Ki_Hyun_12125881_FINM 32000_HW5

May 5, 2023

FINM 32000 - Numerical Methods Spring 2023

Homework 5

Due - 23:59 [CST] May 5th, 2023

Ki Hyun

Student ID: 12125881

0.0.1 Imports

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

0.0.2 Helper-Functions

[2]:
```
# Exponential Ornstein-Uhlenbeck process

class XOU:

    def __init__(self, kappa, alpha, sigma, S0, r):

        self.kappa = kappa
        self.alpha = alpha
        self.sigma = sigma
        self.S0 = S0
        self.r = r
```

[3]:
```
class CallOnForwardPrice:

    def __init__(self, K1, T1, T2):

        self.K1 = K1
        self.T1 = T1
        self.T2 = T2
```

[4]:
```
class MC:

    def __init__(self, N, M, epsilon, seed):
```

```
        self.N = N   # Number of timesteps on each path
        self.M = M   # Number of paths
        self.epsilon = epsilon  # For the dC/dS calculation
        self.rng = np.random.default_rng(seed=seed) # Seeding the random number␣
↪generator with a specified number helps make the calculations reproducible

    def price_call_XOU(self, contract, dynamics):

        # You complete the coding of this function
        # self.rng.normal() generates pseudo-random normals
        X_0 = np.log(dynamics.S0) # getting X_0 from the dynamics
        X_t = np.repeat(X_0, self.M) # the X_t grid
        deltat = contract.T1 / self.N # time-step
        rands = self.rng.normal(size = (self.N, self.M)) # random numbers␣
↪generated
        for t in range(1, self.N):
            X_t = X_t + (dynamics.kappa * (dynamics.alpha - X_t) * deltat +
                         rands[t, :] * dynamics.sigma * np.sqrt(deltat)) # X_t␣
↪based on dynamics

        S_T1 = np.exp(X_t) # the Monte Carlos simulated underlying price at T1
        # calculating forward price at T_1 based on formula
        F_T1 = np.exp(np.exp(-dynamics.kappa * (contract.T2 - contract.T1)) *␣
↪np.log(S_T1) +
                      (1 - np.exp(-dynamics.kappa * (contract.T2 - contract.
↪T1))) * dynamics.alpha +
                      dynamics.sigma**2 / (4 * dynamics.kappa) * (1 - np.exp(-2␣
↪* dynamics.kappa *
                                                                             ␣
↪(contract.T2 - contract.T1))))
        # getting the different call prices
        call_price = np.maximum(F_T1 - contract.K1, 0.0) * np.exp(-dynamics.r *␣
↪contract.T1)
        # standard error of call prices
        standard_error = np.std(call_price) / np.sqrt(self.M)
        # averaging the simulations to get one call price
        call_price = np.mean(call_price)

        # calculation for call delta
        X_0 = np.log(dynamics.S0 + self.epsilon) # little change in S_0 price
        X_t = np.repeat(X_0, self.M)
        for t in range(1, self.N):
            X_t = X_t + (dynamics.kappa * (dynamics.alpha - X_t) * deltat +
                         rands[t, :] * dynamics.sigma * np.sqrt(deltat)) # X_t␣
↪based on dynamics
```

```
        S_T1_prime = np.exp(X_t)
        F_T1_prime = np.exp(np.exp(-dynamics.kappa * (contract.T2 - contract.
↪T1)) * np.log(S_T1_prime) +
                           (1 - np.exp(-dynamics.kappa * (contract.T2 -␣
↪contract.T1))) * dynamics.alpha +
                           dynamics.sigma**2 / (4 * dynamics.kappa) * (1 - np.
↪exp(-2 * dynamics.kappa *
                                                                        ␣
↪   (contract.T2 - contract.T1))))
        call_delta = np.maximum(F_T1_prime - contract.K1, 0.0) * np.
↪exp(-dynamics.r * contract.T1)
        call_delta = (call_delta - call_price)/self.epsilon
        call_delta = np.mean(call_delta)

        return(call_price, standard_error, call_delta)
```

[5]:
```
def delta_forward(dynamics, contract):
    return np.exp(-(dynamics.r - dynamics.kappa) * contract.T2
                  - (1 - np.exp(-dynamics.kappa * contract.T2)) * np.
↪log(dynamics.S0)
                  + (1 - np.exp(-dynamics.kappa * contract.T2)) * dynamics.alpha
                  + dynamics.sigma**2 / (4*dynamics.kappa) * (1 - np.exp(-2 *␣
↪dynamics.kappa * contract.T2)))
```

[6]:
```
def choice(n, k):
    k = np.minimum(n - k, k)
    temp = 1.0
    for i in range(k):
        temp *= (n - i)/(k - i)
    return temp
```

1 Problem 1.

1.1 (a)

Let's consider a portfolio that long one (K, T_2)-forward contract, and short one (F_t, T_2)-forward contract.

First, we know that the (F_t, T_2)-forward contract has time-t value of 0. Moreover, the (K, T_2)-forward contract's time-t value was assumed as f_t. Therefore, the time-t portfolio has the value of f_t.

Second, we may consider the payoff at time T_2. The payoff at time T_2 for the portfolio can be written as $(S_{T_2} - K) - (S_{T_2} - F_t) = F_t - K$. Therefore, the time-$t$ portfolio value can also be considered as the discounted payoff of $e^{-r(T_2-t)}(F_t - K)$ (given the constant interest rate of r).

Assuming no arbitrage, we may conclude:

$$f_t = e^{-r(T_2-t)}(F_t - K)$$

1.2 (b)

We cannot simply replace "stock" with "crude oil" due to the part "At time T_2, deliver the stock, and receive F_t". For stocks, it may be easier to assume frictionless market where delivering the stock does not inflict any cost. Nevertheless, for crude oil, delivering the asset would result in cost that may fail the argument of $F_t = S_t e^{r(T_2-t)}$.

1.3 (c)

[7]:
```
hw5dynamics = XOU(kappa=0.472, alpha=4.4, sigma=0.368, S0=106.9, r=0.05)
```

[8]:
```
hw5contract = CallOnForwardPrice(K1=103.2, T1=0.5, T2=0.75)
```

[9]:
```
hw5MC = MC(N=100, M=10**5, epsilon=0.01, seed=0)
# Change M if necessary
```

[10]:
```
(call_price, standard_error, call_delta) = hw5MC.
↪price_call_XOU(hw5contract,hw5dynamics)
```

[11]:
```
print(call_price, standard_error)
```

```
7.725480196410691 0.04186002821111604
```

1.4 (d)

[12]:
```
print(call_delta)
```

```
0.3418041783687046
```

1.5 (e)

We know from (a) that:

$$f_t = e^{-r(T_2-t)}(F_t - K)$$

Therefore,

$$f_0 = e^{-rT_2}(F_0 - K)$$

In other words,

$$\frac{\delta f_0}{\delta S} = \frac{\delta}{\delta S} e^{-rT_2}(F_0 - K) = e^{-rT_2}\frac{\delta}{\delta S}F_0$$

The formula for F_t from the dynamics was also given as:

$$F_t = \mathbf{E}_t(S_{T_2}) = \exp\left[e^{-\kappa(T_2-t)}\log S_t + (1-e^{-\kappa(T_2-t)})\alpha + \frac{\sigma^2}{4\kappa}(1-e^{-2\kappa(T_2-t)})\right]$$

Therefore,

$$F_0 = \exp\left[e^{-\kappa T_2}\log S_0 + (1-e^{-\kappa T_2})\alpha + \frac{\sigma^2}{4\kappa}(1-e^{-2\kappa T_2})\right]$$

Substituting this equation back into the partial differentiation gives:

$$\begin{aligned}
\frac{\delta f_0}{\delta S} &= e^{-rT_2}\frac{\delta}{\delta S}F_0 \\
&= e^{-rT_2}\frac{\delta}{\delta S}\exp\left[e^{-\kappa T_2}\log S_0 + (1-e^{-\kappa T_2})\alpha + \frac{\sigma^2}{4\kappa}(1-e^{-2\kappa T_2})\right] \\
&= e^{-rT_2}\frac{\delta}{\delta S}\exp\left[e^{-\kappa T_2}\log S_0\right] \times \exp\left[(1-e^{-\kappa T_2})\alpha + \frac{\sigma^2}{4\kappa}(1-e^{-2\kappa T_2})\right] \\
&= \exp\left[-rT_2 + (1-e^{-\kappa T_2})\alpha + \frac{\sigma^2}{4\kappa}(1-e^{-2\kappa T_2})\right]\frac{\delta}{\delta S}S^{e^{-\kappa T_2}} \\
&= \exp\left[-rT_2 + (1-e^{-\kappa T_2})\alpha + \frac{\sigma^2}{4\kappa}(1-e^{-2\kappa T_2})\right]e^{-\kappa T_2}S^{e^{-\kappa T_2}-1} \\
&= \exp\left[-rT_2 - \kappa T_2 + (e^{-\kappa T_2}-1)\log S_0 + (1-e^{-\kappa T_2})\alpha + \frac{\sigma^2}{4\kappa}(1-e^{-2\kappa T_2})\right] \\
&= \exp\left[-(r-\kappa)T_2 - (1-e^{-\kappa T_2})\log S_0 + (1-e^{-\kappa T_2})\alpha + \frac{\sigma^2}{4\kappa}(1-e^{-2\kappa T_2})\right]
\end{aligned}$$

Therefore, analytically, $\frac{\delta f_0}{\delta S}$ can be calculated as:

$$\frac{\delta f_0}{\delta S} = \exp\left[-(r-\kappa)T_2 - (1-e^{-\kappa T_2})\log S_0 + (1-e^{-\kappa T_2})\alpha + \frac{\sigma^2}{4\kappa}(1-e^{-2\kappa T_2})\right]$$

The numerical value of this partial differentiation can be calculated by substituting the constants.

[13]:
```
forward_delta = delta_forward(hw5dynamics, hw5contract)
print(forward_delta)
```

```
1.3123701857785868
```

1.6 (f)

For replication of a long one call would be determined by:

$$\frac{\delta C}{\delta f_0}$$

Using rules of differentiation, the above can also be expressed as

$$\frac{\delta C}{\delta f_0} = \frac{\delta C}{\delta S} / \frac{\delta f_0}{\delta S}$$

The above can be calculated using the Monte Carlo simulation calculated $\frac{\delta C}{\delta S}$ and analytically calculated $\frac{\delta f_0}{\delta S}$

[14]:
```
print(call_delta / forward_delta)
```

```
0.260447991026193
```

Therefore, approximately 0.260 units of forward contracts at time-0 would replicate the call

1.7 (g)

The holder of the contract would purchase 5,000 barrels if $(F_{T_1} - K) > 0$.

On the other hand, the holder of the contract would purchase 4,000 barrels if $(F_{T_1} - K) < 0$.

In mathematical equation:

$$\theta = \begin{cases} 5000 & if\ (F_{T_1} - K) > 0 \\ 4000 & if\ (F_{T_1} - K) < 0 \end{cases}$$

Therefore, this "purchase agreement" contract would be equivalent to 4,000 long forward contract with delivery date T_2 and delivery price K, and 1,000 long option contract from (a).

The time-0 value of this contract can be calculated without simulation as:

$$1000 \cdot C(S_0) + 4000 \cdot f_0$$

This value can be calculated as below:

[15]:
```
F_0 = np.exp(np.exp(-hw5dynamics.kappa * hw5contract.T2) * np.log(hw5dynamics.
↪S0)
             + (1 - np.exp(-hw5dynamics.kappa * hw5contract.T2)) * hw5dynamics.
↪alpha
             + hw5dynamics.sigma**2 / (4*hw5dynamics.kappa) * (1 - np.exp(-2 *␣
↪hw5dynamics.kappa * hw5contract.T2)))
f_0 = np.exp(-hw5dynamics.r * hw5contract.T2) * (F_0 - hw5contract.K1)
```

[16]:
```
print(1000 * call_price + 4000 * f_0)
```

```
3989.6430457929664
```

2 Problem 2.

2.1 (a)

Let P be the fraction of the pot that Patrik will collect. Then,

$$P = \begin{cases} 1 & \text{with prob } \frac{34}{44} \\ 0 & \text{with prob } \frac{10}{44} \end{cases}$$

Since P follows a Bernoulli distribution:

$$\mathbf{E}[P] = \frac{34}{44}$$

Moreover,

$$Var[P] = \frac{34}{44} \times \frac{10}{44} = \frac{34 \cdot 10}{44^2}$$

Therefore, the standard deviation is

$$\sqrt{\frac{34 \cdot 10}{44^2}} = \frac{2\sqrt{85}}{44} = \frac{\sqrt{85}}{22} \approx 0.4191$$

2.2 (b)

This time P will take the values:

$$P = \begin{cases} 1 & \text{with prob } (\frac{34}{44})^3 \\ \frac{2}{3} & \text{with prob } 3 \cdot (\frac{34}{44})^2 \cdot \frac{10}{44} \\ \frac{1}{3} & \text{with prob } 3 \cdot \frac{34}{44} \cdot (\frac{10}{44})^2 \\ 0 & \text{with prob } (\frac{10}{44})^3 \end{cases}$$

Therefore, the expected value can be expressed as:

$$\mathbf{E}[P] = 1 \cdot (\frac{34}{44})^3 + \frac{2}{3} \cdot 3 \cdot (\frac{34}{44})^2 \cdot \frac{10}{44} + \frac{1}{3} \cdot 3 \cdot \frac{34}{44} \cdot (\frac{10}{44})^2 + 0 \cdot (\frac{10}{44})^3$$

and can be calculated as:

[17]:
```
prob = 34/44

exp_p = (1 * prob**3 +
         2/3 * 3 * prob**2 * (1 - prob) +
         1/3 * 3 * prob * (1 - prob)**2 +
         0 * (1 - prob)**3)

print(exp_p)
```

```
0.7727272727272727
```

Now the variance can be expressed as:

$$
\begin{aligned}
Var[P] &= \mathbf{E}[(P - \mathbf{E}[P])^2] \\
&= (1 - \mathbf{E}[P])^2 \cdot (\frac{34}{44})^3 + (\frac{2}{3} - \mathbf{E}[P])^2 \cdot 3 \cdot (\frac{34}{44})^2 \cdot \frac{10}{44} + (\frac{1}{3} - \mathbf{E}[P])^2 \cdot 3 \cdot \frac{34}{44} \cdot (\frac{10}{44})^2 + (0 - \mathbf{E}[P])^2 \cdot (\frac{10}{44})^3
\end{aligned}
$$

and can be calculated as:

[18]:
```
var_p = ((1 - exp_p)**2 * prob**3 +
         (2/3 - exp_p)**2 * 3 * prob**2 * (1 - prob) +
         (1/3 - exp_p)**2 * 3 * prob * (1 - prob)**2 +
         (0 - exp_p)**2 * (1 - prob)**3)
```

Therefore, the standard deviation can be calculated getting the square root of the variance:

[19]:
```
print(np.sqrt(var_p))
```

```
0.24195029428289866
```

2.3 (c)

This time P will take the values:

$$
P = \begin{cases}
1 & \text{with prob } \dfrac{\binom{34}{3}}{\binom{44}{3}} \\
\frac{2}{3} & \text{with prob } \dfrac{\binom{34}{2}\binom{10}{1}}{\binom{44}{3}} \\
\frac{1}{3} & \text{with prob } \dfrac{\binom{34}{1}\binom{10}{2}}{\binom{44}{3}} \\
0 & \text{with prob } \dfrac{\binom{10}{3}}{\binom{44}{3}}
\end{cases}
$$

The expectation can be calculated, similar to (b):

$$\mathbf{E}[P] = \sum P_i \times \mathbf{P}[P = P_i]$$

and the numeric value is

[20]:
```
probs = np.array([choice(10, 3 - _) * choice(34, _)/choice(44, 3) for _ in␣
 ↪range(4)])
ps = np.array([_/3 for _ in range(4)])
exp_p_c = np.sum(probs * ps)
print(exp_p_c)
```

```
0.7727272727272727
```

The variance can be calculated, similar to (b):

$$Var[P] = \sum (P_i - \mathbf{E}[P])^2 \times \mathbf{P}[P = P_i]$$

and the numeric value is

[21]:
```
var_p_c = np.sum((ps - exp_p_c)**2 * probs)
print(var_p_c)
```

```
0.055817156768530975
```

Therefore, the standard deviation can be calculated taking the square root of the variance as:

[22]:
```
print(np.sqrt(var_p_c))
```

```
0.23625654862570683
```

The standard deviation is smaller in part (c) compared to part (b). This make sense when thinking about the probability of getting extreme outcomes (i.e., 1 or 0). Without replacement, there is lower probability to get extreme outcomes since drawing cards in a streak in both direction happens with lower probability–compared to draws made with replacement.