Answer – 1

1.1.a Plug-in estimate of $\boldsymbol{\theta}$

Given,

Y = New – Old
Z = Old – Placebo

$$\theta = \frac{\mathrm{E}(Y)}{\mathrm{E}(Z)}$$

To find,
$\hat{\theta} = \bar{Y} / \bar{Z}$

The plugin estimates of θ : -0.0713

1.1.b Bootstrapping & Confidence Interval

To obtain the 95% confidence interval for θ , given B = 1000 replications, we have:

1. Sample the '8' subjects from the given data frame
2. From these subjects sampled above, get the 'Y' and 'Z' values respectively
3. Mean : -0.067
4. Using the percentile method, the 95% confidence interval is found as : (-0.23, 0.19)

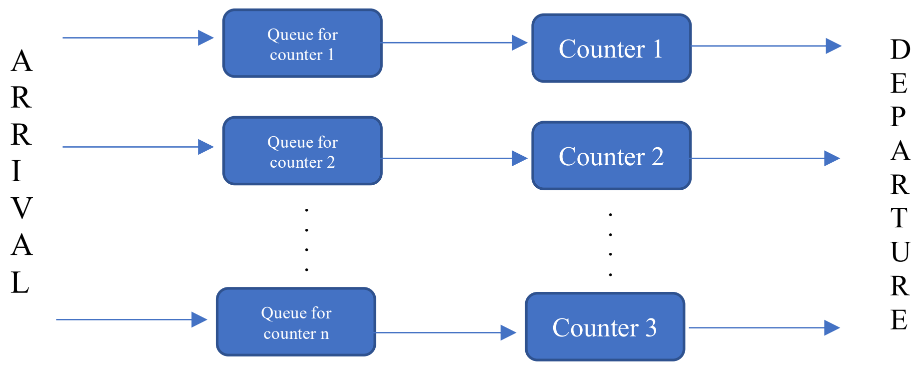
Since the 95% confidence interval of theta does not matches the required condition of $|\theta| \leq 0.20$, it can be concluded that the new drug is not a bioequivalent. In other words, the new drug is significantly different.

NOTE: The confidence interval calculations has been done using the 2.5 percentile and 97.5 percentile value. This is because we are not sure about the distribution of the data.

Answer – 2

2.1.a Problem summary

This project will aim to perform a process stimulation for Air Secure to find the optimum number of service desks required such that 90% of the customer have a waiting time of less than 8 minutes. For this project, data for 1000 customers inter arrival time(time between the arrival of customers) and service time(time taken for the service) has been given. Using this dataset, a discrete event system stimulation study is carried to cater to the requirements of Air Secure. The queueing system built in this project can be represented as:



2.1.b Problem Assumptions

The following assumptions are made in this project:

1. On arrival, customers will choose smallest queue and will remain in that queue until severed The shortest queue will also include the count of whether or not a customer is present in the service desk. This will help customers to select the service desk that are free rather than those that are occupied by other customers.
2. Since the distribution is not available for the entire time frame of this experiment, the samples will be obtained via bootstrapping. Due to this, to calculate the 95% confidence interval, I will be using the percentile method.

2.1.c Problem Objective

The objective for this project is: To find the minimum number of service desk required so that 90% of the customers do not have to wait for more than 8 minutes

2.2.a Variables Specifications

The following variables and their characteristic are used for this study:

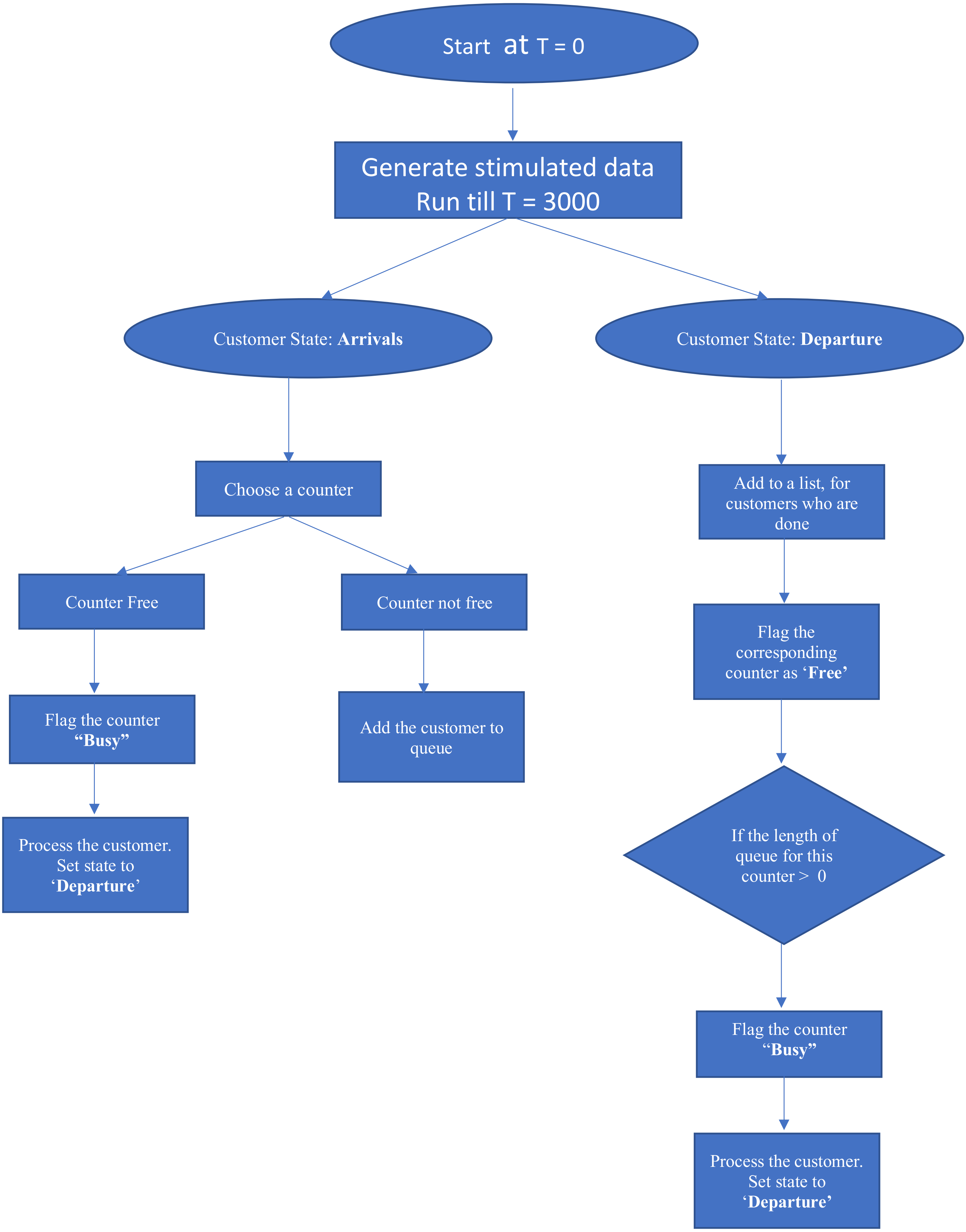
1. ***System***: This is a discrete-event systems (DES), because the state variables changes at discrete points in time
2. ***Components or Elements of the system***: Components that define the system
    a. *Customer*: The various attributes used to define customer are:
        i. Arrival Time: Time at which customer arrives
        ii. Service start time: Time at which the service starts for this customer
        iii. Service end time: Time at which the service ends for this customer
        iv. Counter ID: The service desk chosen by the customer on arrival. Customer always choose the service desk with minimum number of customers present in the queue (As defined in 2.1.b **Problem Assumptions**)
    b. *Service Desk/Counter*: The various attributes used to define the service desk are:
        i. Counter Occupied or not: A flag indicating if the service desk/counter is occupied or not
        ii. Queue of a counter: The queue in front of each service desk/counter. Every queue will follow FIFO (First in First Out)
3. ***Attributes***: Defining certain characteristic of the components
    a. Number of counters: Number of counters or service desk in the system
    b. FIFO queue: The queue will be strictly First in First Out (FIFO)
    c. Time constrain: For this system, the stimulation will be run for 3000 units of time.
4. ***Interaction between components:*** Defining the interaction between various components of the system
    a. *Customer*:
        i. On arriving, a customer will choose the service desk. The customer chooses the service desk with shortest queue
        ii. Customer will either get directly severed if there is no one in the queue or will wait in the queue for the chosen service desk
    b. *Service Desk/Counter*:
        i. A service desk /counter will either be empty or will be serving a customer
        ii. As soon as one customer leaves the counter, the service desk will serve the customer who is next in the queue for that counter (FIFO)
5. ***Derived attributes***:
    a. Waiting time: For a customer, waiting time can be calculated as:
        Waiting time = Time of service start for the customer - Time of arrival of customer

2.2.b Project Dynamics

The formal set-up of the above system is the following:

1. ***System state:*** Variables that describe the process (Definitions above)
    a. Customer:
        i. Arrival Time: Float datatype
        ii. Service start time: List datatype of size N (Number of counters)
        iii. Service end time: List datatype of size N (Number of counters)
        iv. Counter ID: Integer datatype. For e.g. 0 for counter '1'
        v. State Type: String datatype with two values
            - ***"Arrival"***: Signifying arrival of a customer at certain time
            - ***"Departure"***: Signifying that the processing is done or process completion
    b. Counters:
        i. Flag (Counter Occupied or not) : Binary datatype (1,0)
        ii. Counter Queue: Dictionary data type. Keys of the dictionary represent each counter. The value of the dictionary is a list.
2. ***System event***: Variables that change the system state
    a. Current time: The state of system changes in accordance to the current time.
3. ***Distribution***: Unknown distributions, hence samples obtained via bootstrapping
4. ***Queue***: First in First Out (FIFO)
5. ***Interaction***: As defined above
6. ***Derived attributes***: As defined above
7. ***Result Evaluation***: To evaluate the result **95% Confidence Interval** of the probability of waiting time less than 8 minutes will used. Since we do not know the distribution of the results, to calculate the 95% confidence interval percentile (2.5%, 97.5%) will be used.

Queue Dynamic Diagram



2.3 Result & Analysis

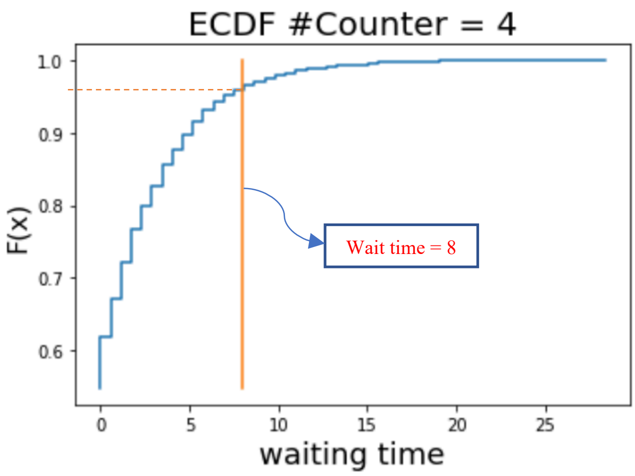
The stimulation process was done until T(Time) = 3000 units. About 3600(Number of events changes) events were obtained for this stimulation. Upon experimenting with varied number of counters, the following observations were made:

Number of Service desk	Mean waiting time for 90% random customer (minutes)	95% Confidence Interval of probability of customer wait time less than 8 minutes (2.5%, 97.5%)
2	476.90	(0.001, 0.03)
3	8.43	(0.5, 0.69)
4	1.44	(0.94, 0.99)

The **95% Confidence Interval of probability of customer wait time less than 8 minutes** was used to determine the probability of customers with waiting time less than 8 will used. Since we do not know the distribution of the results, to calculate the 95% confidence interval percentile (2.5%, 97.5%) was used.

The **mean waiting time for 90% random customer (minutes)** was calculated to understand the decrease in the mean waiting time of customers for a given number of service desk

The **ecdf plot for counters = 4,** can be seen as:



It is evident that, to cater to 90% of the customers, such that they don't have to wait for more than 8 minutes, Air Secure needs to have 4 service desks. With 4 service desks, the mean waiting time is about 1.44 minutes. The probability of customer waiting less than 8 minutes has a 95% confidence interval (0.92, 0.99) . From ecdf plot, it can be concluded that, with 4 counters, the probability of customers with wait time less than or equal to 8 minutes is greater than 90%

2.4 Conclusion

From this analysis, it can be concluded that a minimum of 4 services desks will ensure 90% or more of the customers will be served within 8 minutes of their arrival. This has been verified using the mean, confidence interval and the ecdf plot. If 3 service desks are deployed, then about 50% of the customers will have to wait for more than 8 minutes. Deploying more than 4 service desks will be waste of resource.

2.5 **Code** (Check Appendix point 2 below)

Appendix

1. Question 1 code

```
import numpy as np
import pandas as pd
import random
random.seed(12345)

df = pd.read_csv('bio_data.csv')
df['Z'] = df['old - placebo']
df['Y'] = df['new - old']
# Plug-in
theta_hat = np.mean(df['Y'])/np.mean(df['Z'])
theta_hat
# Bootstrap
bootstrap_theta = []
for i in range(1,1000):
  df_sample = np.random.choice(df['subject'], size= 8, replace=True)
  y_sample = []
  z_sample = []
  for i in df_sample:
    y_sample.append(list(df[df['subject'] == i]['Y'])[0])
    z_sample.append(list(df[df['subject'] == i]['Z'])[0])
  # append all the thetas
  all_theta.append(np.mean(y_sample)/np.mean(z_sample))
# Get the values
print("Mean: " , np.mean(all_theta))
print(np.percentile(all_theta,2.5) , np.percentile(all_theta,97.5))
```

2. Question 2 code

```
import pandas as pd
import os
import numpy as np
import heapq
import matplotlib.pyplot as plt

# Get the data
df = pd.read_csv('data_orIginal.csv')
# Set the number of counters
n = 4
# Get the queue for all the counters
queue = {0: [] , 1 : [] , 2 : [] , 3 : [] }
############################################################
############################################################

# Create a class object defining the system state
# counter_id will tell which counter will the customer prefer depending on the lenght of the queue
# arrival_time  is the time of arrival
# event_type  is the state of the customer, namely "Arrived" or "Departure"
# all_containers_start, all_containers_end : A list to currate the start time and end time
```

```
class state:
   def __init__(self, arrival_time, event_type, counter_id, all_containers_start, all_containers_end ):
      # Arrivalt time
      self.m_arrival_time = arrival_time
      self.m_event_type = event_type
      self.m_counter_id = counter_id
      self.m_all_containers_start = all_containers_start
      self.m_all_containers_end = all_containers_end
   def Print(self):
      print(self.m_arrival_time, self.m_counter_id , (",self.m_event_type,"))

###########################################################
###########################################################

# This is for priority queue
priorityQueue = []
# All done events
ell = []
# FLAG container if occupied
counter = [0 for _ in range(n)]
#indicates the current time
t_current = 0
# Create first object
interval_ = np.random.choice(df['inter_arrival_time'], size= 1, replace=True)[0]
# Get the data
t_current +=  interval_
data = state(t_current , "ARRIVAL", 0,[-1] * n, [-1] * n)
# push the entry in the queue s
heapq.heappush(priorityQueue,(t_current,data))


while(t_current < 3000 ):
   # Get the object in the priority queue
   obj = heapq.heappop(priorityQueue)
   # the event details
   event = obj[1]
   t_current = obj[0]
   # If arrival
   ###########################################################
   ###########################################################
   if (event.m_event_type) == "ARRIVAL":
      # identify the counter that has shortest queues
      len_ =  [len(value) for key, value in queue.items()]
      # shortest queue includes the customer in the counter as well
      for i in range(len(len_)):
         len_[i] += counter[i]
      counter_ = len_.index(min(len_))
      event.m_counter_id = counter_
      # For the object that has arrived, check if the counter is free or not
      if counter[event.m_counter_id] == 0:
         # make it busy
         counter[event.m_counter_id] = 1
         # mark the event as done
         event.m_event_type = "DONE"
         # change the counter start time and end time
         event.m_all_containers_start[event.m_counter_id] = t_current
         event.m_all_containers_end[event.m_counter_id] = t_current + np.random.choice(df['service_time'])
         heapq.heappush(priorityQueue,(event.m_all_containers_end[event.m_counter_id] ,event))
      # if the counter is busy, move the object to the list
      else:
         queue[event.m_counter_id].append(event)
      ###########################################################
      # During the arrival we stimulate the arrival of others
```

```
        inter_arrival = np.random.choice(df['inter_arrival_time'])
        # Get the next time
        t_next = t_current + inter_arrival
        #Choose a counter depending on the length of the queue
        len_ =  [len(value) for key, value in queue.items()]
        for i in range(len(len_)):
          len_[i] += counter[i]
        counter_ = len_.index(min(len_))
        data = state(t_next, "ARRIVAL", counter_ , [-1] * n  ,[-1] * n)
        # push to the queue
        heapq.heappush(priorityQueue,(t_next ,data))
    ##########################################################
    ##########################################################
    # At departute
    elif (event.m_event_type) == "DONE":
        # Add events to the list
        ell.append(event)
        # Free the counter
        counter[event.m_counter_id] = 0
        # If list is the queue is greater than 0,
        if(len(queue[event.m_counter_id])!=0):
          counter[event.m_counter_id] = 1
          # get the first customer in the queue
          queue_waiting = queue[event.m_counter_id].pop(0)
          # Set the event as
          queue_waiting.m_event_type = "DONE"
          queue_waiting.m_all_containers_start[event.m_counter_id] = t_current
          queue_waiting.m_all_containers_end[event.m_counter_id] = t_current + 
np.random.choice(df['service_time'])
          # push the event to priority queue
          heapq.heappush(priorityQueue,(queue_waiting.m_all_containers_end[event.m_counter_id]
,queue_waiting))
    else:
        print("ERROR")



##########################################################
##########################################################

# Get the waiting time
all_waiting_time = []
for i in range(len(ell)):
    waiting_time =  ell[i].m_all_containers_start[ell[i].m_counter_id]   - ell[i].m_arrival_time
    all_waiting_time.append(waiting_time)


##########################################################
##########################################################

# Get the edcf plot
import statsmodels.api as sm # recommended import according to the docs
import matplotlib.pyplot as plt

sample = all_waiting_time
ecdf = sm.distributions.ECDF(sample)

x = np.linspace(min(sample), max(sample))
y = ecdf(x)
plt.step(x, y)

plt.plot( [8] * 50 , np.linspace(min(y), max(y)))
plt.title('ECDF #Counter = 4', fontsize=20)
```

```
plt.xlabel('waiting time', fontsize=18)
plt.ylabel('F(x)', fontsize=16)
plt.show()

##############################################################
#############################################################
# To esttimate the mean and the standard deviation, we have
N = len(all_waiting_time)
all_probability = []
for i in range(1,N):
    # get 100 random index
    batch = np.random.choice(all_waiting_time , size = 100, replace= True)
    if (len(batch)) != 0:
        # Get the probability
        all_probability.append(len([i for i in batch if i <= 8])/len(batch))

tmean = np.mean(all_probability)
tstd = np.std(all_probability)
print(np.percentile(all_probability,2.5) , np.percentile(all_probability,97.5))
```

References

Confidence Interval : [Link]
Lab 6 codes
Lecture slides