finm320-24-hw4

April 26, 2024

1 Question 1

1.1 (a)

$$dS_t = \sigma S_t^{1+\alpha} dW_t, \quad S_0 = 100$$

$$dC(S_t,t) = \left(\frac{\partial C}{\partial t}dt\right) + \left(\frac{\partial C}{\partial S_t}dS_t\right) + \frac{1}{2}\frac{\partial^2 C}{\partial S_t^2} b^2 S_t^{2+2\alpha} dt$$

$$= \left(\frac{\partial C}{\partial t}dt\right) + \left(\frac{\partial C}{\partial S_t}\sigma S_t^{1+\alpha} dW_t\right) + \frac{1}{2}\frac{\partial^2 C}{\partial S_t^2}\sigma^2 S_t^{2+2\alpha} dt$$

$$= \left(\frac{\partial C}{\partial t} + \frac{1}{2}\frac{\partial^2 C}{\partial S_t^2}\sigma^2 S_t^{2+2\alpha}\right) dt + \frac{\partial C}{\partial S_t}\sigma S_t^{1+\alpha} dW_t$$

Therefore, the drift term is given by rC and the corresponding PDE is:

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

with the terminal condition:

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

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

```
/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}"
```

[2]:
```python
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]:
```
hw4dynamics = CEV(volcoeff=3, alpha=-0.5, rGrow=0, r=0.05, S0=100)
```

[4]:
```
class Put:

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

[5]:
```
hw4contract=Put(T=0.25,K=100)
```

[6]:
```
class FD_CrankNicolson_Engine:

    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 = 0.5 * volcoeff**2 * S**(2*(1 + alpha))
        g = rGrow * S
        h = -r
        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)
            # 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)
```

[7]:
```
hw4FD = FD_CrankNicolson_Engine(SMax=200,SMin=50,deltaS=0.1,deltat=0.0005)
```

[8]:
```
(S0_all, putprice) = hw4FD.price_put_CEV(hw4contract,hw4dynamics)
```

[9]:
```
# 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]]
```

1.2 (c)

$$\begin{aligned}\Delta := \frac{\partial C}{\partial S} &\approx \frac{C(S+\Delta S,t)-C(S-\Delta S,t)}{2\Delta S}\\ &= \frac{5.8704-5.9665}{2\cdot 0.1} = -0.4805\end{aligned}$$

$$\begin{aligned}\Gamma := \frac{\partial^2 C}{\partial S^2} &\approx \frac{C(S+\Delta S,t)-2C(S,t)+C(S-\Delta S,t)}{(\Delta S)^2}\\ &= \frac{5.8704-2\cdot 5.9183+5.9665}{0.1^2} = 0.03\end{aligned}$$

1.3 (d)

[10]:
```
hw4dynamics2 = CEV(volcoeff=0.3, alpha=0, rGrow=0.05, r=0.05, S0=100)

(S0_all, putprice) = hw4FD.price_put_CEV(hw4contract,hw4dynamics2)

displayStart = hw4dynamics2.S0-hw4FD.deltaS*1.5
displayEnd   = hw4dynamics2.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      5.3973]
 [100.       5.442 ]
 [ 99.9      5.487 ]]
```

2 Question 2

2.1 (a)

[11]:

```
def calculate_expectation_and_std_simple(wins, total):
    # Calculate the expectation
    E = wins / total

    # Calculate the standard deviation
    std = np.sqrt((total - wins) / total * (0 - E) ** 2 + wins / total * (1 -␣
 ↪E) ** 2)

    return E, std

# Define parameters
wins = 34
total = 44

# Calculate expectation and standard deviation
expectation, std_deviation = calculate_expectation_and_std_simple(wins, total)

# Format the results
print(f"The expectation is {expectation:.2%}")
print(f"The standard deviation is {std_deviation:.2%}")
```

```
The expectation is 77.27%
The standard deviation is 41.91%
```

This scnario can be modeled as a Bernoulli process since Patrik's outcome is binary, either winning or losing. Extending this concept, it can be framed as a Binomial Distribution with only one trial, denoted as $n = 1$. Here, the variable X, which ranges from 0 to 1, signifies the proportion of the pot that Patrik secures.

For a Binomial Distribution, we have:

$$\mathbb{E}[X] = x \times p = 1 \times \frac{34}{44} + 0 \times \frac{10}{44} = 0.7727$$

$$\mathrm{Var}[X] = \sigma^2[X] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2 = x^2 \times p \times q = \frac{34}{44} \times \frac{10}{44} = 0.1756$$

$$\mathrm{Std}[X] = \sigma[X] = \sqrt{\mathrm{Var}[X]} = 0.4191$$

2.2 (b)

[12]:

```
import math

def calculate_fraction_expectation_and_std(wins, total, trials):
    # Calculate expectation
    expectation = trials * (wins / total) * (1/trials)
```

```
    # Calculate variance
    variance = trials * ((wins / total) * (1 - wins / total)) / (trials**2)

    # Standard deviation
    std_deviation = math.sqrt(variance)

    return expectation, std_deviation

# Define parameters
wins = 34
total = 44
trials = 3

# Calculate expectation and standard deviation
expectation, std_deviation = calculate_fraction_expectation_and_std(wins,
 ↪total, trials)

# Print the results with formatting
print(f"The expectation is {expectation:.2%}")
print(f"The standard deviation is {std_deviation:.2%}")
```

```
The expectation is 77.27%
The standard deviation is 24.20%
```

For the total expectation, with (n = 3), by replacement, we have:

$$\mathbb{E}[X] = \mathbb{E}[X_1] + \mathbb{E}[X_2] + \mathbb{E}[X_3] = x \times n \times p = \frac{1}{3} \times 3 \times \frac{34}{44} = 0.77273$$

$$\mathrm{Var}[X] = \mathrm{Var}[X_1] + \mathrm{Var}[X_2] + \mathrm{Var}[X_3] = x^2 \times n \times p \times q = \left(\frac{1}{3}\right)^2 \times 3 \times \frac{34}{44} \times \frac{10}{44} = 0.05854$$

$$\mathrm{Std}[X] = \sqrt{\mathrm{VAR}[X]} = 0.24195$$

2.3 (c)

[13]:
```
import math
def calculate_hypergeometric_expectation_std(total_wins, total_losses,
 ↪num_draws):
    total_outcomes = total_wins + total_losses
    E = 0
    VAR = 0

    # Calculate the expectation
    for i in range(num_draws + 1):
```

```python
        pi = math.comb(total_wins, i) * math.comb(total_losses, num_draws - i) /
 ↪ math.comb(total_outcomes, num_draws)
        E += pi * i / num_draws

    # Calculate the variance
    for i in range(num_draws + 1):
        pi = math.comb(total_wins, i) * math.comb(total_losses, num_draws - i) /
 ↪ math.comb(total_outcomes, num_draws)
        VAR += pi * ((i / num_draws) - E) ** 2

    # Calculate standard deviation
    std = math.sqrt(VAR)

    return E, std

# usage with provided values
total_wins = 34
total_losses = 10
num_draws = 3
E, std = calculate_hypergeometric_expectation_std(total_wins, total_losses,␣
 ↪num_draws)

# Formatting the results
print(f"The expectation is {E:.2%}.")
print(f"The standard deviation is {std:.2%}.")
```

```
The expectation is 77.27%.
The standard deviation is 23.63%.
```

For the total expectation without replacement:

The corresponding payout (fractional total pot) for each X is 0, $\frac{1}{3}$, $\frac{2}{3}$, and 1. We have:

$$P(X=0) = \frac{\binom{34}{0}\binom{10}{3}}{\binom{44}{3}} = 0.00906$$

$$P(X=1) = \frac{\binom{34}{1}\binom{10}{2}}{\binom{44}{3}} = 0.11552$$

$$P(X=2) = \frac{\binom{34}{2}\binom{10}{1}}{\binom{44}{3}} = 0.42359$$

$$P(X=3) = \frac{\binom{34}{3}\binom{10}{0}}{\binom{44}{3}} = 0.45183$$

$$\mathbb{E}[X] = 0 \times P(X=0) + \frac{1}{3} \times P(X=1) + \frac{2}{3} \times P(X=2) + 1 \times P(X=3) = 0.77273$$

$$\mathbb{E}[X^2] = 0^2 \times P(X=0) + \left(\frac{1}{3}\right)^2 \times P(X=1) + \left(\frac{2}{3}\right)^2 \times P(X=2) + 1^2 \times P(X=3) = 0.65292$$

$$\mathrm{Var}[X] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2 = 0.05582$$

$$\mathrm{Std}[X] = \sqrt{\mathrm{Var}[X]} = 0.23626$$

The standard deviation is smaller without replacement. This result makes sense because the reduced variability in the card deck due to not replacing cards leads to less variance in outcomes, making extreme results slightly less probable.