hw4_sample solution

February 24, 2022

0.1 FinM 32000 HW4 Sample Solution

0.1.1 Problem 1

Part 1a $dC_t = \frac{\partial C}{\partial S_t} dS_t + \frac{\partial C}{\partial t} dt + \frac{1}{2}\frac{\partial^2 C}{\partial S_t^2}(dS_t)^2 = \frac{\partial C}{\partial S_t}\sigma S_t^{1+\alpha} + \frac{\partial C}{\partial t} dt + \frac{1}{2}\frac{\partial^2 C}{\partial S_t^2}\sigma^2 S_t^{2(1+\alpha)} dt$

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

with terminal condition

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

Part 1b Here g=0 and h=-r

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

[2]:
```
class Dynamics:
    pass
```

[3]:
```
class Contract:
    pass
```

[4]:
```
hw4contract=Contract()
hw4contract.T = 0.25
hw4contract.K = 100
```

[5]:
```
class FD:
    pass
```

[6]:
```
hw4FD=FD()
hw4FD.SMax=200
hw4FD.SMin=50
hw4FD.deltaS=0.1
hw4FD.deltat=0.0005
```

[7]:
```
# You complete the coding of this function

def pricer_put_CEV_CrankNicolson(contract,dynamics,FD):
```

```
# returns array of all initial spots,
# and the corresponding array of put prices

    volcoeff=dynamics.volcoeff
    alpha=dynamics.alpha
    r=dynamics.r

    T=contract.T
    K=contract.K

    # SMin and SMax denote the smallest and largest S in the _interior_.
    # The boundary conditions are imposed one step _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}

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

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

    ratio=deltat/deltaS
    ratio2=deltat/deltaS**2
    f = 0.5*(volcoeff**2)*(S**(2+2*alpha))  # You fill in with an array of the same size as S.
    g = dynamics.drift*S # You fill in with an array of the same size as S.
    h = -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*ratio*g
    G = ratio2*f-0.50*deltat*h
    H = 0.5*ratio2*f-0.25*ratio*g

    RHSmatrix = diags([H[:-1], 1-G, F[1:]], [1,0,-1], shape=(numS,numS), format="csr")
```

```
    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)*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]*(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 want a more intelligent solver that can run faster
        # by taking advantage of the special structure of the matrix.
        # Specifically, in this case, we want to use a solver that recognizes␣
↪the SPARSE MATRIX structure.
        # Try spsolve, imported from scipy.sparse.linalg
        putprice = np.maximum(putprice, K-S)

    return(S, putprice)
```

[8]:
```
hw4dynamics=Dynamics()

hw4dynamics.volcoeff = 3
hw4dynamics.alpha = -0.5
hw4dynamics.r = 0.05
hw4dynamics.S0 = 100
hw4dynamics.drift = 0
```

[9]:
```
(S0_all, putprice) = pricer_put_CEV_CrankNicolson(hw4contract,hw4dynamics,hw4FD)
```

[10]:
```
# 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=np.logical_and(S0_all>displayStart, S0_all<displayEnd)
np.set_printoptions(precision=4, suppress=True)
```

[11]:
```python
print(np.stack((S0_all, putprice),1)[displayrows])
```

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

Part 1c $\Delta = \frac{C_0(S_0{=}100.1)-C_0(S_0{=}99.9)}{2\Delta S} = (5.87041 - 5.9665)/0.2 = -0.4805$

$\Gamma = \frac{C_0(S_0{=}100.1)-2C_0(S_0{=}100.0)+C_0(S_0{=}99.9)}{(\Delta S)^2} = 0.0300$

Part 1d Here g=rS and h = -r

[12]:
```python
hw4dynamics.volcoeff = 0.3
hw4dynamics.alpha = 0
hw4dynamics.r = 0.05
hw4dynamics.S0 = 100
hw4dynamics.drift = 0.05
```

[13]:
```python
(S0_all, putprice) = pricer_put_CEV_CrankNicolson(hw4contract,hw4dynamics,hw4FD)
```

[14]:
```python
# 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=np.logical_and(S0_all>displayStart, S0_all<displayEnd)
np.set_printoptions(precision=4, suppress=True)
```

[15]:
```python
print(np.stack((S0_all, putprice),1)[displayrows])
```

```
[[100.1      5.3973]
 [100.       5.442 ]
 [ 99.9      5.487 ]]
```

0.1.2 Problem 2

Part a $\Delta = N(\frac{log(\frac{S_0 e^{rT}}{K})}{\sigma\sqrt{T}} + \sigma\frac{\sqrt{T}}{2})$

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

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

Part b

[16]:
```python
import numpy as np
from scipy.stats import norm
```

[17]:
```python
def BScallPrice(sigma,S,r,K,T):
    sd = sigma*np.sqrt(T)
    F = S*np.exp(r*T)
    d1 = np.log(F/K)/sd+sd/2
    d2 = d1-sd
    return S*norm.cdf(d1)-K*norm.cdf(d2)*np.exp(-r*T)
```

[18]:
```python
def KfromDelta(S,sigma,r,T,delta):
    invN = norm.ppf(delta)
    num = S*np.exp(r*T)
    den = np.exp(sigma*np.sqrt(T)*invN - 0.5*sigma**2*T)
    return num/den
```

[19]:
```python
K1 = KfromDelta(300,0.4,0.01,1/12,0.25)
K2 = KfromDelta(300,0.4,0.01,1/12,0.75)
C1 = BScallPrice(0.4,300,0.01,K1,1/12)
C2 = BScallPrice(0.4,300,0.01,K2,1/12)
```

[20]:
```python
# delta = 0.25
[K1, C1]
```

[20]:
```
[326.7403577236786, 4.882592053953928]
```

[21]:
```python
# delta = 0.75
[K2, C2]
```

[21]:
```
[279.61093160299833, 26.103562887425056]
```

Part c 25-delta call: $\Delta \frac{S_0}{C_0} = 0.25 \frac{300}{4.8826} = 15.36$

75-delta call: $\Delta \frac{S_0}{C_0} = 0.75 \frac{300}{26.1036} = 8.62$

[ ]:
```python

```