MIS 381N - Stochastic Control and Optimization Project 5

Jushira Thelakkat, Zhiyi(Claire) Yang, Jiayan (Will) Lu, Sidhaarthan Velur Gopalakrishnan

April 25, 2018

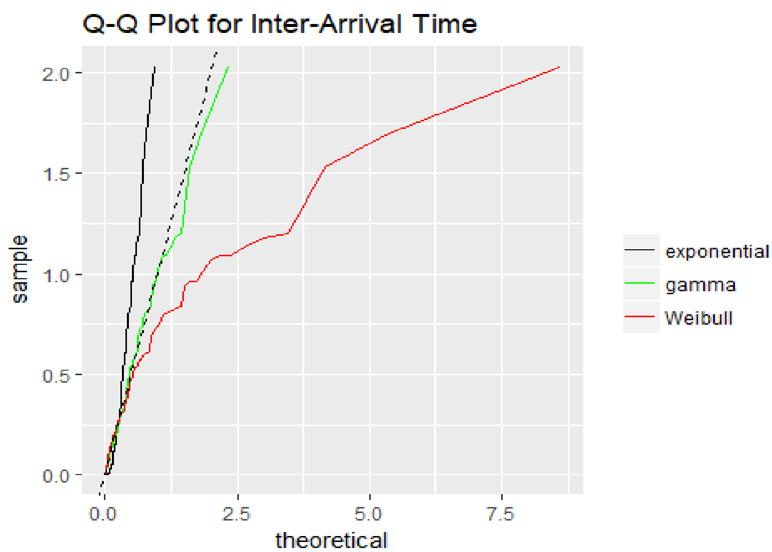
```
rm(list=ls())
library(MASS)
library(ggplot2)
load('queue.rdata')
```

Question 1

Use the function "fitdistr" in the R package MASS, to find the right parameters for inter-arrival and service time distributions. Try Gamma, Exponential and the Weibull distributions. Please use qqplots to show the goodness of fits.

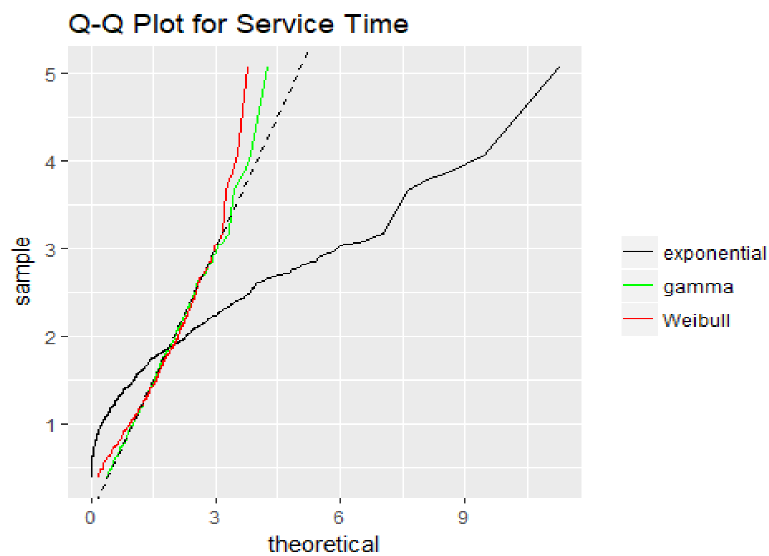
```
# Calculating Inter-Arrival TIme
IA <- diff(A)

# Fit distributions for Inter-Arrival Time
e_fit_i <- fitdistr(IA, "exponential")
g_fit_i <- fitdistr(IA, "gamma")
w_fit_i <- fitdistr(IA, "weibull")

# Fit distributions for Service Time
e_fit_s <- fitdistr(S, "exponential")
g_fit_s <- fitdistr(S, "gamma")
w_fit_s <- fitdistr(S, "weibull")

# Convert variables to a dataframe
service_arrival <- data.frame(S,A)
inter_arrival <- data.frame(IA)

# QQ-Plot for Inter-Arrival Time
ggplot(inter_arrival, aes(sample = IA)) +
  stat_qq(distribution = qexp,
          dparams = list(e_fit_i$estimate[1]),
          geom = "line",
          aes(color = "exponential")) +
  stat_qq(distribution = qgamma,
          dparams = list(g_fit_i$estimate[1], g_fit_i$estimate[2]),
          geom = "line",
          aes(color = "gamma")) +
  stat_qq(distribution = qweibull,
          dparams = list(w_fit_i$estimate[1], w_fit_i$estimate[2]),
```

```
          geom = "line",
          aes(color = "Weibull")) +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed") +
  scale_color_manual(name = "",
                     values = c("exponential" = "black",
                                "gamma" = "green",
                                "Weibull" = "red"),
                     breaks = c("exponential",
                                "gamma",
                                "Weibull")) +
  labs(title = "Q-Q Plot for Inter-Arrival Time")
```




```
# QQ-Plot for Service Time
ggplot(service_arrival, aes(sample = S)) +
  stat_qq(distribution = qexp,
          dparams = list(e_fit_s$estimate[1]),
          geom = "line",
          aes(color = "exponential")) +
  stat_qq(distribution = qgamma,
          dparams = list(g_fit_s$estimate[1], g_fit_s$estimate[2]),
          geom = "line",
          aes(color = "gamma")) +
  stat_qq(distribution = qweibull,
          dparams = list(w_fit_s$estimate[1], w_fit_s$estimate[2]),
          geom = "line",
          aes(color = "Weibull")) +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed") +
  scale_color_manual(name = "",
                     values = c("exponential" = "black",
                                "gamma" = "green",
```

```
                              "Weibull" = "red"),
                    breaks = c("exponential",
                               "gamma",
                               "Weibull")) +
  labs(title = "Q-Q Plot for Service Time")
```



Looking at the Q-Q plot for both Inter-Arrival times and Customer Wait times, we can see that the gamma distribution is the closest to the actual distribution among Weibull, Exponential and Gamma distribution

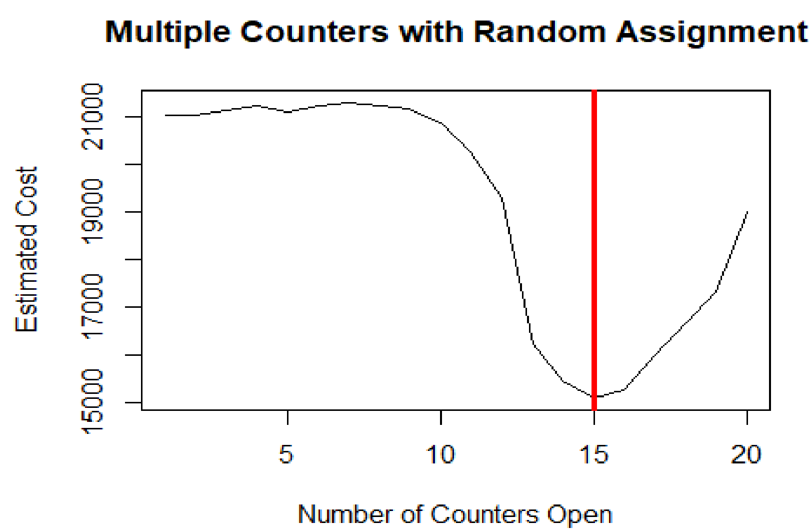
Question 2

Use the distribution that fits the best to simulate the following queuing system. In this system, there are a number of checkout counters open, each with its own queue. Arriving customers randomly join one queue. Use your simulation to find the optimal number of checkout counters to keep open. What is the expected cost (salary + penalty)?

```
# Assigning the number of simulations which refers to the number of customers
walking into the store
N = 10000

# 10,000 simulations

# Vector to hold the cost with different counters open
totalCost = rep(NA, 20)

# Setting seed
set.seed(123)
```

```
for (i in 1:20)
{
  # Simulate the counter that was picked by the customer randomly
  counter = sample(1:i, N, replace=TRUE)
  # Matrix which contains customers as rows and counters as columns
  counterSetup = matrix(0, N, i)
  for (j in (1:length(counter)))    # populate matrix
  {
    counterSetup[j,counter[j]] = 1
  }

  # Using gamma distribution to simulate the arrival and service times
  gamma_inter = rgamma(n=N-1, shape=g_fit_i$estimate[1], rate=g_fit_i$estimate[2])
  gamma_arrival = c(0, cumsum(gamma_inter))
  gamma_service = rgamma(n=N, shape=g_fit_s$estimate[1], rate=g_fit_s$estimate[2])
  # Hold costs for each of the counter
  individualCost = rep(NA, i)
  # Maximum duration of a line
  D_max = rep(NA, i)
  for (k in (1:i))
  {
    arrivalLine = matrix(0, N, i)     # simulated arrival times
    serviceLine = matrix(0, N, i)     # simulated service times
    arrivalLine[,k] = counterSetup[,k] * gamma_arrival
    serial = min(which(arrivalLine[,k] != 0))
    arrivalLine = arrivalLine[,k][arrivalLine[,k] > 0]
    serviceLine[,k] = counterSetup[,k] * gamma_service
    serviceLine = serviceLine[,k][serviceLine[,k] > 0]
    T = rep(NA,length(arrivalLine))    # service start times
    D = rep(NA,length(arrivalLine))    # durations
    W = rep(NA,length(arrivalLine))    # wait times

    T[1] = min(arrivalLine) # set first service start time as the first arrival time in the line
    D[1] = T[1] + serviceLine[serial] # set duration as the first arrival time + the first service time
    W[1] = 0 # set first wait time to 0

    for (z in 2:length(arrivalLine)) # loop to calculate service times and durations
    {
      T[z] = max(D[z-1], arrivalLine[z])
      D[z] = T[z] + serviceLine[z]
    }
    W = T - arrivalLine # calculate wait times
    individualCost[k] = sum(W > 10) # calculate cost per line
    D_max[k] = max(D) # save max duration to use as minutes all counters are open for calculating final cost
```

```
  }
  totalCost[i] = sum(individualCost, na.rm=TRUE) + 40 * max(D_max)/60 * i
}

# Total cost for different counter configurations
totalCost

##  [1] 21005.66 21021.18 21126.84 21202.98 21073.81 21226.26 21278.22
##  [8] 21226.32 21159.34 20852.64 20231.04 19274.68 16229.98 15455.91
## [15] 15089.52 15232.51 15991.07 16678.85 17320.31 18978.77

# Plot the individual cost for different counters
plot(1:20, totalCost, type='line', xlab='Number of Counters Open', ylab='Esti
mated Cost', main='Multiple Counters with Random Assignment')

# Minimum of the costs
min(totalCost)

## [1] 15089.52

# Counter configuration which has the lowest cost
which.min(totalCost)

## [1] 15

# Mark the line in the plot
abline(v = which.min(totalCost) ,lwd=3, col="red")
```



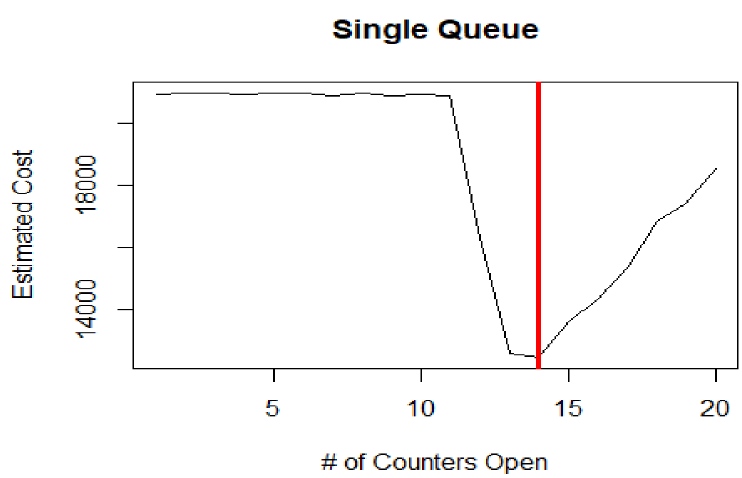
When there is an option to open as many counters as we want between 1 and 20 to tackle the Black Friday sale, assuming that the customer joins any queue randomly, with 10,000 customers, we see that 15 is the ideal number of counters that should be kept open to reduce the costs as much as possible. As we saw in the first part, the inter-arrival and wait times are modeled using a gamma distribution.

Question - 3

Next, we will try another queuing configuration, where all queues are combined and made into one queue. All customers join this queue and then move to any counter that opens up. Again, use a simulation to find the optimal number of checkout counters to keep open in this configuration. What is the expected cost (salary + penalty)?

```
# Assigning the number of simulations which refers to the number of customers
walking into the store
N = 10000

# Vector to hold the cost with different counters
totalCost = rep(NA, 20)

# Using gamma distribution to simulate the arrival and service times
for (i in 1:20)
{
  gamma_inter = rgamma(n=N-1, shape=g_fit_i$estimate[1], rate=g_fit_i$estimat
e[2])
  gamma_arrival = c(0, cumsum(gamma_inter))
  gamma_service = rgamma(n=N, shape=g_fit_s$estimate[1], rate=g_fit_s$estimat
e[2])

  D = rep(0, i) # vector to hold service end times
  W = rep(NA, N) # wait times
  T = 0 # service start time
  D[1] = T + gamma_service[1] # first service end time = start time + first s
ervice time
  W[1] = 0 # first wait time is 0

  for (p in 2:N) # Loop through the different customers
  {
    x = which.min(D) # assign each customer the counter with the least servic
e end time
    old_D = D[x] # counter's previous service end time
    T = max(old_D, gamma_arrival[p])

# service start time = max of customer arrival time and old_D
    D[x] = T + gamma_service[p]

    W[p] = max((old_D - gamma_arrival[p]), 0) # if the customer arrives after
the previous service end time, the wait time is 0
  }
  totalCost[i] = sum(W > 10) + (max(D)/60 * 40 * i)
}

# Total cost for different counters
totalCost
```

```
## [1] 20915.47 20953.56 20972.87 20920.65 20952.36 20968.24 20863.99
## [8] 20971.70 20877.41 20923.01 20883.40 16342.43 12612.59 12476.29
## [15] 13626.64 14339.81 15369.59 16838.85 17443.31 18546.82

# Plot the individual cost for different counters
plot(1:20, totalCost, type='line', xlab='# of Counters Open', ylab='Estimated Cost', main='Single Queue')

# Minimum of the costs
min(totalCost)

## [1] 12476.29

# Counter configuration which has the lowest cost
which.min(totalCost)

## [1] 14

# Mark the line in the plot
abline(v = which.min(totalCost) ,lwd=3, col="red")
```



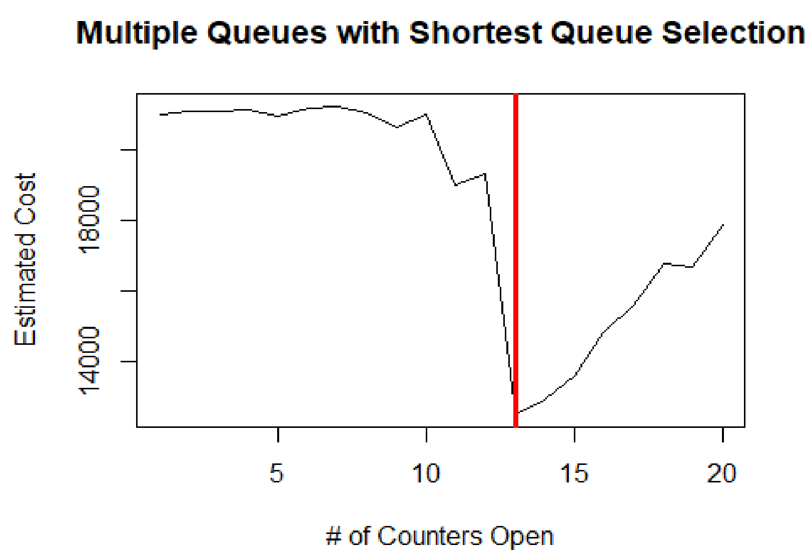
The second scenario here is similar to self-checkout counters in H-E-B. Every customer joins one queue, and then whenever a counter opens, the customer next in line joins the queue. Solving for this method, we see that 14 is the ideal number of counters which should be set up. This problem was solved by simulating for 10,000 customers.

Question - 4

(Extra credit) Now we will consider the queuing configuration in 2 where each counter has its own queue, but arriving customers join the queue that has the shortest queue length. Use a simulation to find the optimal number of checkout counters to keep open in this configuration. What is the expected cost (pay + penalty)?

```r
# Assigning the number of simulations which refers to the number of customers
walking into the store
N = 10000

# Vector to hold the cost with different counters
totalCost = rep(NA, 20)

# Using gamma distribution to simulate the arrival and service times
for (i in 1:20)
{
  gamma_inter = rgamma(n=N-1, shape=g_fit_i$estimate[1], rate=g_fit_i$estimat
e[2])
  gamma_arrival = c(0, cumsum(gamma_inter))
  gamma_service = rgamma(n=N, shape=g_fit_s$estimate[1], rate=g_fit_s$estimat
e[2])

  queueLength = rep(0,i)

# length of queues for each of the counters
  D = rep(0,i)

# service end times
  queueLength[1] = 1
  W = rep(NA,N)
  T = 0
  D[1] = T + gamma_service[1]
  W[1] = 0

  for (p in 2:N)
  {
    queueLength[(D<=gamma_arrival[p]) & (queueLength!=0)] = queueLength[(D<=g
amma_arrival[p]) & (queueLength!=0)] - 1

# if a customer arrives after the service end times of some counters, then qu
eueLength decreases by 1
    x = which.min(queueLength)

# customer chooses queue with shortest length
    queueLength[x] = queueLength[x]+1

# after customer joins the line, queue Length increases by 1
    T = max(D[x], gamma_arrival[p])
    old_D = D[x]
    D[x] = T + gamma_service[p]

    W[p] = max((old_D - gamma_arrival[p]),0)

  }
  totalCost[i] = sum(W > 10) + (max(D)/60 * 40 * i)
}
```

```r
# Total cost for different counters
totalCost

##  [1] 20997.14 21084.22 21096.41 21112.37 20938.41 21163.47 21227.21
##  [8] 21048.03 20623.40 20984.71 18993.21 19295.98 12481.65 12883.21
## [15] 13562.51 14783.07 15566.41 16762.72 16687.74 17863.59

# Plot the individual cost for different counters
plot(1:20, totalCost, type='line', xlab='# of Counters Open', ylab='Estimated
Cost', main='Multiple Queues with Shortest Queue Selection')


# Minimum of the costs
min(totalCost)

## [1] 12481.65

# Counter configuration which has the lowest cost
which.min(totalCost)

## [1] 13

# Mark the line in the plot
abline(v = which.min(totalCost) ,lwd=3, col="red")
```



The third scenario here is by far the most common situation that we face in grocery stores. There are multiple counters, and the customers join the queue which has the smallest number of customers. Solving for this method, we see that 13 is again the ideal number of counters which should be set up. This problem was solved by simulating for 10,000 customers.