Student name: Steven Xie

Student number: 998979627

Program & Department: MEng MIE

Homework 3

Problem 1.

(a).

To represent the Load/Unload and clean time using the distributions in PythonSim, we load the data in R and use fitdistrplus package to fit the data with different distribution and compare the goodness of fit both visually and statistically.

For Load/Unload data, lognormal, Weibull and gamma distribution are all "not rejected". However, the lognormal distribution does not fit well on the tail of Q-Q plot. Weibull distribution is not available in PythonSim package. Gamma distribution fits the best visually and has the best p value. Note that the shape of fitted gamma distribution is 2.095223 which is very close to 2. We can use Erlang distribution to represent gamma distribution when the shape is an integer. The Load/Unload can be represented by SimRNG.Erlang(2, 0.2377534, 2).

For Clean data, Weibull, gamma, and exponential distribution are accepted. The exponential distribution fit has the highest p value and is available in PythonSim. The Clean can be represented by SimRNG.Expon(1.822926, 2).

Code shown below.

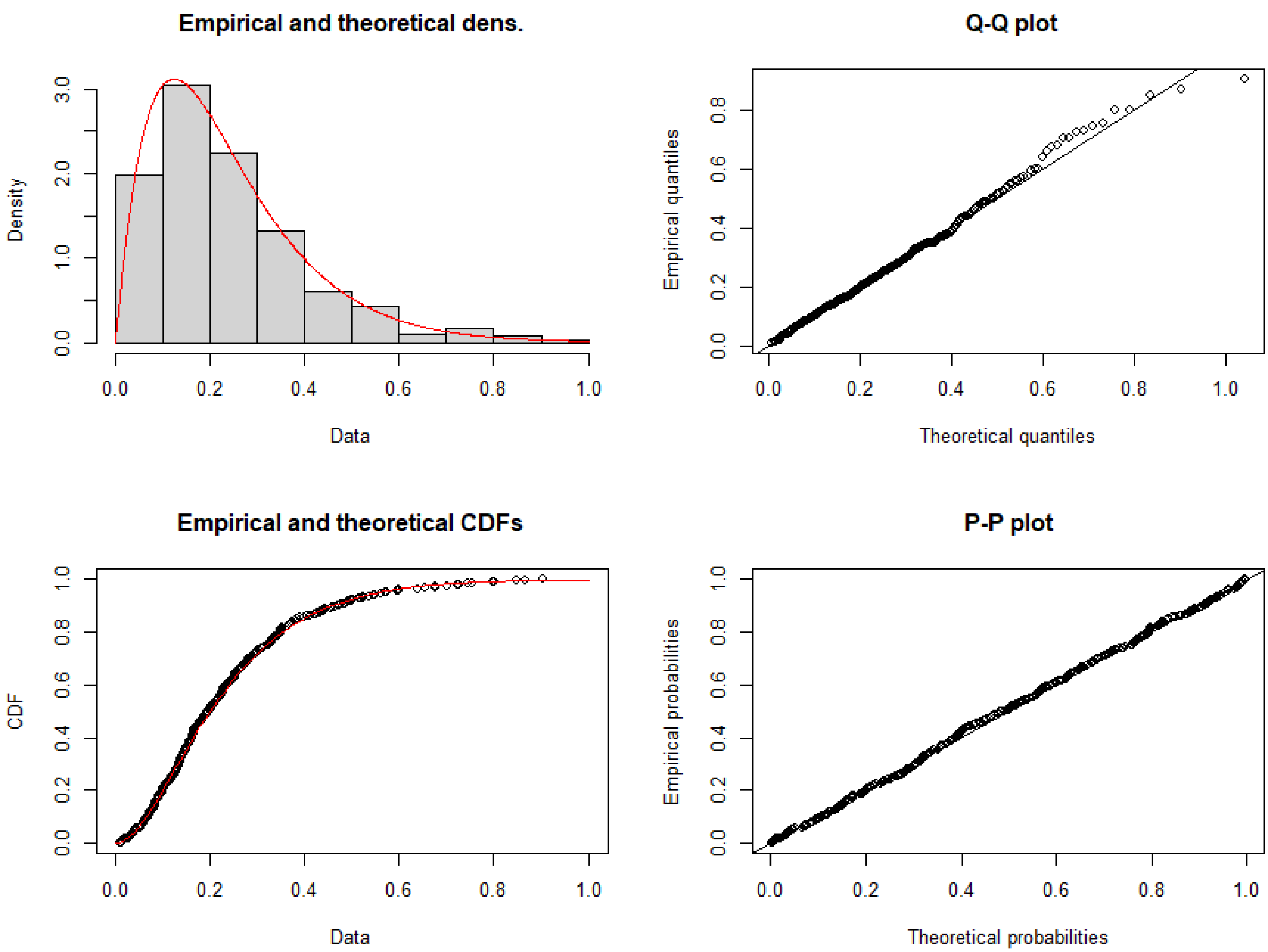
For Load/Unload distribution

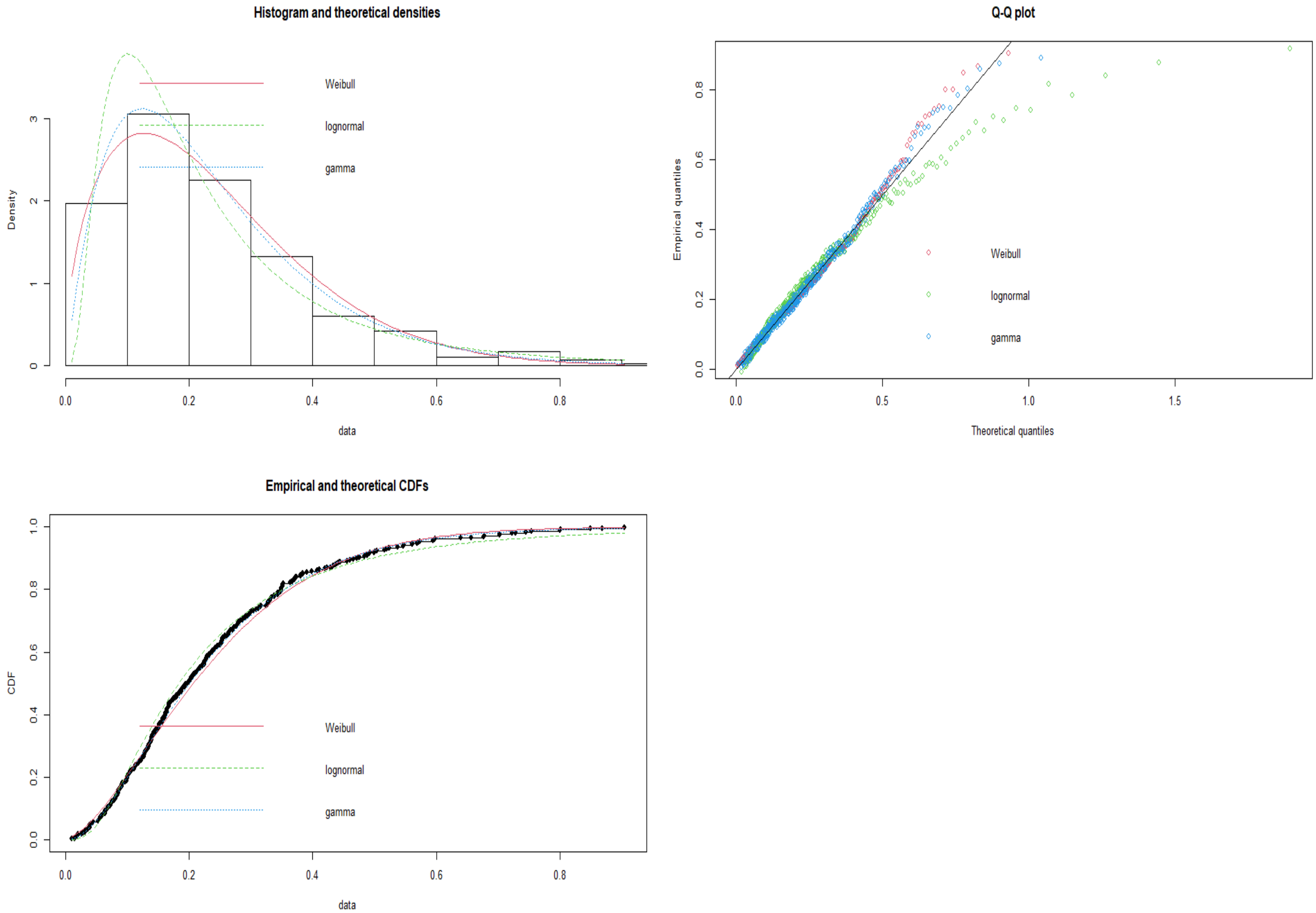
```r
library(fitdistrplus)
Data <- read.csv("SemiconductorData.csv", header=TRUE,
                    sep=",", na.strings=c("NA", ""), stringsAsFactors=FALSE, as.is=TRUE)
MyData <- Data$LOADUNLOAD
plotdist(Data$LOADUNLOAD, histo = TRUE, demp = TRUE)
descdist(MyData)
# fit a lognormal distribution to the LOS samples
fln <- fitdist(MyData, "lnorm")
summary(fln)
plot(fln)

# fit a weibull distribution
fw <- fitdist(MyData, "weibull")
summary(fw)
plot(fw)

# fit a gamma distribution
fg <- fitdist(MyData, "gamma")
summary(fg)
plot(fg)


# compare the fits
par(mfrow = c(2, 2))
plot.legend <- c("Weibull", "lognormal", "gamma")
denscomp(list(fw, fln, fg), legendtext = plot.legend)
qqcomp(list(fw, fln, fg), legendtext = plot.legend)
cdfcomp(list(fw, fln, fg), legendtext = plot.legend)

# Perform GofFit tests
gof_results <- gofstat(list(fw, fln, fg), fitnames = c("weibull", "lnorm", "gamma"))
gof_results
gof_results$kstest

gof_results$chisq
gof_results$chisqpvalue
gof_results$chisqtable
```

Important Run Results

```
> descdist(MyData)
summary statistics
------
min:  0.0103516   max:  0.904541
median:  0.1968625
mean:  0.2377534
estimated sd:  0.1665446
estimated skewness:  1.341048
estimated kurtosis:  4.947782
> gof_results$kstest
       weibull          lnorm          gamma
"not rejected" "not rejected" "not rejected"
> gof_results$chisq
 weibull    lnorm    gamma
18.07778 30.85809 13.50062
> gof_results$chisqpvalue
   weibull      lnorm      gamma
0.31936186 0.01402876 0.63586202
```

```
> # fit a gamma distribution
> fg <- fitdist(MyData, "gamma")
> summary(fg)
Fitting of the distribution ' gamma ' by maximum likelihood
Parameters :
       estimate Std. Error
shape 2.095223  0.1379891
rate  8.813089  0.6554110
Loglikelihood:  226.4825   AIC:  -448.965   BIC:  -440.9821
Correlation matrix:
          shape      rate
shape 1.0000000 0.8855823
rate  0.8855823 1.0000000
```




Histogram and theoretical densities
Density
data
Weibull
lognormal
gamma
Q-Q plot
Empirical quantiles
Theoretical quantiles
Weibull
lognormal
gamma
Empirical and theoretical CDFs
CDF
data
Weibull
lognormal
gamma

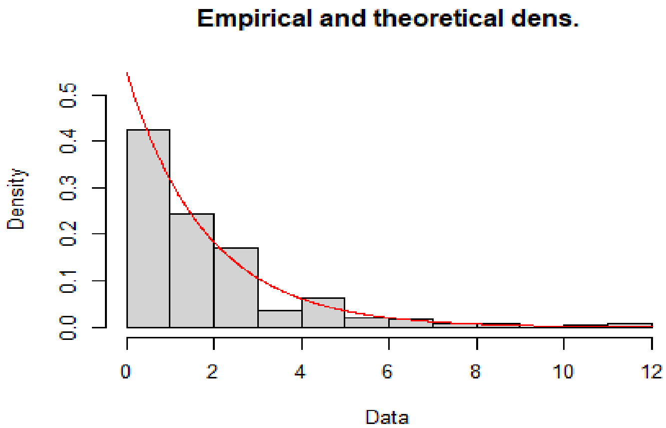
For Clean data distribution:

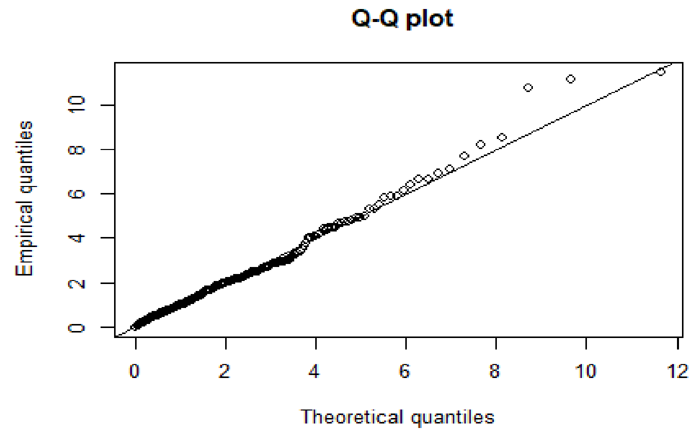
```r
library(fitdistrplus)
Data <- read.csv("SemiconductorDataCLEAN.csv", header=TRUE,
                 sep=",", na.strings=c("NA", ""), stringsAsFactors=FALSE, as.is=TRUE)
MyData <- Data$CLEAN
plotdist(MyData, histo = TRUE, demp = TRUE)
descdist(MyData)
# fit a lognormal distribution to the LOS samples
fln <- fitdist(MyData, "lnorm")
summary(fln)
plot(fln)

# fit a weibull distribution
fw <- fitdist(MyData, "weibull")
summary(fw)
plot(fw)

# fit a gamma distribution
fg <- fitdist(MyData, "gamma")
summary(fg)
plot(fg)

# fit an expon distribution
fex <- fitdist(MyData, "exp")
summary(fex)
plot(fex)

# compare the fits
par(mfrow = c(2, 2))
plot.legend <- c("Weibull", "lognormal", "gamma", "expon")
denscomp(list(fw, fln, fg, fex), legendtext = plot.legend)
qqcomp(list(fw, fln, fg, fex), legendtext = plot.legend)
cdfcomp(list(fw, fln, fg, fex), legendtext = plot.legend)

# Perform GofFit tests
gof_results <- gofstat(list(fw, fln, fg, fex), fitnames = c("weibull", "lnorm", "gamma", "expon"))
gof_results
gof_results$kstest

gof_results$chisq
gof_results$chisqpvalue
gof_results$chisqtable
```

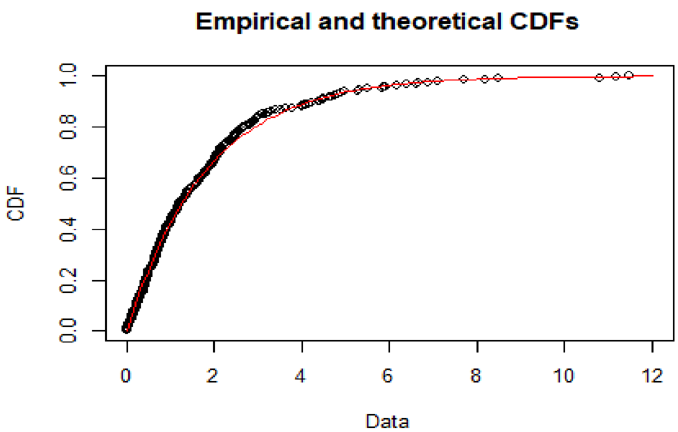
Important Run Results

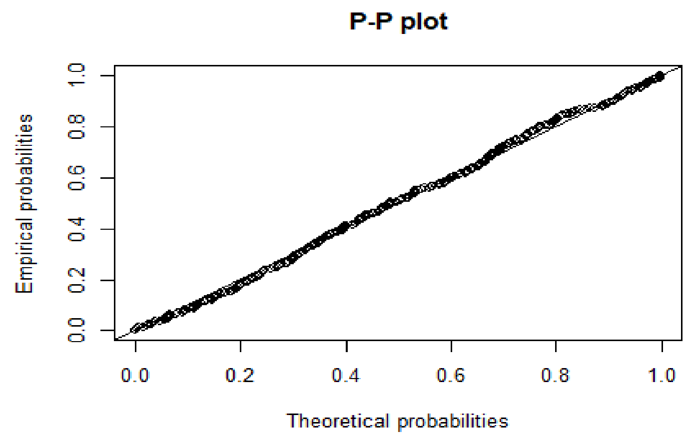
```
> descdist(MyData)
summary statistics
------
min:  0.00157231   max:  11.4891
median:  1.21073
mean:  1.822926
estimated sd:  1.875516
estimated skewness:  2.201125
estimated kurtosis:  9.375394
```

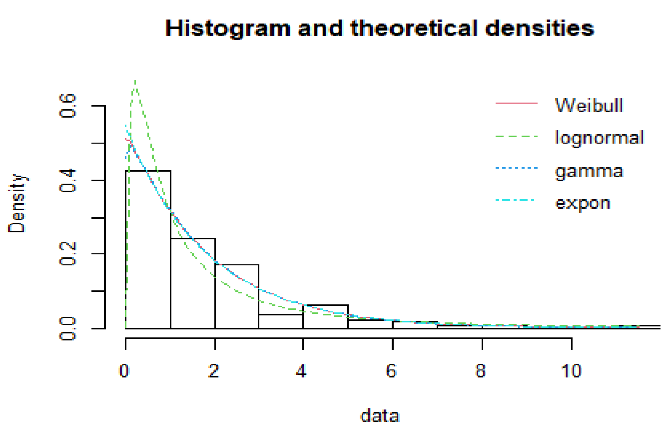
```
> # fit an expon distribution
> fex <- fitdist(MyData, "exp")
> summary(fex)
Fitting of the distribution ' exp ' by maximum likelihood
Parameters :
       estimate Std. Error
rate 0.5485687 0.03167152
Loglikelihood:  -480.1328   AIC:  962.2657   BIC:  965.9695
> plot(fex)
```

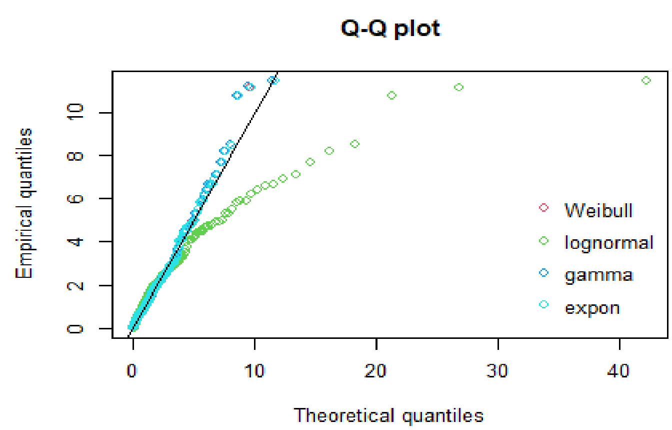

Empirical and theoretical dens.
Density
Data

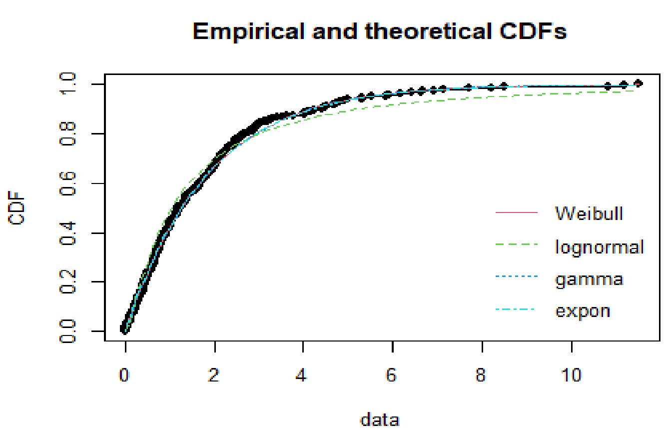

Q-Q plot
Empirical quantiles
Theoretical quantiles


Empirical and theoretical CDFs
CDF
Data


P-P plot
Empirical probabilities
Theoretical probabilities


Histogram and theoretical densities
Density
data
Weibull
lognormal
gamma
expon


Q-Q plot
Empirical quantiles
Theoretical quantiles
Weibull
lognormal
gamma
expon


Empirical and theoretical CDFs
CDF
data
Weibull
lognormal
gamma
expon

(b).

To estimate the long-run average cycle time for each product, we first build the event-based simulation model. The model contains 6 events: Release, Clean, Load, Oxidize, Unload and EndOfProduction. At Release event, we give p = 0.6 (SimRNG.Uniform(0,1,1) <= 0.6) to release type C product and mark the ClassNum 1 for product type C, then we schedule the Clean event in 15 min. The Entity class in SimClasses has an additional attribute self.Cycle with default value of 0 to count the number of cycles in production. Otherwise, the product type D is released with ClassNum 2 and Cycle = 0.

In the Clean event, Cycle number is checked to identify which event is the product from. If the product is from the Release event when cycle number equals 0, if the clean server is not all busy, seize the product and schedule the Load event. If the product is from Unload event, free the server unload, if the queue of unload is greater than 0, the unload server seizes a product and schedule the Clean event.

In the Load event, free the clean server by 1. If the load servers are not all busy, seize the product and schedule the Oxidize event. If the queue of clean is greater than 0, the clean server seizes a product and schedule the Load event.

In the Oxidize event, everything else is like load event, but the product type C has 2.7 hr oxidize time, product type D has 2 hr oxidize time.

In the Unload event, add 1 to the product cycle. For product type C if the product cycle is less than 5, schedule the Clean event, otherwise schedule the EndOfProducton event. For product type D, if the product cycle is less than 3, schedule the Clean event, otherwise schedule the EndOfProduction event.

In the EndOfProduction event, free the unload server by 1, record the total time of 2 product types in system separately. Add the mean of cycle time after each product is finished to a list. Check the previous Unload event queue, if the queue of unload is greater than 0, ask the unload server to seize a product and apply the same algorithm in the unload event.

To determine the warmup period and total run length, we first set the run length to 5000 and the warmup to 0 and run 1 replication of the simulation model. Plot the mean cycle time for each product type vs the number of products released. We found that the mean cycle time starts to vary around a common value at 300 production of product C, and the mean cycle time starts to vary around a common value at 200 production of product D. Therefore we determine that an appropriate warmup period could be (200 + 300) times the release rate 1 which gives us warmup d = 500. Take m = 10d we use run length 5500 for the simulation model. Then run the simulation model with 10 replications, compute the 95% CI for total mean cycle time for each product type.

Simulation Result:

For product C, estimated long run average cycle time with 95% CI is:

(29.377328585289614, 29.277672307412974, 29.476984863166255)

For product D, estimated long run average cycle time with 95% CI is:

(15.681798938289173, 15.576679156489538, 15.786918720088808)

Code shown below

```
import SimFunctions
import SimRNG
import SimClasses
import numpy as np
import scipy.stats as stats
import matplotlib.pyplot as plt

def t_mean_confidence_interval(data,alpha): # compute the CI with set alpha
    a = 1.0 * np.array(data)
    n = len (a)
    m, se = np.mean(a), stats.sem(a)
    h = stats.t.ppf(1 - alpha /2,n-1)*se
    return m, m-h, m+h

ZSimRNG = SimRNG.InitializeRNSeed()

# set the queues and servers
Queue_clean = SimClasses.FIFOQueue()
Queue_load = SimClasses.FIFOQueue()
Queue_oxidize = SimClasses.FIFOQueue()
Queue_unload = SimClasses.FIFOQueue()
Server_clean = SimClasses.Resource()
Server_load = SimClasses.Resource()
Server_oxidize = SimClasses.Resource()
Server_unload = SimClasses.Resource()
TotalTime1 = SimClasses.DTStat()
TotalTime2 = SimClasses.DTStat()
Calendar = SimClasses.EventCalendar()


TheCTStats = []
TheDTStats = []
TheQueues = []
TheResources = []

TheDTStats.append(TotalTime1)
TheDTStats.append(TotalTime2)
TheQueues.append(Queue_clean)
TheQueues.append(Queue_load)
TheQueues.append(Queue_oxidize)
TheQueues.append(Queue_unload)
TheResources.append(Server_clean)
TheResources.append(Server_load)
TheResources.append(Server_oxidize)
TheResources.append(Server_unload)

# standard setup but only Totaltime are used
AllQueue_clean = []
AllQueue_load = []
```

```python
AllQueue_oxidize = []
AllQueue_unload = []
AllTotalTime1 = []
AllMeanTC = []
AllMeanTD = []
AllTotalTime2 = []
AllUtil1 = []
AllUtil2 = []

# set the number of servers
Server_clean.SetUnits (9)
Server_load.SetUnits (2)
Server_oxidize.SetUnits (11)
Server_unload.SetUnits (2)

# set the run length and warmup period
RunLength = 5000
WarmUp = 0

def Release():
    if SimRNG.Uniform(0, 1, 3) <= 0.6: # p = 0.6, release type C product
        SimFunctions.Schedule(Calendar, "Release", 1)
        ProductC = SimClasses.Entity()
        ProductC.ClassNum = 1
        ProductC.Cycle = 0
        SimFunctions.SchedulePlus(Calendar, "Clean", 0.25, ProductC) # 0.25 hr = 15
minutes
    else: # p = 0.4, release type C product
        SimFunctions.Schedule(Calendar, "Release", 1)
        ProductD = SimClasses.Entity()
        ProductD.ClassNum = 2
        ProductD.Cycle = 0
        SimFunctions.SchedulePlus(Calendar, "Clean", 0.25, ProductD)

def Clean(Product):
    Queue_clean.Add(Product)
    if Product.Cycle == 0: #Product from release, first cycle
        if Server_clean.Busy < 9:
            Server_clean.Seize(1)
            NextProduct = Queue_clean.Remove() # customer leaves the queue if served
            SimFunctions.SchedulePlus(Calendar, "Load", SimRNG.Expon(1.822926, 2),
NextProduct)
    else: #Product from Unload event, not first cycle
        Server_unload.Free(1)
        if Queue_unload.NumQueue() > 0:
            Server_unload.Seize(1)
            NextProduct = Queue_unload.Remove()  # customer leaves the queue if served

            if NextProduct.ClassNum == 1:
                if NextProduct.Cycle < 5:
                    SimFunctions.SchedulePlus(Calendar, "Clean", SimRNG.Erlang(2,
0.2377534 , 2), NextProduct)
                else:
                    SimFunctions.SchedulePlus(Calendar, "EndOfProduction",
SimRNG.Erlang(2, 0.2377534 , 2), NextProduct)
            elif NextProduct.ClassNum == 2:
                if NextProduct.Cycle < 3:
                    SimFunctions.SchedulePlus(Calendar, "Clean", SimRNG.Erlang(2,
0.2377534 , 2), NextProduct)
                else:
                    SimFunctions.SchedulePlus(Calendar, "EndOfProduction",
SimRNG.Erlang(2, 0.2377534 , 2), NextProduct)
```

```
def Load(Product):
    Server_clean.Free(1)
    Queue_load.Add(Product)

    if Server_load.Busy < 2:
        Server_load.Seize(1)
        NextProduct = Queue_load.Remove() # customer leaves the queue if served
        SimFunctions.SchedulePlus(Calendar, "Oxidize", SimRNG.Erlang(2, 0.2377534 , 2),
NextProduct)
    if Queue_clean.NumQueue() > 0:
        Server_clean.Seize(1)
        NextProduct = Queue_clean.Remove()  # customer leaves the queue if served
        SimFunctions.SchedulePlus(Calendar, "Load", SimRNG.Expon(1.822926, 2),
NextProduct)


def Oxidize(Product):
    Server_load.Free(1)
    Queue_oxidize.Add(Product)

    if Server_oxidize.Busy < 11:
        Server_oxidize.Seize(1)
        NextProduct = Queue_oxidize.Remove()  # customer leaves the queue if served
        if NextProduct.ClassNum == 1:
            SimFunctions.SchedulePlus(Calendar, "Unload", 2.7, NextProduct)
        elif NextProduct.ClassNum == 2:
            SimFunctions.SchedulePlus(Calendar, "Unload", 2, NextProduct)

    if Queue_load.NumQueue() > 0:
        Server_load.Seize(1)
        NextProduct = Queue_load.Remove() # customer leaves the queue if served
        SimFunctions.SchedulePlus(Calendar, "Oxidize", SimRNG.Erlang(2, 0.2377534 , 2),
NextProduct)

def Unload(Product):
    Server_oxidize.Free(1)
    Queue_unload.Add(Product)
    Product.Cycle += 1
    if Server_unload.Busy < 2:
        Server_unload.Seize(1)
        NextProduct = Queue_unload.Remove()  # customer leaves the queue if served

        if NextProduct.ClassNum == 1:
            if Product.Cycle < 5:
                SimFunctions.SchedulePlus(Calendar, "Clean", SimRNG.Erlang(2, 0.2377534 ,
2), NextProduct)
            else:
                SimFunctions.SchedulePlus(Calendar, "EndOfProduction", SimRNG.Erlang(2,
0.2377534 , 2), NextProduct)
        elif NextProduct.ClassNum == 2:
            if Product.Cycle < 3:
                SimFunctions.SchedulePlus(Calendar, "Clean", SimRNG.Erlang(2, 0.2377534 ,
2), NextProduct)
            else:
                SimFunctions.SchedulePlus(Calendar, "EndOfProduction", SimRNG.Erlang(2,
0.2377534 , 2), NextProduct)

    if Queue_oxidize.NumQueue() > 0:
        Server_oxidize.Seize(1)
        NextProduct = Queue_oxidize.Remove()  # Product leaves the queue if served
        if NextProduct.ClassNum == 1:
            SimFunctions.SchedulePlus(Calendar, "Unload", 2.7, NextProduct)
```

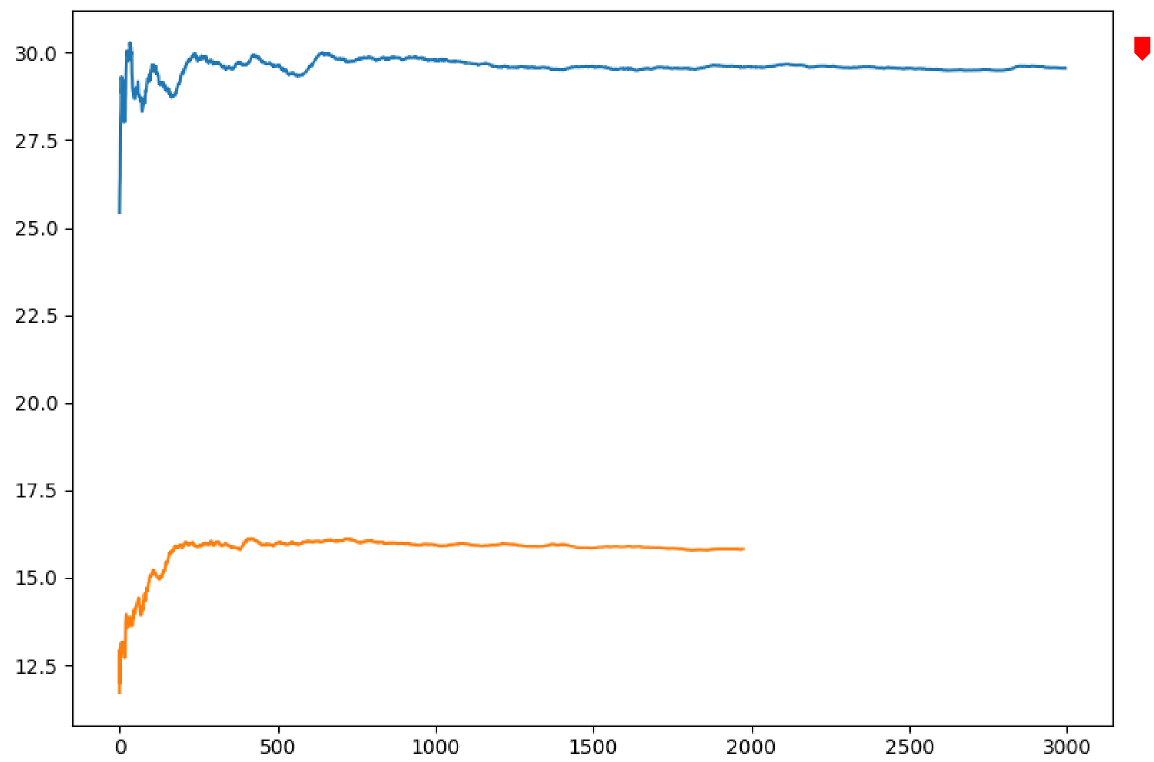
```
        elif NextProduct.ClassNum == 2:
            SimFunctions.SchedulePlus(Calendar, "Unload", 2, NextProduct)


def EndOfProduction(Product): # give an object to the EndOfProduction function to record total time in system
    Server_unload.Free(1)
    if Product.ClassNum == 1:
        TotalTime1.Record(SimClasses.Clock - Product.CreateTime) # Record the ProductC's total time in system
        AllMeanTC.append(TotalTime1.Mean())
    elif Product.ClassNum == 2:
        TotalTime2.Record(SimClasses.Clock - Product.CreateTime) # Record the ProductD's total time in system
        AllMeanTD.append((TotalTime2.Mean()))
    if Queue_unload.NumQueue() > 0:
        Server_unload.Seize(1)
        NextProduct = Queue_unload.Remove()  # customer leaves the queue if served

        if NextProduct.ClassNum == 1:
            if Product.Cycle < 5:
                SimFunctions.SchedulePlus(Calendar, "Clean", SimRNG.Erlang(2, 0.2377534 , 2), NextProduct)
            else:
                SimFunctions.SchedulePlus(Calendar, "EndOfProduction", SimRNG.Erlang(2, 0.2377534 , 2), NextProduct)
        elif NextProduct.ClassNum == 2:
            if Product.Cycle < 3:
                SimFunctions.SchedulePlus(Calendar, "Clean", SimRNG.Erlang(2, 0.2377534 , 2), NextProduct)
            else:
                SimFunctions.SchedulePlus(Calendar, "EndOfProduction", SimRNG.Erlang(2, 0.2377534 , 2), NextProduct)

# replication loop
for reps in range(0, 1, 1):

    SimFunctions.SimFunctionsInit(Calendar, TheQueues, TheCTStats, TheDTStats, TheResources)
    SimFunctions.Schedule(Calendar, "Release", 1)
    SimFunctions.Schedule(Calendar, "EndSimulation", RunLength)
    SimFunctions.Schedule(Calendar, "ClearIt", WarmUp)

    NextEvent = Calendar.Remove()
    SimClasses.Clock = NextEvent.EventTime
    if NextEvent.EventType == "Release":
        Release()
    elif NextEvent.EventType == "Clean":
        Clean(NextEvent.WhichObject)
    elif NextEvent.EventType == "Load":
        Load(NextEvent.WhichObject)
    elif NextEvent.EventType == "Oxidize":
        Oxidize(NextEvent.WhichObject)
    elif NextEvent.EventType == "Unload":
        Unload(NextEvent.WhichObject)
    elif NextEvent.EventType == "EndOfProduction":
        EndOfProduction(NextEvent.WhichObject)
    elif NextEvent.EventType == "ClearIt":
        SimFunctions.ClearStats(TheCTStats, TheDTStats)

    while NextEvent.EventType != "EndSimulation":
        NextEvent = Calendar.Remove()
        SimClasses.Clock = NextEvent.EventTime
```

```
        if NextEvent.EventType == "Release":
            Release()
        elif NextEvent.EventType == "Clean":
            Clean(NextEvent.WhichObject)
        elif NextEvent.EventType == "Load":
            Load(NextEvent.WhichObject)
        elif NextEvent.EventType == "Oxidize":
            Oxidize(NextEvent.WhichObject)
        elif NextEvent.EventType == "Unload":
            Unload(NextEvent.WhichObject)
        elif NextEvent.EventType == "EndOfProduction":
            EndOfProduction(NextEvent.WhichObject)
        elif NextEvent.EventType == "ClearIt":
            SimFunctions.ClearStats(TheCTStats, TheDTStats)

    # plot the mean cycle time for each product to determine the warmup period
    plt.plot(range(0, int(TotalTime1.N())),AllMeanTC) # Mean Cycle time of product C vs
number of Product C
    plt.plot(range(0, int(TotalTime2.N())), AllMeanTD) # Mean Cycle time of product D vs
number of Product D
    plt.show()
    # Record the mean total time in system of each product
    AllTotalTime1.append(TotalTime1.Mean())
    AllTotalTime2.append(TotalTime2.Mean())
    print(AllTotalTime1[-1],AllTotalTime2[-1])

# Print the result of 95% CI
print("Estimate Long Run Time for product C is:
",t_mean_confidence_interval(AllTotalTime1,0.05))
print("Estimate Long Run Time for product D is:
",t_mean_confidence_interval(AllTotalTime2,0.05))
```

Added self.Cycle in SimClasses.py to count the number of cycles in production

```
class Entity():
    # This is the generic Entity that has a single attribute CreateTime
    def __init__(self):
        # Executes with the Entity object is created to initialize variables
        # Add additional problem-specific attributes here
        self.CreateTime = Clock
        self.Classtype = 0
        self.Cycle = 0
```

The plot of mean total time of each product type vs production number.



Run 10 replications and result shown below

```
C:\Users\steve\AppData\Local\Programs\Python\Python38-32\python.exe "C:/Users/steve/Desktop/MIE1613HS Stochastic Simulation/HW3/HW3Q1b.py"
29.559916356768415 15.818235696372696
29.326294785244738 15.491585674397868
29.600916602541812 15.764362105718117
29.244759084869006 15.566376561865548
29.35361552446455 15.534359143759003
29.5079475516872 15.52237932998842
29.196726030985754 15.887584146568203
29.4106451131004 15.745837244977753
29.32628372119385 15.648351819955344
29.24618108204038 15.838917659278362
Estimate Long Run Time for product C is:  (29.377328585289614, 29.277672307412974, 29.476984863166255)
Estimate Long Run Time for product D is:  (15.681798938289173, 15.576679156489538, 15.786918720088808)

Process finished with exit code 0
```

Problem 2

(a).

The approach is to adapt the M/M/Infinity model for simulation, run the simulation for 24 hours period and record the maximum number of drones. Run n replications and sorted all maximum numbers of drones from all replications and computed the required quantile (0.98) to get the minimum number of drones required.

The non-homogeneous Poisson arrival rate of calls is computed with Thinning method. If the U(0,1) < λ/max(λ), accept and schedule the call, where λ is the arrival rate at time period t. The simulation starts at 12am with 0 calls, so λ is [60,240,120] hourly in period [5,17,24]. The busy time for drone is exponentially distributed with mean of 0.75 hr.

QueueLength is used to record the number of drones busy, MaxQueue is used to record the maximum number of drones busy in each replication.

(b).

Simulation Result:

The mean and 95% CI of required minimum number of drones: 183 (179 187)

Code shown below.

```
import SimClasses
import SimFunctions
import SimRNG
import math
import numpy as np
import scipy.stats as stats

SimClasses.Clock = 0
MeanAR = [60,240,120] # The call arrival rate in hourly
QueueLength = SimClasses.CTStat() # record the number of drones busy
N = 0
MaxQueue = 0
RepNum = 100
MeanBusy = 0.75 # The mean time for drones to be busy after call

ZSimRNG = SimRNG.InitializeRNSeed()
Calendar = SimClasses.EventCalendar()

TheCTStats = []
TheDTStats = []
TheQueues = []
TheResources = []

TheCTStats.append(QueueLength)

AllQueueLength = []
AllMaxQueue = []
AllN = []

def t_mean_confidence_interval(data,alpha): # compute the CI with set alpha
```

```python
    a = 1.0 * np.array(data)
    n = len (a)
    m, se = np.mean(a), stats.sem(a)
    h = stats.t.ppf(1 - alpha /2,n-1)*se
    return m, m-h, m+h

def NSPP(MeanAR, Stream):
    ar = 240
    PossibleArrival = SimClasses.Clock + SimRNG.Expon(1/max(MeanAR), Stream)
    if SimClasses.Clock <= 5:
        ar = MeanAR[0]
    elif 5 < SimClasses.Clock <= 17:
        ar = MeanAR[1]
    elif 17 < SimClasses.Clock <= 24:
        ar = MeanAR[2]

    if SimRNG.Uniform(0, 1, Stream) <= ar/ max(MeanAR):
        PossibleArrival = PossibleArrival + SimRNG.Expon(1/max(MeanAR), Stream)
    nspp = PossibleArrival - SimClasses.Clock
    return nspp

def Callin():
    global MaxQueue
    global N
    interarrival = NSPP(MeanAR,1)
    SimFunctions.Schedule(Calendar, "Callin", interarrival)
    N = N + 1
    QueueLength.Record(N)
    if N > MaxQueue:
        MaxQueue = N
    SimFunctions.Schedule(Calendar, "ReadyDrone", SimRNG.Expon(MeanBusy, 2))


def ReadyDrone():
    global N
    N = N - 1
    QueueLength.Record(N)


for Reps in range(0, RepNum, 1):
    N = 0
    MaxQueue = 0
    SimFunctions.SimFunctionsInit(Calendar, TheQueues, TheCTStats, TheDTStats,
TheResources)
    interarrival = NSPP(MeanAR,1)
    SimFunctions.Schedule(Calendar, "Callin", interarrival)
    SimFunctions.Schedule(Calendar, "EndSimulation", 24)

    NextEvent = Calendar.Remove()
    SimClasses.Clock = NextEvent.EventTime
    if NextEvent.EventType == "Callin":
        Callin()
    elif NextEvent.EventType == "ReadyDrone":
        ReadyDrone()

    while NextEvent.EventType != "EndSimulation":
        NextEvent = Calendar.Remove()
        SimClasses.Clock = NextEvent.EventTime
        if NextEvent.EventType == "Callin":
            Callin()
        elif NextEvent.EventType == "ReadyDrone":
            ReadyDrone()
```

```
    AllQueueLength.append(QueueLength.Mean())
    AllMaxQueue.append(MaxQueue)
    AllN.append(N)

    #print(Reps+1, QueueLength.Mean(), MaxQueue, N)
# estimating the 0.98th quantile
quantile_index = int(np.ceil(RepNum * 0.98) - 1)
capacity = np.sort(AllMaxQueue)[quantile_index]
capacitylb95 = np.floor(capacity - 1.96 * np.sqrt(capacity *(1-0.98)))
capacityub95 = np.ceil(capacity + 1.96 * np.sqrt(capacity *(1-0.98)))
print("Estimated required minimum number of drones is:", capacity, capacitylb95,
capacityub95)
print("Estimated Expected Average # of drones busy is:",
t_mean_confidence_interval(AllQueueLength,0.05))
print("Estimated Expected Max # of drones busy:",
t_mean_confidence_interval(AllMaxQueue,0.05))
```

Run Result:

```
Estimated required minimum number of drones is: 183 179.0 187.0
Estimated Expected Average # of drones busy is: (106.70755115341052, 106.23400397088893, 107.1810983359321)
Estimated Expected Max # of drones busy: (162.55, 160.7383641639488, 164.36163583605122)

Process finished with exit code 0
```

Problem 3

(a).

Given the GBM, S(t)/S(t-1) fits a lognormal distribution. The approach is to compute S(t)/S(t-1) in Excel as rate of change and fit the rate to lognormal distribution in R with package fitdistrplus.

Result:

$\mu = 0.001525905$

$\sigma = 0.034080501$

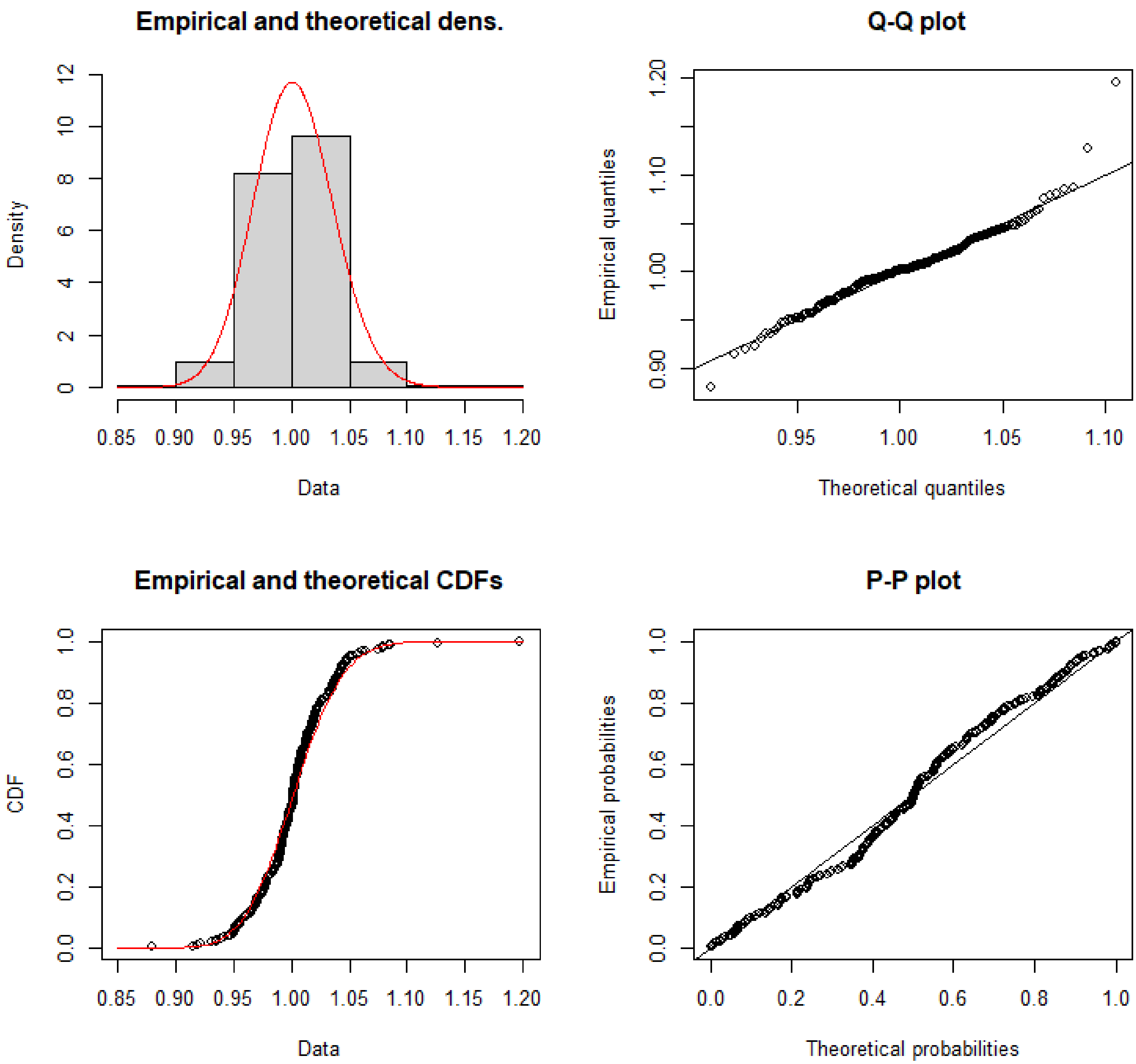
(b).

The Goodness-of-fit statistics shows that the rate although is accepted, does not fit so well to the original rate. The p value is only 0.0207 which indicates the hypothesis is at risk to be rejected. The cdf and P-P plots show that it is a good fit, the Q-Q shows that the tail is a little far from fitness line. Overall, the GBM is an acceptable fit to the stock price but not an excellent fit.

Code:

```
library(fitdistrplus)
Data <- read.csv("TeslaPrices.csv", header=TRUE,
                 sep=",", na.strings=c("NA", ""), stringsAsFactors=FALSE, as.is=TRUE)
Rate <- Data$Rate
plotdist(Rate, histo = TRUE, demp = TRUE)
descdist(Rate)

# fit a lognormal distribution to the LOS samples
fln <- fitdist(Rate, "lnorm")
summary(fln)
plot(fln)

gof_results <- gofstat(fln, fitnames = c("lnorm"))
gof_results
gof_results$kstest

gof_results$chisq
gof_results$chisqpvalue
gof_results$chisqtable
```

```
> library(fitdistrplus)
> Data <- read.csv("TeslaPrices.csv", header=TRUE,
+                  sep=",", na.strings=c("NA", ""), stringsAsFactors=FALSE, as.is=TRUE)
> Rate <- Data$Rate
> plotdist(Rate, histo = TRUE, demp = TRUE)
> descdist(Rate)
summary statistics
------
min:  0.880097   max:  1.196412
median:  1.00177
mean:  1.002112
estimated sd:  0.03449133
estimated skewness:  0.6638931
estimated kurtosis:  7.466622
> # fit a lognormal distribution to the LOS samples
> fln <- fitdist(Rate, "lnorm")
> summary(fln)
Fitting of the distribution ' lnorm ' by maximum likelihood
Parameters :
           estimate  Std. Error
meanlog 0.001525905 0.002151142
sdlog   0.034080501 0.001515198
Loglikelihood:  491.5999   AIC:  -979.1999   BIC:  -972.1489
Correlation matrix:
        meanlog sdlog
meanlog       1     0
sdlog         0     1

> plot(fln)
> gof_results <- gofstat(fln, fitnames = c("lnorm"))
> gof_results
Goodness-of-fit statistics
                                  lnorm
Kolmogorov-Smirnov statistic 0.07808953
Cramer-von Mises statistic   0.32358426
Anderson-Darling statistic   1.75266115

Goodness-of-fit criteria
                                   lnorm
Akaike's Information Criterion -979.1999
Bayesian Information Criterion -972.1489
> gof_results$kstest
        lnorm
"not rejected"
> gof_results$chisq
[1] 25.35513
> gof_results$chisqpvalue
[1] 0.02072341
> gof_results$chisqtable
          obscounts theocounts
<= 0.9516 16.000000  16.729915
<= 0.9661 16.000000  19.824321
<= 0.9756 16.000000  18.894840
<= 0.9853 16.000000  23.906191
<= 0.9907 16.000000  14.875758
<= 0.9941 16.000000   9.652301
<= 0.9984 16.000000  12.567109
<= 1.002  16.000000  10.297980
```


Empirical and theoretical dens.
Density
Data
Q-Q plot
Empirical quantiles
Theoretical quantiles
Empirical and theoretical CDFs
CDF
Data
P-P plot
Empirical probabilities
Theoretical probabilities

Problem 4.

(a).

With x0 = 0, γ = 2.

F(x) = $\frac{1}{\pi}\arctan\left(\frac{x}{2}\right) + 0.5$

$F^{-1}(x)$ = 2tan[(x-0.5) π]

Generate U (0,1) and substitute in $F^{-1}(x)$

Code Shown Below.

```
set.seed(1)
a = runif(1000) # generate the random U(0,1)
b = 2*tan((a-0.5)*pi) # Use the inverse function to generate samples
b
plot(b)
summary(b)
```

```
> summary(b)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-484.245   -1.900   -0.105   10.025    1.962 9172.356
```

(b).

Median = -0.105

Half inter-quantile range = 1.962 – (-1.900) = 3.862

Substitute F(-0.105) = 0.5 to original F(x)

X_0 = -0.105

Substitute F(1.962)-F(-1.9) = 3.862 with X_0 = -0.105 to original F(x)

γ = 1.925

The estimated parameter X_0 = -0.105, γ = 1.925 are very close the original parameters x_0 = 0, γ = 2.

Problem 5.

(a).

$d/dt\ G(t) = \lambda_1 p e^{-\lambda 1t} + \lambda_2 e^{-\lambda 2t} - p\ \lambda_2 e^{-\lambda 2t}$

$\sigma^2_A = 1.4$

$p = 0.785$

$\lambda_1 = 2p = 1.57$

$p/\lambda_1 = (1 - p)/\lambda_2$

$\lambda_2 = 0.43$

$\Lambda(t) = t^2$

$\lambda = d/dt\ \Lambda(t) = 2t = 1 \qquad t = 0.5$

$Ge(t) = \lambda_1 p e^{-\lambda 1t} + \lambda_2 e^{-\lambda 2t} - p\ \lambda_2 e^{-\lambda 2t} = 0.637$

(b).

$E(N(10))/Var(N(10)) = 1$