Ki_Hyun_12125881_FINM 32000_HW 7

May 19, 2023

FINM 32000 - Numerical Methods Spring 2023

Homework 7

Due - 23:59 [CST] May 19th, 2023

Ki Hyun

Student ID: 12125881

0.0.1 Imports

[1]:
```
import numpy as np
from sklearn.linear_model import LinearRegression
import scipy.optimize
import warnings
```

[2]:
```
warnings.filterwarnings("ignore")
```

0.0.2 Helper-Functions

[3]:
```
class GBM:

    def __init__(self,S0,r,sigma):
        self.S0 = S0
        self.r = r
        self.sigma = sigma
```

[4]:
```
class Put:

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

[5]:
```
class MC:

    def __init__(self,M,N,seed,algorithm):

        self.M = M # Number of paths
```

```
        self.N = N      # Number of time periods
        self.rng = np.random.default_rng(seed=seed) # Seeding the random number␣
↪generator with a specified number helps make the calculations reproducible

        self.algorithm = algorithm
        #'value' for Value-based approach (Longstaff-Schwartz) -- problem 1a
        #'policy' for Policy optimization -- problem 1b

    def price_americanPut_GBM(self,contract,dynamics):

        r=dynamics.r
        sigma=dynamics.sigma
        S0=dynamics.S0

        K=contract.K
        T=contract.T

        N=self.N
        M=self.M
        dt=T/N

        Z = self.rng.normal(size=(M,N))

        paths = S0*np.exp((r-sigma**2/2)*dt*np.tile(np.
↪arange(1,N+1),(M,1))+sigma*np.sqrt(dt)*np.cumsum(Z,axis=1))

        payoffDiscounted = np.maximum(0,K-paths[:,-1])
        #This is the payoff (cashflow) along each path,
        #discounted to time nn (for nn=N,N-1,...)
        #It corresponds to the far right-hand column in each page of the
        #Excel worksheet
        #I'm initializing it for time nn=N.

        #You could make payoffDiscounted
        #to be a matrix because it depends on nn.
        #But I will just reuse a 1-dimensional array,
        #by overwriting the time nn+1 entries at time nn.

        for nn in np.arange(N-1,0,-1):
            continuationPayoffDiscounted = np.exp(-r*dt)*payoffDiscounted
            # This is the CONTINUATION payoff (cashflow) along each path,
            # discounted to time nn (for nn=N-1,N-2,...)
            # It corresponds to the blue column in each page of the Excel␣
↪worksheet
            # Note that payoffDiscounted comes from the previous iteration
            # -- which was at time nn+1.  So now we discount back to time nn.
```

```python
        X=paths[:,nn-1]
        exerciseValue = K-X

        if self.algorithm == 'value':
            # This is the value function (Longstaff-Schwartz) approach. For problem 1a

            basisfunctions = np.stack((np.ones(M), X, X**2), axis = 1)#FILL THIS IN.  You may use np.stack
                    # This will be an M-by-3 array containing the basis functions (Same ones as L7.9-7.10, and Excel)

            # conducting regression only on the paths that are in-the-money
            A = basisfunctions[(exerciseValue > 0), :]
            y = continuationPayoffDiscounted[exerciseValue > 0]
            lm = LinearRegression()
            lm.fit(X = A, y = y)
            coefficients = lm.coef_ # FILL THIS IN
                    # This will be an array of 3 estimated "betas".

            estimatedContinuationValue = np.dot(basisfunctions, coefficients) # FILL THIS IN
                    # with an array of length M.
                    # This is similar to the Red column in Excel

            whichPathsToExercise = (exerciseValue >= np.maximum(estimatedContinuationValue,0))
                    #This is a length-M array of Booleans

        elif self.algorithm == 'policy':
            # This is the policy optimization approach to Reinforcement learning.  For problem 1b

            (a_opt,b_opt) = scipy.optimize.minimize(negofMCaverageOfExpectedPayouts,(0,0),
                                                     args=(X,exerciseValue,continuationPayoffDiscounted),
                                                     method='Nelder-Mead').x
                #Chose Nelder-Mead optimizer because it is generating reasonable results with minimal coding effort
                #But gradient methods, done properly, usually run faster

            whichPathsToExercise = ((softExercise(X, a_opt, b_opt) >= 0.5) & (exerciseValue > 0))
                #FILL THIS IN, using the right-hand side of the last equation on the homework sheet
```

```
                    #This obtains the hard exercise decision from the optimized␣
↪soft exercise function
                    #It should be a length-M array of Booleans (as it was in␣
↪the "value" approach.
                    #But here it comes from the softExercise function)

            else:
                raise ValueError('Unknown algorithm type')


            payoffDiscounted[whichPathsToExercise] =␣
↪exerciseValue[whichPathsToExercise]
            # FILL THIS IN -- see the
                # "discounted cashflow along path"
            # column in Excel
            payoffDiscounted[np.logical_not(whichPathsToExercise)] =␣
↪continuationPayoffDiscounted[np.logical_not(whichPathsToExercise)]
            # FILL THIS IN -- see the
                # "discounted cashflow
            # along path" column in Excel

        # The time-0 calculation needs no regression
        continuationPayoffDiscounted = np.exp(-r*dt)*payoffDiscounted
        estimatedContinuationValue = np.mean(continuationPayoffDiscounted)
        putprice = max(K-S0,estimatedContinuationValue)

        return(putprice)
```

[6]:

```
# for Policy optimization approach, problem 1b
#
# If b<<0 then this function essentially returns nearly 1 if X<a, or nearly 0␣
↪if X>a
# but with some smoothing of the discontinuity, using a sigmoid function, to␣
↪help the optimizer

def softExercise(X,a,b):
    return 1/(1+np.exp(-b*(X-a)))
```

[7]:

```
# for Policy optimization approach, problem 1b

def negofMCaverageOfExpectedPayouts(coefficients, x, exercisePayoff,␣
↪continuationPayoff):

    p = softExercise(x,*coefficients)

    # p and exercisePayoff and continuationPayoff are all length-M arrays
```

```
    return -np.mean(p * exercisePayoff + (1 - p) * continuationPayoff)# FILL␣
↪THIS IN

## You fill in, what to return.  It should be the negative of the expression␣
↪inside the max() on the homework sheet.
## Need to take the negative because we are calling "minimize" but we want to␣
↪do _maximization_
```

1 Problem 1.

[8]:
```
hw7dynamics = GBM(S0=1, r=0.03, sigma=0.20)
```

[9]:
```
hw7contract = Put(K=1.1, T=4)
```

1.0.1 (Problem 1a)

[10]:
```
hw7MC_a = MC(M=10000, N=4, seed=0, algorithm='value')
```

[11]:
```
hw7MC_a.price_americanPut_GBM(hw7contract,hw7dynamics)
```

[11]: 0.13779107851434708

1.0.2 (Problem 1b)

[12]:
```
hw7MC_b = MC(M=10000, N=4, seed=0, algorithm='policy')
```

[13]:
```
hw7MC_b.price_americanPut_GBM(hw7contract,hw7dynamics)
```

[13]: 0.16263529459015832

2 Problem 2.

2.1 (a)

The 2nd derivative of $F(u,v)$ with respect to u is:

$$\begin{aligned}\frac{\delta^2}{\delta u^2}F(u,v) &= \mathbf{E}[(iR_1)^2 e^{iuR_1+ivR_2}] \\ &= \mathbf{E}[-R_1^2 e^{iuR_1+ivR_2}] \\ &= -\mathbf{E}[R_1^2 e^{iuR_1+ivR_2}]\end{aligned}$$

If we evaluate this second derivative at $u=0, v=0$:

$$\frac{\delta^2}{\delta u^2} F(0,0) = -\mathbf{E}[R_1^2]$$

Similarly, 2nd derivative of $F(u,v)$ with respect to v is:

$$\begin{aligned}\frac{\delta^2}{\delta v^2} F(u,v) &= \mathbf{E}[(iR_2)^2 e^{iuR_1+ivR_2}] \\ &= \mathbf{E}[-R_2^2 e^{iuR_1+ivR_2}] \\ &= -\mathbf{E}[R_2^2 e^{iuR_1+ivR_2}]\end{aligned}$$

If we evaluate this second derivative at $u=0, v=0$:

$$\frac{\delta^2}{\delta v^2} F(0,0) = -\mathbf{E}[R_2^2]$$

Since we know that

$$\mathbf{E}(R_1^2 + R_2^2) = \mathbf{E}(R_1^2) + \mathbf{E}(R_2^2)$$

The answer in terms of F would be:

$$\mathbf{E}(R_1^2 + R_2^2) = -\frac{\delta^2}{\delta u^2} F(0,0) - \frac{\delta^2}{\delta v^2} F(0,0)$$

2.2 (b)

By definition:

$$\psi(w) := \mathbf{E}[e^{iw(4R_1-3R_2)}]$$

Furthermore,

$$\begin{aligned}\mathbf{E}[e^{iw(4R_1-3R_2)}] &= \mathbf{E}[e^{i4wR_1-3wR_2}] \\ &= F(4w,-3w)\end{aligned}$$

Therefore, $\psi(w)$ can be expressed as $F(4w,-3w)$.

2.3 (c)

Again, by definition,

$$G(x,y) := \mathbf{E}e^{ixR_3+iyR_4}$$

Additionally,

$$
\begin{aligned}
\mathbf{E}e^{ixR_3+iyR_4} &= \mathbf{E}e^{ixR_3} \times e^{iyR_4} \\
&= (\mathbf{E}e^{ixR_3}) \times (\mathbf{E}e^{iyR_4}) \\
&(\because R_3 \perp\!\!\!\perp R_4) \\
&= (\mathbf{E}e^{ixR_1}) \times (\mathbf{E}e^{iyR_2}) \\
&(\because R_3 \sim R_1,\ R_4 \sim R_2) \\
&= F(x,0) \times F(0,y)
\end{aligned}
$$

Ultimately,

$$
G(x,y) = F(x,0) \times F(0,y)
$$