MA4260 Stochastic Operations Research CA

Wang Jiangyi
National University of Singapore

1 Time-Based Simulation

In this question, we focus on the inventory of base ingredient B (equivalent to equipment A). We want to pick the optimal quantity q^*. Actually, I have 2 plans, which can be discribed as follows:

Take **p=20 case** as an example. In this example, firstly, observe that **when q is big enough**, i.e., $q > 100$, the total cost per week will be very large. Therefore, we can ignore these bad q and our candidate for optimal quantity q^* is [1, 2, ..., 100]

Plan A: We simulate for 100 times, and for each time, we do following things: total weeks are 10,000 and calculate the **Total Cost Per Week** (**denote as** $TCPW$) for each candidate q, denote as $T_{i,q}$ (i represents the i-th time). Then, we calculate the mean of $T_{i,q}$ with respect to i and denote as T_q. Finally, we find the optimal quantity q^* by minimizing $\{T_q\}_{q=1}^{100}$.

Plan B: We only simulate 1 time. We set total weeks to 1,000,000 and calculate the total cost per week for each candidate. Then, we find the optimal one.

Notice that for both 2 plans, we **all need the similar scale of computation**, which means it is reasonable to make comparison between the 2 plans. We can compare the 2 plans as follows:

1.1 p=20 Case

1.1.1 Plan A

Firstly, we consider **Plan A**. Basically, the simplest part is just to visualize the Mean Curve of 100 repeats for each quantity. Then, according to the **Mean Total Cost Per Week for each quantity**, we can find the smallest one (**Mean TCPW**) and determine the optimal quantity q^* as the corresponding quantity. This process is shown in Figure 1:

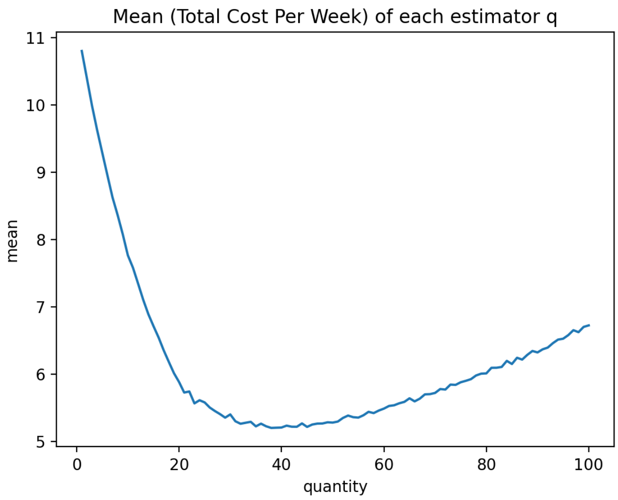

Figure 1: Mean Curve for each quantity for Plan A

In Figure 1, it can be observed that, when **q=38**, the Mean TCPW attains its minimum **5.199**. Therefore, the optimal quantity $q^* = 38$.

Discussion Part:

To be more precise, actually, in this plan, we find one estimator $TCPW(q) = \frac{\sum_{i=1}^{100} TCPW(i,q)}{100}$ for each choice of q, and the estimator (of optimal quantity) q^* can be viewed as inducing by estimators $\{TCPW(q)\}_{q=1}^{100}$. Therefore, **the quality of estimators $\{TCPW(q)\}_{q=1}^{100}$ directly influences the quality of estimator** q^*. Since we have 100 repeated times, we can check the property of $\{TCPW(q)\}_{q=1}^{100}$ (i.e., **variance and histogram**) to show our **confidence** on $TCPW(q^*)$. This is a good way to show that, **my solution q^* is stable and quite close to the actual mean.**

The following part is the visualization process:

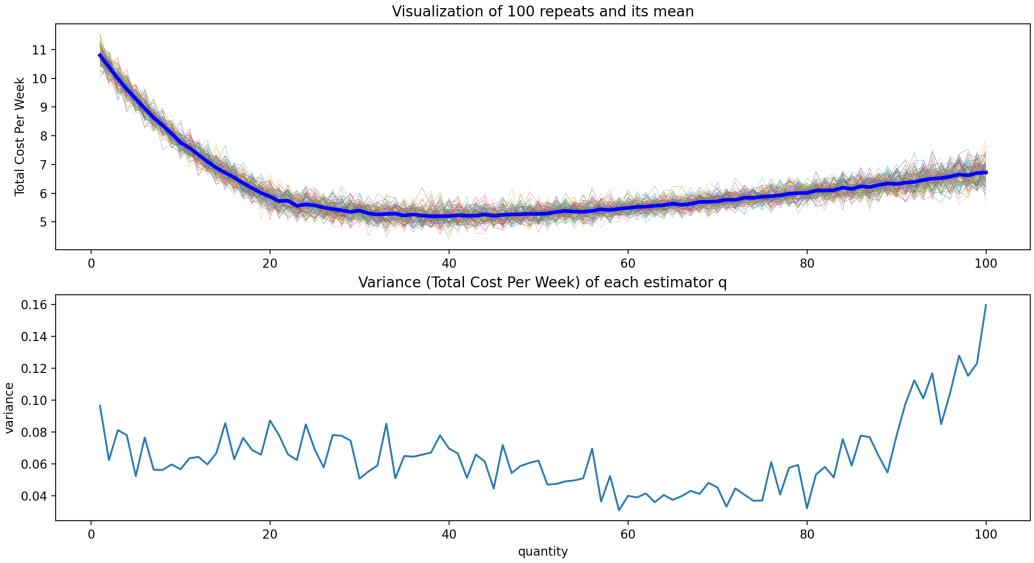

Figure 2: 100 repeated TCPW and corresponding Variance for each quantity

In the first subfigure of Figure 2, those **multiple slender lines** represent those 100 repeated times, and the **blue stout lin**e represents its mean. In the second subfigure of Figure 2, it is the corresponding **variance of 100 repeated experiments** for each quantity.

Since we know that our optimal quantity $q^* = 38$, then we can check the $TCPW(i,q)$ for 100 repeated experiements (i=1, 2, ..., 100) when $q = 38$. The histogram is shown as follows:

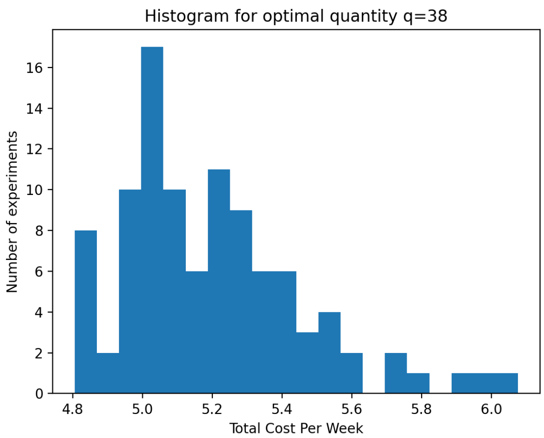

Figure 3: Histogram of $TCPW(i, 38)$ for i=1, 2, ..., 100

In Figure 3, we can see that, $TCPW(i, 38)$ are gathered around the $TCPW(38) = 5.199$. In real life, the process is just one realization of such experiment. Therefore, we **have much confidence** that our estimator $q^* = 38$ and $TCPW(38) = 5.199$ is accurate. This is also can be deduced by **variance of** $q* = 38$ **is around 0.07, which is very small.**

To conclude, in **Plan A**, we have much confidence our solution q^* is acutally **a rather good solution** with respect to **histogram and variance**.

1.1.2 Plan B

In this Plan, we are only able to give the TCPW Curve since we only have 1 repeated experiment. The curve is given as follows:

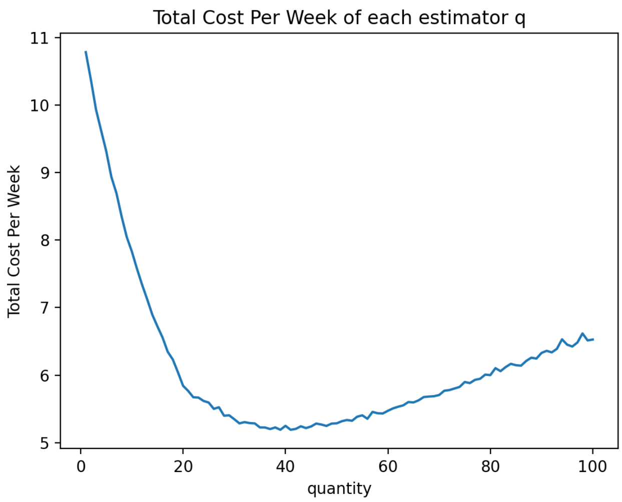

Figure 4: TCPW Curve for each quantity for Plan B

In Figure 4, it can be observed that, when **q=41**, the Mean TCPW attains its minimum **5.191**. Therefore, the optimal quantity $q^* = 41$.

Although these quantities are quite close to our previous result ($TCPW(38) = 5.199$ **and** $q^* = 38$), here, we have no evidence to say that our estimator is good. That is because, we only have 1 experiment here. Although this experiment contains 1,000,000 weeks, it is difficult for us to claim this is a good estimator **without the help of statistics quantity like variance and histogram**.

1.1.3 Disccusion

From the results of **Plan A** and **Plan B**, it is obvious that, with **Plan A**, we can **attain more insights** about our estimators naturally. That is to say, we **have more evidence** to show that our solution is good. Therefore, in the following cases (p=4 and p=10), we prefer **Plan A** to show our results. **Since the explanation part is quite similar, we omit the similar part and only give the necessary explanation.**

1.2 p=4 Case

When **p=4**,our candidate for optimal quantity q^* is [1, 2, ..., 30]. The Mean TCPW Curve for each quantity can be shown as follows:

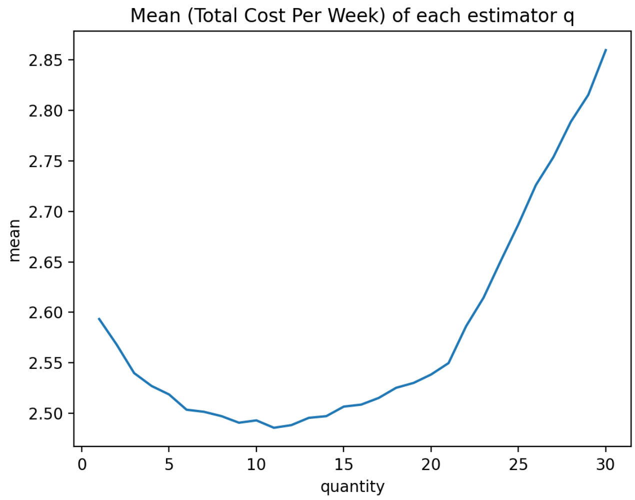

Figure 5: Mean TCPW Curve for each quantity

In Figure 5, it can be observed that, when **q=11**, the Mean TCPW attains its minimum **2.485**. Therefore, the optimal quantity $q^* = 11$.

The visualization of 100 repeated experiments and its variance is shown in Figure 6:

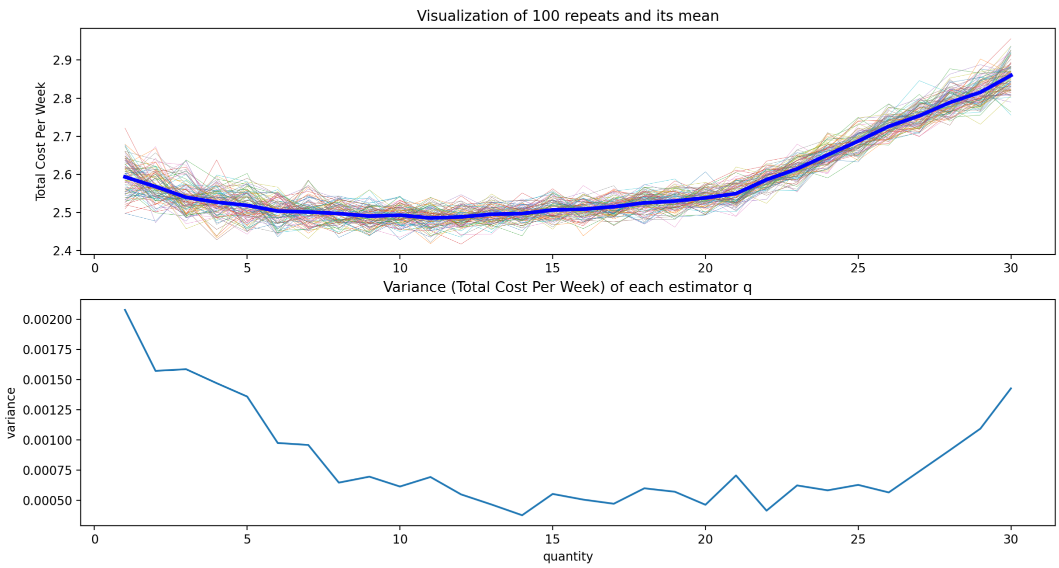

Figure 6: 100 repeated TCPW and corresponding Variance for each quantity

From Figure 6, it can be observed that, when $q^* = 11$, the corresponding **variance for 100 experiments is around 0.0075**, which is extremely small. This is the evidence that **our estimator is very stable**.

The histogram of 100 experiments when $q^* = 11$ is:

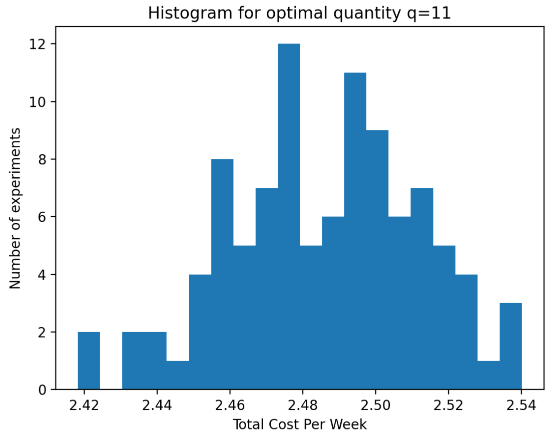

Figure 7: Histogram of $TCPW(i, 11)$ for i=1, 2, ..., 100

From Figure 7, it can be observed that $TCPW(i, 11)$ are gathered around $TCPW(11) = 2.485$. This also **gives us more confidence** that, when one realization of such expermient happens in real life, **it will also be closely to** $TCPW(11) = 2.485$. Therefore, our estimator $q^* = 11$ is accurate in this sence.

1.3 p=10 Case

When **p=10**,our candidate for optimal quantity q^* is [1, 2, ..., 50]. The Mean TCPW Curve for each quantity can be shown as follows:

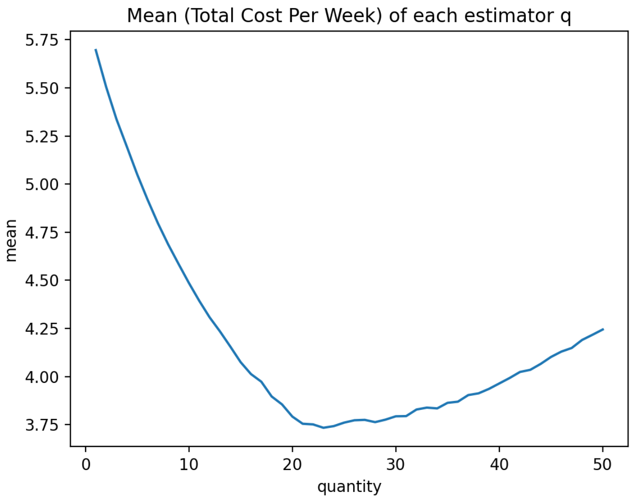

Figure 8: Mean TCPW Curve for each quantity

In Figure 8, it can be observed that, when **q=23**, the Mean TCPW attains its minimum **3.733**. Therefore, the optimal quantity $q^* = 23$.

The visualization of 100 repeated experiments and its variance is shown in Figure 9:

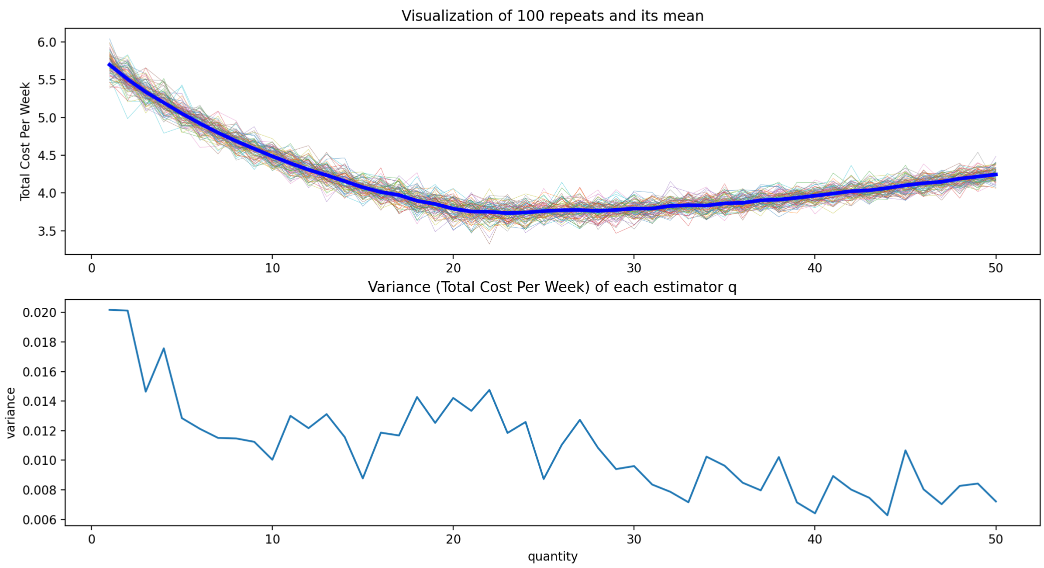

Figure 9: 100 repeated TCPW and corresponding Variance for each quantity

From Figure 9, it can be observed that, when $q^* = 23$, the corresponding **variance for 100 experiments is around 0.012**, which is extremely small. This is the evidence that **our estimator is very stable**.

The histogram of 100 experiments when $q^* = 23$ is:

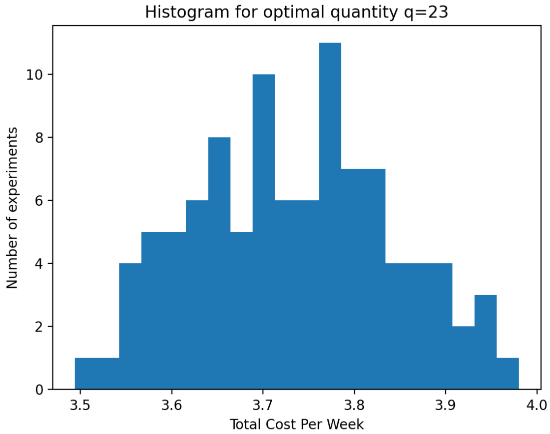

Figure 10: Histogram of $TCPW(i, 23)$ for i=1, 2, ..., 100

From Figure 10, it can be observed that $TCPW(i, 23)$ are gathered around $TCPW(23) = 3.733$. This also **gives us more confidence** that, when one realization of such expermient happens in real life, **it will also be closely to** $TCPW(23) = 3.733$. Therefore, our estimator $q^* = 23$ is accurate in this sence.

2 Event-Based Simulation

2.1 Waiting Time

Here, we give **2 figures** about the distribution of Waiting Time for a random passengers, one is the **histogram** and the other is **kernel density estimation (KDE)**:

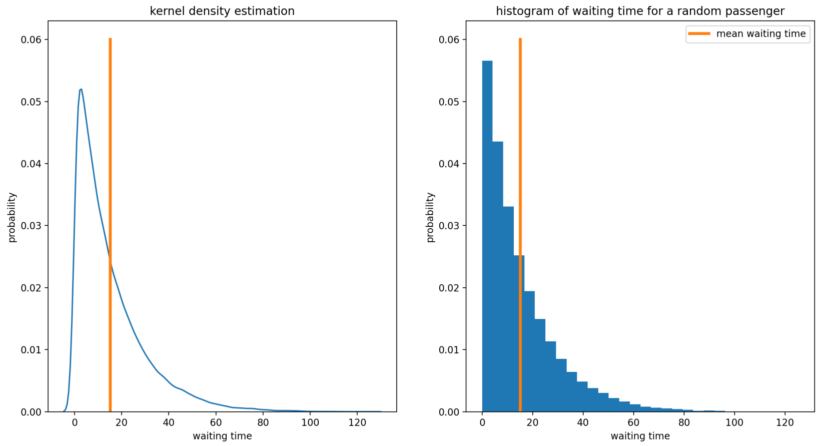

Figure 11: KDE and Histogram of waiting time distribution

Mean Waiting Time is around **15.20**.

2.2 Number of Passengers in One Bus

Here, we also give **2 figures** about the distribution of Number of Passengers in One Bus, one is the **histogram** and the other is **kernel density estimation (KDE)**:

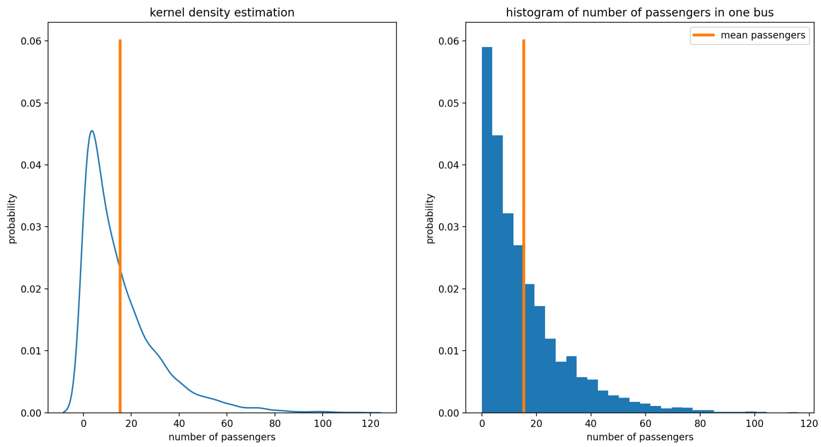

Figure 12: KDE and Histogram of number of passengers distribution

Mean Number of Passengers in One Bus is around **15.40**.

2.3 Zero-Passenger Bus

Notice that the sojourn time for two passengers $S_i \sim exp(1)$ for i = 1, 2, ..., 100,000, and the sojourn time for two buses $\tilde{S}_j \sim exp(\frac{1}{15})$ for j = 1, 2, ..., 10,000. Therefore, if we denote the occurence time for 100,000-th passenger as $W_{100,000}$ and that for 10,000-th bus as $\tilde{W}_{10,000}$, then we have:

$$\begin{aligned} \mathbb{E}[W_{100,000}] = 100,000 * \mathbb{E}[S_i] = 100,000 \\ \mathbb{E}[\tilde{W}_{10,000}] = 10,000 * \mathbb{E}[\tilde{S}_j] = 150,000 \end{aligned} \tag{1}$$

Therefore, for almost all cases, there exists Zero-Passenger Buses.

Disccusion:

In our previous results (especially in **Number of Passengers in One Bus**), actually we **kick out** Zero-Passenger Buses. That is because, **our aim is to estimate the distribution of Passengers in Bus**.

In real life, everything happens according to the same reference of time. That is to say, in order to achieve our aim, we should make our estimation until $W_{100,000}$ (the occurence time of 100,000-th passenger) since we know that the occurence time of the last passenger will be much earlier than that of the last bus with almost 1 probability. Afterwards, there are supposed to have more passengers whose sojourn time still satisfy $exp(1)$. However, we do not have such passengers here just **for the limit of Event-Based Simulation**.

Therefore, to compensate for this, we should **kick out those Afterwards Zero-Passenger Buses**. In our experiment, there are **6495 buses which are effective** (total is 10,000 buses). Only by this, we can attain the relatively accurate estimation the distribution of Passengers in One Bus.

3 Fitting Stochastic Model

Here, according to the question, we are free to choose the number of bins k. Here, we choose 20 candidates of k, i.e., [3, 4, ..., 22]. Also, we choose the partiion $(\beta_{i-1}, \beta_i]$ by **letting random varaible have equal probability being each partition**.

Therefore, we can determine $\{\beta_i\}_{i=0}^{k-1}$ by solving:

$$\begin{aligned} \int_0^{\beta_i} \hat{\lambda} e^{-\hat{\lambda} t} dt = \frac{i}{k} \\ \rightarrow \beta_i = -\frac{ln(1-\frac{i}{k})}{\hat{\lambda}} \end{aligned} \tag{2}$$

Then, we can calculate $\chi^2(k)$ for $k \in [3, 4, ..., 22]$ and compare with $\chi^2_{k-2}(\alpha)$ where $\alpha = 0.5$. The result of testing is as follows:

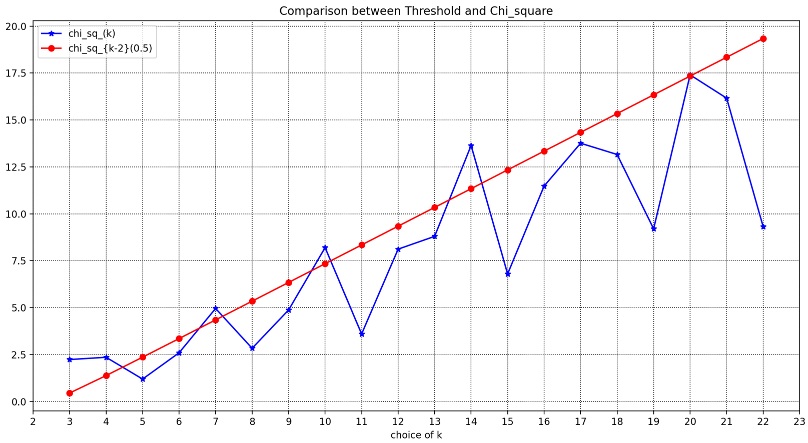

Figure 13: χ^2 test for different k

In Figure 13, red line represents $\chi^2_{k-2}(0.5)$ and blue line represents $\chi^2(k)$.

Explanation:

Take $k = 11$ as an example, since the blue dot ($\chi^2(11)$) is **less than** the red dot ($\chi^2_9(0.5)$). Therefore, **if we choose 11 bins**, then we tend to accept that our observation comes from exponential distribution.

Observation:

1. This test is sensitive to the choice of k value.

2. On average, with different choices of k, we accept our hypothesis H_0 more frequently (**15 times**). Therefore, we have more confidence to say that our observation comes from exponential distribution.

4 Stochastic Programming

The extensive form of this stochastic programming:

$$
\begin{aligned}
min\ \ & 150x_{wh} + 230x_{co} + 260x_{su} \\
& - 0.5(170w_{wh,ab} - 238y_{wh,ab} + 150w_{co,ab} - 210y_{co,ab} + 36w_{s.fav,ab} + 10w_{s.unf,ab}) \\
& - 0(170w_{wh,av} - 238y_{wh,av} + 150w_{co,av} - 210y_{co,av} + 36w_{s.fav,av} + 10w_{s.unf,av}) \\
& - 0.5(170w_{wh,be} - 238y_{wh,be} + 150w_{co,be} - 210y_{co,be} + 36w_{s.fav,be} + 10w_{s.unf,be})
\end{aligned}
$$

$$
\begin{aligned}
s.t.\ \ & x_{wh} + x_{co} + x_{su} \leq 500 \\
& 3x_{wh} + y_{wh,ab} - w_{wh,ab} \geq 200 \quad 3.6x_{co} + y_{co,ab} + w_{co,ab} \geq 240 \\
& w_{s.fav,ab} + w_{s.unf,ab} = 24x_{su} \quad w_{s.fav,ab} \leq 6000 \\
& 2.5x_{wh} + y_{wh,av} - w_{wh,av} \geq 200 \quad 3x_{co} + y_{co,av} + w_{co,av} \geq 240 \\
& w_{s.fav,av} + w_{s.unf,av} = 20x_{su} \quad w_{s.fav,av} \leq 6000 \\
& 2x_{wh} + y_{wh,be} - w_{wh,be} \geq 200 \quad 2.4x_{co} + y_{co,be} + w_{co,be} \geq 240 \\
& w_{s.fav,be} + w_{s.unf,be} = 16x_{su} \quad w_{s.fav,be} \leq 6000
\end{aligned}
$$

The solution of this optimization problem is:

$$
\begin{aligned}
& x^*_{wh} = 150 \\
& x^*_{co} = 100 \\
& x^*_{su} = 250 \\
& w^*_{wh,ab} = 250 \quad y^*_{wh,ab} = 0 \quad w^*_{co,ab} = 120 \quad y^*_{co,ab} = 0 \quad w^*_{s.fav,ab} = 6000 \quad w^*_{s.unf,ab} = 0 \\
& w^*_{wh,av} = 0 \quad y^*_{wh,av} = 0 \quad w^*_{co,av} = 0 \quad y^*_{co,av} = 0 \quad w^*_{s.fav,av} = 0 \quad w^*_{s.unf,av} = 0 \\
& w^*_{wh,be} = 100 \quad y^*_{wh,be} = 0 \quad w^*_{co,be} = 0 \quad y^*_{co,be} = 0 \quad w^*_{s.fav,be} = 4000 \quad w^*_{s.unf,be} = 0
\end{aligned}
$$

Discussion:

The reason that **all second-stage variables for average case are 0** is, we assign 0 probabilty to average case. Therefore, all kinds of value for average-case second-stage variables will not make any difference to this optimization problem.

5 Appendix

5.1 Q1-Code

5.1.1 Main function A

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

def TCPW(q, p, c, h, week):
# initialization
x_inv = q;  # inventory level
x_dis = 0;  # remaining failure days
T = 0;  # total cost
PURCHASEPERMITTED = True;

# update process
for iter in range(week):
if x_dis > 0:
x_dis -= 1;
PURCHASEPERMITTED = False;
else:
PURCHASEPERMITTED = True;

if x_inv > 0:
x_inv -= 1;
else:
T += p;

if PURCHASEPERMITTED == True:
u = random.uniform(0, 1);
if u < 0.95:
if x_inv <= q - 2:
x_inv += 2;
T += 2 * c;
elif x_inv == q - 1:
x_inv += 1;
T += c;
else:
x_dis = 19;
T += h * x_inv
return T/week;

# factory produce equipment Model
# aim: determine the optimal inventory quantity threshold q

#-------------------------------------------------------------------------
#-------------------------------------------------------------------------

# total repeat times
repeat = 100;
# total weeks
week = 10000;
# threshold inventory quantity
Q = [i+1 for i in range(50)];
# parameter
p = 10;
c = 1;
h = 0.1;
# record for average cost
record_aver_each = np.zeros((repeat, len(Q)));


for rep in range(repeat):
count = 0;
for q in Q:
aver_T = TCPW(q, p, c, h, week);
record_aver_each[rep, count] = aver_T;
count += 1;
```

```
# calculate mean, variance
record_mean = np.mean(record_aver_each, axis = 0); # calculate the mean for each column
record_variance = np.var(record_aver_each, axis = 0); # calculate the variance for each
                                                column

idx_opt = np.argmin(record_mean);
quantity_opt = Q[idx_opt];
tcpw_opt = np.min(record_mean)

print('the optimal quantity is: ', quantity_opt);
print('the minimum average cost (per week) is: ', tcpw_opt);

# visualization of record_mean
fig = plt.figure(1);
ax = fig.add_subplot(1,1,1);
ax.plot(Q, record_mean);
plt.title('Mean (Total Cost Per Week) of each estimator q');
plt.xlabel('quantity');
plt.ylabel('mean')
plt.show()

# visualization of record_variance
fig = plt.figure(2);
ax = fig.add_subplot(2,1,2);
ax.plot(Q, record_variance);
plt.title('Variance (Total Cost Per Week) of each estimator q');
plt.xlabel('quantity');
plt.ylabel('variance')
plt.show()


# visualization of all 100 repeats (10000 weeks each)
ax = fig.add_subplot(2,1,1);
for rep in range(repeat):
ax.plot(Q, record_aver_each[rep,:], linewidth = .2);
ax.plot(Q, record_mean, linewidth = 3, color='b');
plt.title('Visualization of 100 repeats and its mean');
plt.ylabel('Total Cost Per Week')
plt.show()

# visualization of optimal choice of quantity q^{*}
optimal_record = record_aver_each[:, idx_opt];
plt.figure(4);
plt.title('Histogram for optimal quantity q=23');
plt.xlabel('Total Cost Per Week');
plt.ylabel('Number of experiments')
plt.hist(optimal_record, bins=20);
```

5.1.2 Main funciton B

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

def TCPW(q, p, c, h, week):
# initialization
x_inv = q;  # inventory level
x_dis = 0;  # remaining failure days
T = 0;  # total cost
PURCHASEPERMITTED = True;

# update process
for iter in range(week):
if x_dis > 0:
x_dis -= 1;
PURCHASEPERMITTED = False;
else:
PURCHASEPERMITTED = True;

if x_inv > 0:
x_inv -= 1;
else:
T += p;

if PURCHASEPERMITTED == True:
u = random.uniform(0, 1);
if u < 0.95:
if x_inv <= q - 2:
x_inv += 2;
T += 2 * c;
elif x_inv == q - 1:
x_inv += 1;
T += c;
else:
x_dis = 19;
T += h * x_inv
return T/week;
# factory produce equipment Model
# aim: determine the optimal inventory quantity threshold q

#---------------------------------------------------------------------------
#---------------------------------------------------------------------------

# total weeks
week = 1000000;
# threshold inventory quantity
Q = [i+1 for i in range(100)];
# parameter
p = 20;
c = 1;
h = 0.1;
# record for average cost
record_aver = [0] * len(Q);

count = 0;

for q in Q:
aver_T = TCPW(q, p, c, h, week);
record_aver[count] = aver_T;
count += 1;

idx_opt = record_aver.index(min(record_aver));
quantity_opt = Q[idx_opt];
print('the optimal quantity is: ', quantity_opt);
print('the minimum average cost (per week) is: ', min(record_aver));

fig = plt.figure();
ax = fig.add_subplot(1,1,1);
```

```
ax.plot(Q, record_aver);
plt.title('Total Cost Per Week of each estimator q');
plt.xlabel('quantity');
plt.ylabel('Total Cost Per Week')
plt.show()
```

5.1.3 TCPW(Total Cost Per Week)

```
# Total cost per week
def TCPW(q, p, c, h, week):
# initialization
x_inv = q;  # inventory level
x_dis = 0;  # remaining failure days
T = 0;  # total cost
PURCHASEPERMITTED = True;

# update process
for iter in range(week):
if x_dis > 0:
x_dis -= 1;
PURCHASEPERMITTED = False;
else:
PURCHASEPERMITTED = True;

if x_inv > 0:
x_inv -= 1;
else:
T += p;

if PURCHASEPERMITTED == True:
u = random.uniform(0, 1);
if u < 0.95:
if x_inv <= q - 2:
x_inv += 2;
T += 2 * c;
elif x_inv == q - 1:
x_inv += 1;
T += c;
else:
x_dis = 19;
T += h * x_inv
return T/week;
```

5.2 Q2-Code

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
# Passenger and Bus Model

# PASSENGER
scale1 = 1; # the reciprocal of lambda
num1 = 100000;
x_passenger = np.random.exponential(scale1, num1);
y_passenger = [0] * num1;
y_passenger[0] = x_passenger[0];
for idx in range(1, num1):
y_passenger[idx] = x_passenger[idx] + y_passenger[idx-1];

# BUS
scale2 = 15;
num2 = 10000;
x_bus = np.random.exponential(scale2, num2);
y_bus = [0] * num2;
y_bus[0] = x_bus[0];
for idx in range(1, num2):
y_bus[idx] = x_bus[idx] + y_bus[idx-1];

# T_waiting: Waiting time for k-th Passenger (k=1,2,...,100000)
# NumOfPeople_bus: Number of people for i-th Bus (i=1,2,...,10000)
T_waiting = [0] * num1;
NumOfPeople_bus = [0] * num2;

start_time = 0;
start_passenger = 0;

for bus_idx in range(num2):
# we want to calculate the number of passengers in the interval ( start_time, end_time ];
end_time = y_bus[bus_idx];
count = 0;
for i in range(start_passenger, num1):
if y_passenger[i] > end_time:
break;
elif start_time < y_passenger[i] <= end_time:
count += 1;

T_waiting[start_passenger : i] = [end_time - passenger_time for passenger_time in
                                   y_passenger[start_passenger : i]];
NumOfPeople_bus[bus_idx] = count;

start_passenger = i;
start_time = end_time;

# Since In those buses, it may happen that starting from one car, all cars afterwards do
                                   not have any passengers.
# Therefore, we should kick out those cars to attain the TRUE distribution for number of
                                   passengers in one bus

for idx in range(num2):
if sum(NumOfPeople_bus[:idx])==100000:
idx_0 = idx;
break;

NumOfPeople_bus_wonull = NumOfPeople_bus[:idx_0];
mean_bus = np.mean(NumOfPeople_bus_wonull);
mean_T = np.mean(T_waiting);

var_bus = np.var(NumOfPeople_bus_wonull);
var_T = np.var(T_waiting);

plt.figure(1)
plt.subplot(1,2,1);
sns.kdeplot(T_waiting)
```

```
plt.plot([mean_T, mean_T], [0, 0.06], linewidth=3);
plt.xlabel('waiting time')
plt.ylabel('probability')
plt.title('kernel density estimation')


plt.subplot(1,2,2)
plt.hist(T_waiting, bins=30, density=True)
plt.xlabel('waiting time')
plt.ylabel('probability')
plt.title('histogram of waiting time for a random passenger')
plt.plot([mean_T, mean_T], [0, 0.06], linewidth=3);
plt.legend(['mean waiting time'])

plt.figure(2)
plt.subplot(1,2,1);
sns.kdeplot(NumOfPeople_bus_wonull)
plt.xlabel('number of passengers')
plt.ylabel('probability')
plt.title('kernel density estimation')
plt.plot([mean_bus, mean_bus], [0, 0.06], linewidth=3);

plt.subplot(1,2,2)
plt.hist(NumOfPeople_bus_wonull, bins=30, density=True)
plt.xlabel('number of passengers')
plt.ylabel('probability')
plt.title('histogram of number of passengers in one bus')
plt.plot([mean_bus, mean_bus], [0, 0.06], linewidth=3);
plt.legend(['mean passengers'])
plt.show()


plt.show()
```

5.3 Q3-Code

```
import math
import matplotlib.pyplot as plt

def exp_test(t, k, alpha = 0.5):
# t: test data
# k: number of bins
# alhpa: the threshold

num = len(t);
estimator_lambda = num / sum(t);

# construct the bins for equal probability
beta = [0] * k;
for i in range(k):
beta[i] = - math.log(1 - i / k) / estimator_lambda

O = [0] * k;
for test_data in t:
for i in range(k-1):
if beta[i] <= test_data < beta[i+1]:
O[i] += 1;
break;

O[k-1] = num - sum(O);
chi_sq = sum([((i - num / k)**2) / (num / k) for i in O]);
return chi_sq

t = [0.01, 0.07, 0.03, 0.08, 0.04,
0.10, 0.05, 0.10, 0.11, 0.17,
1.50, 0.93, 0.54, 0.19, 0.22,
0.36, 0.27, 0.46, 0.51, 0.11,
0.56, 0.72, 0.29, 0.04, 0.73];

# we choose k from [3, 4, 5, ..., 22];
K = list(range(3,23));
chi_sq_record = [0] * len(K);

count = 0;

for k in K:
chi_sq = exp_test(t, k);
chi_sq_record[count] = chi_sq;
count += 1;

print(chi_sq_record);

# since we choose alpha = 0.5
# the threshold is X_{k-2}^2(0.5)
thresh_chi_sq = [0.455, 1.386, 2.366, 3.357, 4.351,
5.348, 6.346, 7.344, 8.343, 9.342,
10.341, 11.340, 12.340, 13.339, 14.339,
15.338, 16.338, 17.338, 18.338, 19.337];
plt.plot(K, chi_sq_record, '-*b');
plt.plot(K, thresh_chi_sq, '-or');
plt.legend(['chi_sq_(k)','chi_sq_{k-2}(0.5)'])
plt.xlabel('choice of k')
plt.xlim(2,23)
new_ticks = list(range(2,24));
plt.xticks(new_ticks)
plt.grid(linestyle=":", color="k")
plt.title('Comparison between Threshold and Chi_square')
plt.show();
```

5.4 Q4-Code

Here, I use **colab** to run my code.

```
# -*- coding: utf-8 -*-
"""q4-4260.ipynb

Automatically generated by Colaboratory.

Original file is located at
https://colab.research.google.com/drive/1k1SOnfsMxdCh3WtJ322efAhVFANxbmsA
"""

# Commented out IPython magic to ensure Python compatibility.
# %pip install -i https://pypi.gurobi.com gurobipy

import gurobipy as gp
from gurobipy import GRB

farm = ['wheat', 'corn', 'sugarbeets']
sell = ['wheat_bad', 'corn_bad', 'sugarbeets_highprice_bad', 'sugarbeets_lowprice_bad',
'wheat_aver', 'corn_aver', 'sugarbeets_highprice_aver', 'sugarbeets_lowprice_aver',
'wheat_good', 'corn_good', 'sugarbeets_highprice_good', 'sugarbeets_lowprice_good']
buy = ['wheat_bad', 'corn_bad',
'wheat_aver', 'corn_aver',
'wheat_good', 'corn_good']

cropyield = {'wheat_aver': 2.5, 'corn_aver': 3.0, 'sugarbeets_aver': 20.0,
  'wheat_good': 3, 'corn_good': 3.6, 'sugarbeets_good': 24.0,
  'wheat_bad': 2, 'corn_bad': 2.4, 'sugarbeets_bad': 16.0}

cropcost = {'wheat': 150, 'corn': 230, 'sugarbeets': 260}
cropconstraint = { 'wheat_bad': 200, 'wheat_aver': 200, 'wheat_good': 200,
  'corn_bad': 240, 'corn_aver': 240, 'corn_good': 240}

cropsellprice = {'wheat_bad' : 170, 'wheat_aver' : 170, 'wheat_good' : 170,
  'corn_bad' : 150, 'corn_aver' : 150, 'corn_good' : 150,
  'sugarbeets_highprice_bad': 36, 'sugarbeets_highprice_aver': 36, '
                                             sugarbeets_highprice_good': 36,
  'sugarbeets_lowprice_bad': 10, 'sugarbeets_lowprice_aver': 10, 'sugarbeets_lowprice_good'
                                             : 10}
cropbuyprice = {'wheat_bad' : 238, 'wheat_aver' : 238, 'wheat_good' : 238,
  'corn_bad' : 210, 'corn_aver' : 210, 'corn_good' : 210}

bad = 0.5
aver = 0
good = 0.5

totalland = 500
maxhighbeets = 6000

model = gp.Model('StochasticProgram')

# Variables
landvar = model.addVars(farm, name="landvar")
sellvar = model.addVars(sell, name="sellvar")
buyvar = model.addVars(buy, name="buyvar")

# Capacity constraint.
model.addConstrs( (landvar[f] >= 0 for f in farm) )
model.addConstrs( (sellvar[f] >= 0 for f in sell) )
model.addConstrs( (buyvar[f] >= 0 for f in buy) )
model.addConstrs( (landvar['wheat']*cropyield[f]+buyvar[f]-sellvar[f]>=cropconstraint[f]
                                             for f in buy[0:6:2]) )
model.addConstrs( (landvar['corn']*cropyield[f]+buyvar[f]-sellvar[f]>=cropconstraint[f] for
                                              f in buy[1:6:2]) )

model.addConstr(sellvar['sugarbeets_highprice_bad']+sellvar['sugarbeets_lowprice_bad']-
                                             landvar['sugarbeets']*cropyield['
                                             sugarbeets_bad']<=0)
model.addConstr(sellvar['sugarbeets_highprice_aver']+sellvar['sugarbeets_lowprice_aver']-
```

```
                                             landvar['sugarbeets']*cropyield['
                                             sugarbeets_aver']<=0)
model.addConstr(sellvar['sugarbeets_highprice_good']+sellvar['sugarbeets_lowprice_good']-
                                             landvar['sugarbeets']*cropyield['
                                             sugarbeets_good']<=0)
model.addConstr(sellvar['sugarbeets_highprice_bad']<= maxhighbeets)
model.addConstr(sellvar['sugarbeets_highprice_aver']<= maxhighbeets)
model.addConstr(sellvar['sugarbeets_highprice_good']<= maxhighbeets)
model.addConstr(gp.quicksum(landvar[f] for f in farm) <= totalland)

obj = gp.quicksum(cropcost[f]*landvar[f] for f in farm) - bad*gp.quicksum(cropsellprice[f]*
                                             sellvar[f] for f in sell[0:4]) - aver*gp.
                                             quicksum(cropsellprice[f]*sellvar[f] for f in
                                             sell[4:8]) - good*gp.quicksum(cropsellprice[f]
                                             *sellvar[f] for f in sell[8:12]) + bad*gp.
                                             quicksum(cropbuyprice[f]*buyvar[f] for f in
                                             buy[0:2]) + aver*gp.quicksum(cropbuyprice[f]*
                                             buyvar[f] for f in buy[2:4]) + good*gp.
                                             quicksum(cropbuyprice[f]*buyvar[f] for f in
                                             buy[4:6])
model.setObjective(obj, GRB.MINIMIZE)

# Verify model formulation

model.write('StochasticProgram.lp')

# Run optimization engine

model.optimize()

print(landvar)

print(sellvar)

print(buyvar)
```