sigma^2}{2}\right) T \\
\Leftrightarrow &\log K = \log S_0 - \sigma\sqrt{T} N^{-1}(\Delta_C) + \left(r + \frac{\sigma^2}{2}\right) T \\
\Leftrightarrow &K = S_0 \frac{e^{\left(r + \frac{\sigma^2}{2}\right) T}}{e^{\sigma\sqrt{T} N^{-1}(\Delta_C)}}
\end{aligned}
$$

Now replacing Δ_C with our given Δ, leads to the ultimate solution:

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

2.2 (b)

First, similar to 2 (a) we can find a relationship between the call price (C) and the delta (Δ).

From the Black-Scholes formula

$$
\begin{aligned}
&C(S_0, 0) = \Delta S_0 - N(d_2) K e^{-rT} \\
&\quad (\because \Delta = N(d_1)) \\
\Leftrightarrow &C(S_0, 0) = \Delta S_0 - N(d_1 - \sigma\sqrt{T}) K e^{-rT} \\
&\quad (\because d_2 = d_1 - \sigma\sqrt{T}) \\
\Leftrightarrow &C(S_0, 0) = \Delta S_0 - N(N^{-1}(\Delta) - \sigma\sqrt{T}) K e^{-rT} \\
&\quad (\because d_1 = N^{-1}(\Delta)) \\
\Leftrightarrow &C(S_0, 0) = \Delta S_0 - N(N^{-1}(\Delta) - \sigma\sqrt{T}) S_0 \frac{e^{\left(r + \frac{\sigma^2}{2}\right) T}}{e^{\sigma\sqrt{T} N^{-1}(\Delta)}} e^{-rT} \\
\Leftrightarrow &C(S_0, 0) = S_0 \left(\Delta - N\left(N^{-1}(\Delta) - \sigma\sqrt{T}\right) \cdot \frac{e^{\frac{\sigma^2}{2} T}}{e^{\sigma\sqrt{T} N^{-1}(\Delta)}} \right)
\end{aligned}
$$

Using this and the given $S_0 = 300$, $T = \frac{1}{12}$, $\sigma = 0.4$, and $r = 0.01$, we can get the strikes and premiums of the 25-delta call and 75-delta call as below:

[17]:
```
S_0 = 300
r = 0.01
sigma = 0.4
T = 1/12
Deltas = [0.25, 0.75]
```

```
strikes = [strike_from_delta(_, S_0, r, sigma , T) for _ in Deltas]

for i in range(len(Deltas)):
    print("The strike for", Deltas[i]*100, "delta call can be estimated as:",
 ↪strikes[i])
```

```
The strike for 25.0 delta call can be estimated as: 326.7403577236785
The strike for 75.0 delta call can be estimated as: 279.6109316029983
```

[18]:
```
premiums = [price_from_delta(_, S_0, sigma , T) for _ in Deltas]

for i in range(len(Deltas)):
    print("The premium for", Deltas[i]*100, "delta call can be estimated as:",
 ↪premiums[i])
```

```
The premium for 25.0 delta call can be estimated as: 4.882592053953938
The premium for 75.0 delta call can be estimated as: 26.103562887425046
```

2.3 (c)

The lambdas of the two options in part (b) can be calculated as:

[19]:
```
for i in range(len(Deltas)):
    print("The lambda for", Deltas[i]*100, "delta call can be estimated as:",
 ↪Deltas[i] * S_0 / premiums[i])
```

```
The lambda for 25.0 delta call can be estimated as: 15.36069349460903
The lambda for 75.0 delta call can be estimated as: 8.619513013236595
```

Therefore, the 25-delta call yields more leverage according to the lambda values.