Inventory Problem of a Perishable Product
STA4001

GUO Yuanxin
118010084

November 3, 2019

Contents

1 Problem Formulation

Consider a perishable product with a maximum lifetime of L periods. Thus, at the end of period t, there may be leftover inventory with a remaining lifetime $1 \leq j \leq L-1$. Let X_tj be the amount of inventory with remaining lifetime less or equal to j, then the system state in period t is $X_t = (X_{t1}, X_{t2}, \cdots, X_{t,L-1})$.

At the beginning of period t, an order for $Q - X_{t-1,L-1}$ units of new inventory placed at the end of period $t-1$ arrives, bringing the total inventory level to Q. A random demand D_t then occurs and is met immediately as much as possible with inventory from the oldest to the newest. Demand across dierent periods are assumed to be i.i.d. and follow Poisson distribution with mean λ. Unmet demand is lost. At the end of period t expired inventory is disposed at a cost c_d per unit, and leftover inventory incurs a holding cost h per unit.

We will soon show that $X = \{X_t, t = 0, 1, \dots\}$ is a DTMC. We use $\mathcal{S}$ to denote the state space of the Markov chain. The objective is to compute the value function:

$$v(x) = \mathbb{E}[\sum_{n=0}^{\infty} \beta^n g(X_n) | X_0 = x]$$

for different initial states $x \in \mathcal{S}$, where $g(x)$ is the single period expected prot function given the state x in the previous period.

1.1 Parameters in this problem

Notation	Parameter	Value
c_p	Selling price	\$1.0
c_v	Variable cost	\$0.4
c_d	Disposal cost	\$0.1
h	Holding cost	\$0.1
λ	Demand rate	17
Q	Total inventory level at the beginning of period	90
L	Maximum lifetime of perishable product	7
β	Discount factor	0.8

2 Analysis of the Process

2.1 Verification of Markovity

To show that this process is a DTMC, we aim to find the relationship between X_n and X_{n+1}. If X_{n+1} is a function of X_n and a random variable, the Markovity is thus shown.

During each period, the system state undergoes following changes:

a. The total inventory level is brought up to 90, while the remaining lifetime of each product reduce by 1. We use vector $Y_t = (Y_{t0}, Y_{t1}, \cdots, Y_{t6})$ to denote the inventory state at the moment, where $Y_t j$ be the amount of inventory with remaining lifetime less or equal to j. At the moment, $Y_t = (X_{t1}, X_{t2}, \cdots, X_{t,L-1}, 90)$.

b. The products are selled at a random demand D_t according to Poisson distribution from the oldest to the newest. Unmet demand is lost. $Y_t = ((X_{t1} - D_t)^+, (X_{t2} - D_t)^+, \cdots, (X_{t6} - D_t)^+, (90 - D_t)^+)$.

c. The expired inventory, namely the products with remaining lifetime equal to 0, are disposed. $Y_t = (0, (X_{t2} - D_t)^+ - (X_{t1} - D_t)^+, \cdots, (X_{t6} - D_t)^+ - (X_{t1} - D_t)^+, (90 - D_t)^+ - (X_{t1} - D_t)^+)$.

d. At the beginning of period $t+1$, the system state becomes $X_{t+1} = ((X_{t2} - D_t)^+ - (X_{t1} - D_t)^+, \cdots, (X_{t6} - D_t)^+ - (X_{t1} - D_t)^+, (90 - D_t)^+ - (X_{t1} - D_t)^+)$.

Hence, it is not difficult to see that the expression of X_{t+1} contains solely terms of X_t and random variable D_t, thus proving that $X = \{X_t, t = 0, 1, \dots\}$ is a DTMC.

2.2 Transition Probabilities

Consider random variable $D_t \sim (Poisson, 17)$ with pmf

$$p_{D_t}(x) = \frac{17^x e^{-17}}{x!}.$$

Then the transition probability p_{ij} is:

$$\begin{cases} p_{D_t}(x), & j = ((i_2 - x)^+ - (i_1 - x)^+, \cdots (i_6 - x)^+ - (i_1 - x)^+, (90 - x)^+ - (i_1 - x)^+),\ x < 90 \\ \sum_{x=90}^{\infty} p_{D_t}(x), & j = ((i_2 - x)^+ - (i_1 - x)^+, \cdots (i_6 - x)^+ - (i_1 - x)^+, (90 - x)^+ - (i_1 - x)^+),\ x = 90 \\ 0, & \text{otherwise} \end{cases}$$

2.3 Value function

The value function $g(x)$ takes previous state X_t as the input and outputs the profit dependent on the realization of the random variable D_{t+1}. The analysis is analogous to that in (2.1).

a. Bringing the total inventory up to 90 requires ordering $(90-X_{t6})$ products. This costs: $c_v \cdot (90 - X_{t6})$.

b. The profit due to selling products depends on the outcome of the demand random variable. The number of sales is $\min(D_{t+1}, 90)$.
Hence this part of the profit is: $c_p \cdot \min(D_{t+1}, 90)$

c. The items with lifetime equal to 0 are expired and to be disposed. There are $(X_1 - D_{t+1})^+$ of them.
The disposal cost is: $c_d \cdot (X_{t1} - D_{t+1})^+$.

d. Finally, all of the rest inventory at a total quantity of $(90 - D_t)^+ - (X_{t1} - D_t)^+$ requires a holding cost.
The holding cost is: $h \cdot (90 - D_t)^+ - (X_{t1} - D_t)^+$.

In summary, the explicit form of the value function $g(x)$ is:

$$\begin{aligned} g(x) &= -c_v(90 - X_{t6}) + c_p \min(D_{t+1}, 90) - c_d(X_{t1} - D_{t+1})^+ - h\left[(90 - D_t)^+ - (X_{t1} - D_t)^+\right] \\ &= \min(D_{t+1}, 90) - 0.4(90 - X_{t6}) - 0.1(X_{t1} - D_{t+1})^+ - 0.1\left[(90 - D_t)^+ - (X_{t1} - D_t)^+\right] \end{aligned}$$

3 Monte Carlo Method

To evaluate the expectation of the value function $v(x)$, it is almost impossible to compute analytically as the state space grows exponentially with L. Thus, we consider using simulation to approximate the result. A common approach is using the Monte Carlo method.

The basic idea of the Monte Carlo simulation is to find a set of sample paths and calculate $\mathbb{E}[\sum_{n=0}^{\infty} \beta^n g(X_n) | X_0 = x]$ over the set. The method approximates well provided both the time horizon and episode are large enough, justified by the Law of Large Numbers.

We run the code in (A) and obtain Figure 1 below. As the number of episodes and the length of time horizon increases, the estimation $\hat{v}(x)$ of the value function becomes more centered around 14.5. The fluctuations of both the estimation and the CI's become milder.

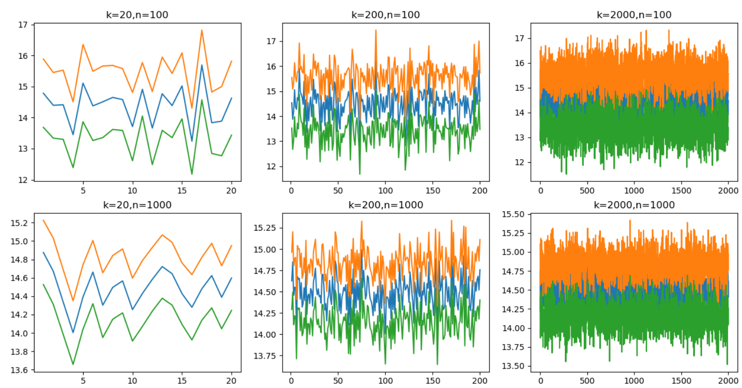

Figure 1: Monte Carlo Method.

4 Neural Network Approach

Consider $|\mathcal{S}|$, the size of the state space. We use the following Python code to compute it:

```
count = 0
for a in range(91):
    for b in range(a,91):
        for c in range(b,91):
            for d in range(c,91):
                for e in range(d,91):
                    for f in range(e,91):
                        count+=1
print("The size of the state space is",count)
```

The output is:

```
The size of the state space is 927048304
```

The state space is horribly large even for L=7, which makes it infeasible for the computer to store the $\hat{v}(x)$ for all $x \in \mathcal{S}$. Therefore, Monte Carlo method is only an expedient for a rather small subset of the state space.

One possible solution to estimate $v(x)$ for all $x \in \mathcal{S}$ is to train a neural network. We can apply the method in (3) to obtain a dataset $(\bar{X}, \bar{Y})$ for a relatively small subset $\bar{X}$, where $\bar{Y} = \{\hat{v}(\bar{x}_k) | \bar{x}_k \in \bar{X}, k = 1, \ldots, K\}$, K is the size of the sample space. Now we can use $(\bar{X}, \bar{Y})$ to train a neural network to approximate the value function over the whole state space.

We run the code in (B) in order to obtain the neural network model and solve the subquestions in Part 2.

4.1 Subquestion (a)

For $x = (10, 20, 30, 40, 50, 80)$, the estimated values are as follows:

$\tilde{v}(x)$ $(K = 20)$	14.52520264
$\tilde{v}(x)$ $(K = 200)$	14.41595547
$\tilde{v}(x)$ $(K = 2000)$	14.52172661
$\hat{v}(x)$	17.06012879080918
CI(95%)	[16.719012309096062,17.4012452725223]

We discover that all of the predictions do not lie in the 95% confidence interval. A tentative explanation of this result is that the x value that we test is an outlier. In other words, the probability of visiting the state is scarce. The neural network model may not perform well when receiving an unlikely input.

4.2 Subquestion (b)(c)

Figure (2) includes the Monte Carlo estimation, neural network prediction and the confidence interval.

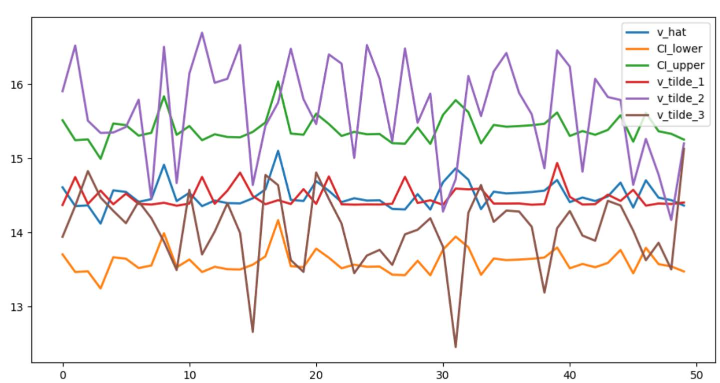

Figure 2: Neural Network Prediction.

The blue line stands for $\hat{v}$, the Monte Carlo estimation. The orange line and the green line are respectively the left and right endpoints of the confidence interval. The red, purple, and brown line correspond to the neural network prediction when $K = 20, 200, 2000$.

The mean square error for the test set is outputted below:

```
Mean squared error for K=20:  0.05379644691028611
Mean squared error for K=200:  1.8122860494687556
Mean squared error for K=2000:  0.5082348063789286
```

A Monte Carlo Python Code

```
import numpy as np
import matplotlib.pyplot as plt

x = [0,0,0,0,0,0]

x_bar_list = []
K = [20,200,2000]
beta = 0.8

k = [np.arange(1,21),np.arange(1,201),np.arange(1,2001)]

def transition(z,d):
    y = z + [90]
    for i in range(7):
        y[i] = max(y[i]-d,0)
    if y[0] > 0:
        for j in range(1,7):
            y[j] = y[j]-y[0]
        y[0] = 0
    return y[1:7]

def g(z,d):
    disposal = max(z[0]-d,0)
    refill = 90-z[5]
    sale = min(d,90)
    profit = 1*sale-0.4*refill-0.1*disposal-0.1*(90-sale-disposal)
    return profit

fig, axs = plt.subplots(2,3)

for a in range(3):
    counter = 0
    while True:
        r = np.random.poisson(17)
        x = transition(x,r)
        counter += 1
        if counter % 1000 == 0:
            x_bar_list.append(x)
        if len(x_bar_list) == K[a]:
            break

    estimate_list_100 = []
    estimate_list_1000 = []
    std_list_100 = []
    std_list_1000 = []
    for state in x_bar_list:
```

```
        x = state
        profit_list = []
        for episode in range(1000):
            profit = 0
            for T in range(50):
                r = np.random.poisson(17)
                profit = profit + (beta**T)*g(x,r)
                x = transition(x,r)
            profit_list.append(profit)
        state_profit_100 = np.mean(profit_list[:100])
        state_profit_1000 = np.mean(profit_list)
        state_std_100 = np.std(profit_list[:100])
        state_std_1000 = np.std(profit_list)
        estimate_list_100.append(state_profit_100)
        estimate_list_1000.append(state_profit_1000)
        std_list_100.append(state_std_100)
        std_list_1000.append(state_std_1000)
    std_list_100 = np.array(std_list_100)
    std_list_1000 = np.array(std_list_1000)

    axs[0,a].plot(k[a], estimate_list_100)
    axs[0,a].plot(k[a], estimate_list_100+1.96*std_list_100/10)
    axs[0,a].plot(k[a], estimate_list_100-1.96*std_list_100/10)
    axs[0,a].set_title('k='+str(K[a])+',n=100')
    axs[1,a].plot(k[a], estimate_list_1000)
    axs[1,a].plot(k[a], estimate_list_1000+1.96*std_list_1000/np.
        sqrt(1000))
    axs[1,a].plot(k[a], estimate_list_1000-1.96*std_list_1000/np.
        sqrt(1000))
    axs[1,a].set_title('k='+str(K[a])+',n=1000')


plt.show()
```

B Neural Network Python Code

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers import Dense

class MyNeuralNetwork:
    def __init__(self, input_dimension):
        # define the keras model
        self.model = Sequential()
        self.model.add(Dense(10, input_dim=input_dimension,
```

```
                                activation='relu'))
        self.model.add(Dense(6, activation='relu'))
        self.model.add(Dense(1, activation='linear'))
        # compile the keras model
        self.model.compile(loss='mean_squared_error', optimizer='
            adam')

class Scaler:
    # save mean and variance of x, y sets
    def __init__(self, x, y):
        self.x_mean = np.mean(x, axis=0)
        self.y_mean = np.mean(y, axis=0)
        self.x_std = np.std(x, axis=0)
        self.y_std = np.std(y, axis=0)


    def get_x(self):
        # return saved mean and variance of x
        return self.x_std, self.x_mean

    def get_y(self):
        # return saved mean and variance of y
        return self.y_std, self.y_mean

def transition(z,d):
    y = z + [90]
    for i in range(7):
        y[i] = max(y[i]-d,0)
    if y[0] > 0:
        for j in range(1,7):
            y[j] = y[j]-y[0]
        y[0] = 0
    return y[1:7]

def g(z,d):
    disposal = max(z[0]-d,0)
    refill = 90-z[5]
    sale = min(d,90)
    profit = 1*sale-0.4*refill-0.1*disposal-0.1*(90-sale-disposal)
    return profit

def data_generator(x,K):
    counter = 0
    x_bar_list = []
    beta = 0.8
    while True:
        r = np.random.poisson(17)
        x = transition(x,r)
```

```python
        counter += 1
        if counter == 1000:
            x_data = x
            x_bar_list.append(x)
        elif (counter % 1000 == 0) and (x not in x_bar_list):
            x_data = np.vstack([x_data,x])
            x_bar_list.append(x)
        if len(x_bar_list) == K:
            break

    estimate_list_100 = []
    std_list_100 = []
    for state in x_bar_list:
        x = state
        profit_list = []
        for episode in range(1000):
            profit = 0
            for T in range(50):
                r = np.random.poisson(17)
                profit = profit + (beta**T)*g(x,r)
                x = transition(x,r)
            profit_list.append(profit)
        state_profit_100 = np.mean(profit_list)
        state_profit_std_100 = np.std(profit_list)
        estimate_list_100.append(state_profit_100)
        std_list_100.append(state_profit_std_100)
    estimate_list_100 = np.array(estimate_list_100)
    std_list_100 = np.array(estimate_list_100)
    return x_data, estimate_list_100[:, np.newaxis], std_list_100
        [:, np.newaxis]



beta = 0.8
initial = [0,0,0,0,0,0]
x_1, y_1 = data_generator(initial,20)[:2]
x_2, y_2 = data_generator(initial,200)[:2]
x_3, y_3 = data_generator(initial,2000)[:2]
neural_network_1 = MyNeuralNetwork(input_dimension=6)
neural_network_2 = MyNeuralNetwork(input_dimension=6)
neural_network_3 = MyNeuralNetwork(input_dimension=6)


normalizer_1 = Scaler(x_1, y_1)
std_x_1, mean_x_1 = normalizer_1.get_x()
x_1_norm = (x_1 - mean_x_1) / (std_x_1+0.000001)
std_y_1, mean_y_1 = normalizer_1.get_y()
y_1_norm = (y_1 - mean_y_1) / (std_y_1+0.000001)
```

```

neural_network_1.model.fit(x_1_norm, y_1_norm, epochs=100, batch_size=8)


normalizer_2 = Scaler(x_2, y_2)
std_x_2, mean_x_2 = normalizer_2.get_x()
x_2_norm = (x_2 - mean_x_2) / (std_x_2+0.000001)
std_y_2, mean_y_2 = normalizer_2.get_y()
y_2_norm = (y_2 - mean_y_2) / (std_y_2+0.000001)

neural_network_2.model.fit(x_2_norm, y_2_norm, epochs=100, batch_size=8)


normalizer_3 = Scaler(x_3, y_3)
std_x_3, mean_x_3 = normalizer_3.get_x()
x_3_norm = (x_3 - mean_x_3) / (std_x_3+0.000001)
std_y_3, mean_y_3 = normalizer_3.get_y()
y_3_norm = (y_3 - mean_y_3) / (std_y_3+0.000001)

neural_network_3.model.fit(x_3_norm, y_3_norm, epochs=100, batch_size=8)


x_0 = np.array([10,20,30,40,50,80])
x_prime = np.vstack([x_0, x_0])
x_0_norm_1 = (x_prime - mean_x_1) / (std_x_1+0.000001)
x_0_norm_2 = (x_prime - mean_x_2) / (std_x_2+0.000001)
x_0_norm_3= (x_prime - mean_x_3) / (std_x_3+0.000001)
v_tilde_norm_1 = neural_network_1.model.predict(x_0_norm_1)
v_prime = v_tilde_norm_1 * std_y_1 + mean_y_1
v_tilde_1 = v_prime[0]
v_tilde_norm_2 = neural_network_2.model.predict(x_0_norm_2)
v_prime = v_tilde_norm_2 * std_y_2 + mean_y_2
v_tilde_2 = v_prime[0]
v_tilde_norm_3 = neural_network_3.model.predict(x_0_norm_3)
v_prime = v_tilde_norm_3 * std_y_3 + mean_y_3
v_tilde_3 = v_prime[0]


profit_list = []
for episode in range(1000):
    profit = 0
    x = x_0.tolist()
    for T in range(50):
        r = np.random.poisson(17)
        profit = profit + (beta**T)*g(x,r)
```

```
        x = transition(x,r)
    profit_list.append(profit)

v_hat = np.mean(profit_list)
v_sigma = np.std(profit_list)
v_CI_lower = v_hat-1.96*v_sigma/np.sqrt(1000)
v_CI_upper = v_hat+1.96*v_sigma/np.sqrt(1000)


print("v_tilde_1:",v_tilde_1)
print("v_tilde_2:",v_tilde_2)
print("v_tilde_3:",v_tilde_3)
print("v_hat:",v_hat)
print("CI(95%):["+str(v_CI_lower)+","+str(v_CI_upper)+"]")

x_test, y_test, std_test = data_generator([0,0,0,0,0,0],50)
x_test_norm_1 = (x_test - mean_x_1) / (std_x_1+0.000001)
x_test_norm_2 = (x_test - mean_x_2) / (std_x_2+0.000001)
x_test_norm_3 = (x_test - mean_x_3) / (std_x_3+0.000001)
y_pred_1 = neural_network_1.model.predict(x_test) * std_y_1 + mean_y_1
y_pred_2 = neural_network_2.model.predict(x_test) * std_y_2 + mean_y_2
y_pred_3 = neural_network_3.model.predict(x_test) * std_y_3 + mean_y_3

mse_1 = mean_squared_error(y_test, y_pred_1)
mse_2 = mean_squared_error(y_test, y_pred_2)
mse_3 = mean_squared_error(y_test, y_pred_3)
print('Mean␣squared␣error␣for␣K=20:␣', mse_1)
print('Mean␣squared␣error␣for␣K=200:␣', mse_2)
print('Mean␣squared␣error␣for␣K=2000:␣', mse_3)


plt.plot(y_test, label="v_hat", lw=2)
plt.plot(y_test-1.96*std_test/np.sqrt(1000), label="CI_lower", lw=2)
plt.plot(y_test+1.96*std_test/np.sqrt(1000), label="CI_upper", lw=2)
plt.plot(y_pred_1, label="v_tilde_1", lw=2)
plt.plot(y_pred_2, label="v_tilde_2", lw=2)
plt.plot(y_pred_3, label="v_tilde_3", lw=2)
plt.legend()
plt.show()
```