Simulation Home Work - 1

Sekhar Mekala

Monday, September 05, 2016

Problem 1.1

System: Cafeteria

Entities: Customers, Chefs, Checkout counters

Attributes: For Customer entity - “Number of items ordered”, For chef - “Experience, Average time to complete the task”

Activities: Placing order, Preparing order, Processing payment

Events: Customer Arrival; Selection of items; Customer departure

State Variables: Number of busy chefs, Number of busy checkout counters, Number of customers waiting to place the order, Number of customers waiting for chef to finish the order, Number of customers waiting to pay

---

System: Grocery Store

Entities: Customers, Checkout counters

Attributes: For Customer entity - “Number of items purchased”

Activities: Making payment for the items purchased

Events: Customer Arrival; Selection of items; Customer departure

State Variables: Number of busy checkout counters, Number of customers waiting in the queue at the cash counter, Number of customers in the store selecting items (Customers not at waiting at the cash counters)

System: Laundromart

Entities: Customers, washing Machines, Dryer

Attributes: For Customer entity - “Number of washing machines used, Number of driers used”

Activities: Making payment to use machines

Events: Customer Arrival; Customer waiting; Customer departure

State Variables: Number of busy washing machines, Number of busy driers, Number of customers waiting in the queue for washing machine(s), Number of customers waiting for dryer(s) to finish the drying

---

System: Fast Food

Entities: Customers, Servers

Attributes: For Customer entity - “Number of items ordered”

Activities: Making payment, Placing order

Events: Customer Arrival; Customer waiting in the queue; Customer getting served; Customer making payment; Customer departure

State Variables: Number of busy servers, Number of Customers waiting to order, Number of currently ordering, Number of customers waiting pay

---

System: Hospital Emergency room

Entities: Patients

Attributes: Patients symptoms/problems

Activities: Filling out insurance details

Events: Patient Arrival; Patient waiting; Patient getting served; Patient departure

State Variables: Number of patients waiting; Number of patients currently getting the treatment

---

System: Taxi cab company with 10 taxis

Entities: Customers

Attributes: Amount paid, from location, to location

Activities: Customer calling for a taxi

Events: Customer boarding the taxi, Customer departing the taxi

State Variables: Number of customers currently waiting for the cab, Number of customers who are riding the cab

---

System: Automobile Assembly line

Entities: Workers

Attributes: Workers skill level

Activities: Worker assembling the units

Events: Worker Staring the assembly process, Finishing the assembly process

State Variables: Number of busy assembly lines, Number of busy workers

Problem 2.1

The inter arrival times probability distribution is given below:

Time between arrivals (Hours)	Probability
0	0.23
1	0.37
2	0.28
3	0.12

Given that the processing times for jobs are normally distributed with mean as 50 minutes and standard deviation as 8 minutes. We will convert all the time units to minutes.

Let us construct a simulation table. We are asked to assume that there are already 2 jobs, the first job will be finished in 25 minutes and the next job will be need 50 minutes to process. Also when we generate the random observations, we will round the time to nearest integer. It is assumed that the 2 existing jobs in the system have arrived at the same time.

We will write the following R function $Queue\_Simulation(Max\_Number)$, which generates random samples based on the requirement. The parameter Max_Number will control the number of observations to be generated. Since it is given that 2 customers are already in the system, the number of observations generated will be $Max\_Number + 2$. The function will return a data frame, with the following attributes:

- Customer_Number - Unique Customer Number
- Arrival_Time - Arrival time of the customer (Clock time)
- Inter_Arrival_Times - Difference between the successive arrival times (Minutes)
- Processing_Time - Time needed to process the job (Minutes)
- Service_Begin_Time - Service begin time (Clock time)
- Service_End_Time - Service end time (Clock time)
- Queue_Waiting_Time - Waiting time in the queue (Minutes)
- Total_System_Time - Total time spent by the customer (Minutes)
- Server_Idle_Time - Time idle by the server before serving the current customer (Minutes)

```
Queue_Simulation <- function(Max_Number)
{
        #Generate Inter arrival times
        Inter_Arrival_Time <- sample((0:3)*60,replace=TRUE,
                                     size=Max_Number,prob=c(0.23,0.37,0.28,0.12))

        #Include the 2 jobs which are already waiting in the queue
        Inter_Arrival_Time <- c(c(0,0),
                                Inter_Arrival_Time)

        #Generate Inter Processing times
        Processing_Time <- round(rnorm(Max_Number,mean=50,sd=8))

        #Include the 2 jobs processing times, which are already waiting in the queue
        Processing_Time <- c(c(25,50),Processing_Time)

        #Compute arrival times
        Arrival_Time <- cumsum(Inter_Arrival_Time)

        #Compute Service begin and end times
        Service_Begin_Time <- vector(length=12)
        Service_End_Time <- vector(length=12)
        Service_Begin_Time[1] <- 0
        Service_End_Time[1] <- Processing_Time[1]

        for (i in 1:(Max_Number + 2 - 1))
        {

          if(Arrival_Time[i+1] > Service_End_Time[i])
          {
             Service_Begin_Time[i+1] <- Arrival_Time[i+1]
```

```
            Service_End_Time[i+1] <- Service_Begin_Time[i+1] +
                                     Processing_Time[i+1] - 1
          }
          else
          {
            Service_Begin_Time[i+1] <- Service_End_Time[i]+1
            Service_End_Time[i+1] <- Service_Begin_Time[i+1] +
                                     Processing_Time[i+1] - 1
          }
        }

        #Compute the waiting time in the queue
        Queue_Waiting_Time <- ifelse(Service_Begin_Time == Arrival_Time,
                                     0, Service_Begin_Time - Arrival_Time-1)

        #Compute the total time spent in the system
        Total_System_Time <- Queue_Waiting_Time + Processing_Time

        Temp <-  Service_Begin_Time[-1] - Service_End_Time[-12]

        #Compute server idle time
        Server_Idle_Time <- c(0,ifelse(Temp == 1, 0, Temp-1))

        #put everything in a data frame
        df <- data.frame(Arrival_Time,
                         Inter_Arrival_Time,Processing_Time,
                         Service_Begin_Time, Service_End_Time,
                         Queue_Waiting_Time,
                         Total_System_Time,Server_Idle_Time)

return(df)
}
```

Let us call the function Queue_Simulation to generate 10 new observations (note that the function will return 12 observations, since we already have 2 jobs in the system, as per the problem).

```r
library(knitr)
library(pander)

df <- Queue_Simulation(10)
pander(df, split.table = 120, style = 'rmarkdown')
```

Table 2: Table continues below

Arrival_Time	Inter_Arrival_Time	Processing_Time	Service_Begin_Time	Service_End_Time
0	0	25	0	25
0	0	50	26	75
0	0	54	76	129
60	60	53	130	182
120	60	57	183	239
120	0	55	240	294
120	0	49	295	343
240	120	35	344	378
300	60	41	379	419
420	120	44	420	463
480	60	51	480	530
540	60	45	540	584

Queue_Waiting_Time	Total_System_Time	Server_Idle_Time
0	25	0
25	75	0
75	129	0
69	122	0
62	119	0
119	174	0
174	223	0
103	138	0
78	119	0
0	44	0
0	51	16
0	45	9

(a). What is the average waiting time in the queue for the 10 new jobs?

The average waiting time for the 10 new jobs is: 70.5 minutes

(b). What is the average processing time for the 10 new jobs?

The average processing time for the 10 new jobs is: 48.4 minutes

(c). What is the maximum time in the system for the 10 new jobs?

The maximum time in the system for the 10 new jobs is: 223 minutes

Problem 2.2

The probability distribution of the Number of customers/day is given below:

Number of Customers/day	Probability
8	0.35
10	0.3
12	0.25
14	0.1

The probability distribution for customer orders is given below:

Number of Dozen ordered/Customer	Probability
1	0.4
2	0.3
3	0.2
4	0.1

Let us assume that 1 unit = 12 bagels

Selling price per unit = $8.40

Cost per unit = $5.80

Selling price of left overs per unit = $4.20

Profit = (Selling price per unit X Number of units bought by customers) - (Cost per unit X Number of units prepared) + (Selling price of left overs per unit X Number of left over units)

Let us write an R function $Demand\_Gen(Days, n)$ that takes Days and number of units prepared as inputs and generates the demand for the specified number of days (function's parameter), calculates the profit made and profit lost (not able to meet the demand). The function will return a data frame with the following variables:

- Day - Number of the Day
- C1, C2, C3 … C14 - Customers 1,2,3 … 14
- Profit_made - Profit made
- Profit_lost - Profit lost, due to not meeting the demand

```
#Day <- 1:10
Demand_Gen <- function(Days, n)
  {
  Day <- 1:Days
Number_of_Customers <- sample(c(8,10,12,14),replace=TRUE,size=length(Day),
                              prob=c(.35,.3,.25,.1))

df <- data.frame(Day,C1=NA,C2=NA,C3=NA,
```

```
C4=NA,C5=NA,C6=NA,
C7=NA,C8=NA,C9=NA,
C10=NA,C11=NA,C12=NA,
C13=NA,C14=NA)



for (i in 1:length(Day))
{
  df[i,2:(Number_of_Customers[i]+1)] <-
    sample(1:4,replace=TRUE,
           size=Number_of_Customers[i],prob=c(.4,.3,.2,.1))

}

profit_made <- vector(length=Days)
profit_lost <- vector(length=Days)

for(i in 1:Days)
  {
    demand <- sum(df[i,2:14],na.rm=TRUE)

    sold <- ifelse(demand > n, n, demand)

    profit_made[i] <-
      sold * 8.4 - n * 5.8 +
      ifelse(n > sold, (n-sold) * 4.2, 0)

    profit_lost[i] <-
      ifelse(demand > n, (demand-n)* 8.4, 0)

  }

df$profit_made <- profit_made
df$profit_lost <- profit_lost

return(df)
}
```

Let us call the function to generate the data for 5 days. We will assume that the baker has prepared 20 units (1 unit = 12 bagels), and the profit_made, profit_lost columns represent the profit details if the baker has prepared 20 units each day

```
df <- Demand_Gen(5,20)
kable(df)
```

Day	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	profit_made	profit_lost
1	1	4	1	1	2	2	3	2	NA	NA	NA	NA	NA	NA	35.2	0
2	1	2	2	2	4	1	1	4	2	2	1	3	NA	NA	52.0	42
3	1	1	3	2	1	2	1	1	3	1	1	3	NA	NA	52.0	0
4	3	3	1	4	2	3	1	1	NA	NA	NA	NA	NA	NA	43.6	0
5	4	4	1	1	2	1	3	1	2	3	1	2	NA	NA	52.0	42

In the above table C1, C2, C3…C14 columns represent the number of units bought by each customer on a specific day. If NA is listed, then it implies that the customer has not visited the store.

To find the optimal number of units, we need to repeat this experiment many times, and find the average profit_made, and average profit_lost and pick a number that maximizes the profit_made and minimizes the profit lost.Let us repeat the 5 days demand for 100 times for each of the Units between 5 to 50 (multiple of 5)

```
#Create the place holder vectors
avg_profit_made <- vector()
avg_profit_lost <- vector()

#Index counter
k <- 1

for(i in seq(from=5,to=50,by=5))
{

    #Temporary vectors
    profit_made_temp <- vector()
    profit_lost_temp <- vector()

       for(j in 1:100)
       {
        df <- Demand_Gen(5,i)

      profit_made_temp <- c(profit_made_temp,df$profit_made)
      profit_lost_temp <- c(profit_lost_temp,df$profit_lost)


       }
    #Calculate the average profit
    avg_profit_made[k] <- sum(profit_made_temp)/500
    avg_profit_lost[k] <- sum(profit_lost_temp)/500



  k<- (k+1)

}

df <- data.frame(Units=seq(from=5,to=50, by=5),
                 Average_profit_made = avg_profit_made,
                 Average_profit_lost=avg_profit_lost)

df1 <- df

library(ggplot2)
library(reshape)

df <- melt(df,id="Units")

ggplot(df,aes(x=Units,y=value,color=variable))+
  geom_point(size=3)+
  geom_line()+
  labs(title="Avg profit made vs Avg profit lost",
```

```r
       x="Dozens of bagels to be made",y="Profit")
```


```r
kable(df1)
```

Units	Average_profit_made	Average_profit_lost
5	13.0000	125.2104
10	25.9916	85.7304
15	38.2272	45.6120
20	44.0872	18.4296
25	43.7900	4.2000
30	36.1428	0.4704
35	28.4200	0.0000
40	20.8568	0.0000
45	13.5372	0.0000
50	4.4788	0.0000

Clearly the graph shows that the baker has to prepare 20 units (or 240 Bagels or 20 Dozens) daily to meet the demand, and maximize the profit. If 20 dozens of bagels are made, the baker can make an average profit of \$44.0872. However, at 20 dozens per day, the baker will not be able to meet the complete demand (see the average profit lost due to not meeting the demand, is \$18.4296 approximately. This resembles that more number of customers returning back empty hands resulting in unsatisfied customers). So if customer satisfaction is important, then the baker should make 25 dozens of bagels, so that only an average of \$4.2 is lost due to not meeting the demand (or much lesser number of unsatisfied customers).

Problem 2.4

Given the following probability distribution for the Time between calls:

Time between calls (Minutes)	Probability
15	0.14
20	0.22
25	0.43
30	0.17
35	0.04

The probability distribution for the service time is given below:

Service time (Minutes)	Probability
5	0.12
15	0.35
25	0.43
35	0.06
45	0.04

It is given that the taxi operates between 9AM to 5PM. So we assume that if the call is received between this time, then the taxi service is provided (even though the service time extends beyond 5PM. For instance if a call is received at 5:00PM for a service time of 25 minutes, the service is still provided. However, if the call is received at 5:01PM, then the service is not provided)

Let us create a function that generates 5 days of data:

```
Gen_Taxi_Calls <- function(Days)
{

#Initialize the data frame to empty.
df <- data.frame()

#Generate the data for each day
  for(a in 1:Days)
    {

    #Inter arrival data
    Inter_Arrival_Calls <- sample(c(15,20,25,30,35),
                                  replace=TRUE,size=32,
                                  prob=c(0.14,.22,.43,.17,.04))

    #arrival Clock Time
    Call_Received_Time <- cumsum(Inter_Arrival_Calls)

    #We will remove all the calls that are
    #received after 480 minutes (8 hours interval, from 9 AM)

    Call_Received_Time <- Call_Received_Time[Call_Received_Time <= 480]

    Inter_Arrival_Calls <- Inter_Arrival_Calls[1:length(Call_Received_Time)]
```

```
    #Code to find the service begin time, end time,
    #customer wait time, overall system time
    Service_Time <- sample(c(5,15,25,35,45),replace=TRUE,
                           size=length(Inter_Arrival_Calls),
                           prob=c(.12,.35,.43,.06,.04))

    Service_Begin_Time <- vector(length=length(Inter_Arrival_Calls))
    Service_End_Time <- vector(length=length(Inter_Arrival_Calls))
    Service_Begin_Time[1] <- Call_Received_Time[1]
    Service_End_Time[1] <- Call_Received_Time[1] + Service_Time[1] - 1

            for (i in 1:(length(Inter_Arrival_Calls) - 1))
            {

              if(Call_Received_Time[i+1] > Service_End_Time[i])
              {
                 Service_Begin_Time[i+1] <- Call_Received_Time[i+1]
                 Service_End_Time[i+1] <- Service_Begin_Time[i+1] + Service_Time[i+1] - 1
              }
              else
              {
                 Service_Begin_Time[i+1] <- Service_End_Time[i]+1
                 Service_End_Time[i+1] <- Service_Begin_Time[i+1] + Service_Time[i+1] - 1
              }
            }


            Customer_Waiting_Time <- ifelse(
                                          Service_Begin_Time == Call_Received_Time,
                                          0, Service_Begin_Time - Call_Received_Time
                                          )

            Total_System_Time <- Customer_Waiting_Time + Service_Time

            Temp <-  Service_Begin_Time[-1] - Service_End_Time[-length(Inter_Arrival_Calls)]

            Taxi_1_Idle_Time <- c(Call_Received_Time[1] - 1,ifelse(Temp == 1, 0, Temp-1))

        Day <- rep(a,length(Inter_Arrival_Calls))

    df <- rbind(df,data.frame(Day,Call_Received_Time,
                              Inter_Arrival_Calls,
                              Service_Time,Service_Begin_Time,
                              Service_End_Time,
                              Customer_Waiting_Time,Taxi_1_Idle_Time))

    }


    return(df)

}
```

Let us generate 5 days of data using the function defined above. We will display only some data (day 1 and 2), since complete data display will clutter the document.

```
df <- Gen_Taxi_Calls(5)
pander(df[df$Day==1 | df$Day == 2,], split.table = 120, style = 'rmarkdown')
```

Table 10: Table continues below

Day	Call_Received_Time	Inter_Arrival_Calls	Service_Time	Service_Begin_Time
1	30	30	25	30
1	45	15	25	55
1	70	25	35	80
1	100	30	5	115
1	130	30	25	130
1	155	25	15	155
1	180	25	15	180
1	210	30	5	210
1	235	25	25	235
1	260	25	15	260
1	275	15	25	275
1	300	25	15	300
1	330	30	5	330
1	355	25	25	355
1	380	25	25	380
1	400	20	45	405
1	420	20	15	450
1	445	25	5	465
1	470	25	25	470
2	30	30	25	30
2	60	30	35	60
2	80	20	35	95
2	105	25	5	130
2	120	15	25	135
2	140	20	25	160
2	160	20	25	185
2	185	25	15	210
2	210	25	45	225
2	235	25	25	270
2	255	20	25	295
2	275	20	15	320
2	300	25	15	335
2	325	25	15	350
2	350	25	35	365
2	370	20	15	400
2	395	25	15	415
2	420	25	15	430
2	445	25	25	445
2	480	35	15	480

Service_End_Time	Customer_Waiting_Time	Taxi_1_Idle_Time
54	0	29

Service_End_Time	Customer_Waiting_Time	Taxi_1_Idle_Time
79	10	0
114	10	0
119	15	0
154	0	10
169	0	0
194	0	10
214	0	15
259	0	20
274	0	0
299	0	0
314	0	0
334	0	15
379	0	20
404	0	0
449	5	0
464	30	0
469	20	0
494	0	0
54	0	29
94	0	5
129	15	0
134	25	0
159	15	0
184	20	0
209	25	0
224	25	0
269	15	0
294	35	0
319	40	0
334	45	0
349	35	0
364	25	0
399	15	0
414	30	0
429	20	0
444	10	0
469	0	0
494	0	10

Let us find the average customer waiting time, and average taxi idle time.

Average customer waiting time = 12.3958333 minutes

Average taxi idle time = 4.3229167 minutes

Let us simulate the scenario, if we have two taxis.

The following R code will generate 5 days of data, for the 2 taxi scenario

```
Gen_Taxi_Calls_2 <- function(Days)
{

#Empty data frame
```

```r
df <- data.frame()

  for(a in 1:Days)
    {
    #set the seed
    set.seed(1)
    Inter_Arrival_Calls <- sample(c(15,20,25,30,35),
                                  replace=TRUE,size=32,prob=c(0.14,.22,.43,.17,.04))

    Call_Received_Time <- cumsum(Inter_Arrival_Calls)

    #We will remove all the calls that are
    #received after 480 minutes (8 hours interval, from 9 AM)

    Call_Received_Time <- Call_Received_Time[Call_Received_Time <= 480]

    Inter_Arrival_Calls <- Inter_Arrival_Calls[1:length(Call_Received_Time)]

    Service_Time <- sample(c(5,15,25,35,45),
                           replace=TRUE,size=length(Inter_Arrival_Calls),
                           prob=c(.12,.35,.43,.06,.04))


    T1_Service_Begin_Time <- vector(length=length(Inter_Arrival_Calls))
    T1_Service_End_Time <- vector(length=length(Inter_Arrival_Calls))

    #T1_Service_Begin_Time <- NA
    #T1_Service_End_Time <- NA

    T1_Service_Begin_Time[1] <- Call_Received_Time[1]
    T1_Service_End_Time[1] <- Call_Received_Time[1] + Service_Time[1] - 1

    T2_Service_Begin_Time <- vector(length=length(Inter_Arrival_Calls))
    T2_Service_End_Time <- vector(length=length(Inter_Arrival_Calls))


for (i in 1:(length(Inter_Arrival_Calls) - 1))
            {

              if(Call_Received_Time[i+1] > T1_Service_End_Time[i])
              {
                 T1_Service_Begin_Time[i+1] <- Call_Received_Time[i+1]
                 T1_Service_End_Time[i+1] <- T1_Service_Begin_Time[i+1] +
                                             Service_Time[i+1] - 1
              }

              else
              {

                  if(Call_Received_Time[i+1] > T2_Service_End_Time[i])
                  {
                     T2_Service_Begin_Time[i+1] <- Call_Received_Time[i+1]
                     T2_Service_End_Time[i+1] <- T2_Service_Begin_Time[i+1] +
```

```
                                                  Service_Time[i+1] - 1
              }


              else
              {
               if(T1_Service_End_Time[i] <= T2_Service_End_Time[i])
                 {
                   T1_Service_Begin_Time[i+1] <- T1_Service_End_Time[i]+1
                   T1_Service_End_Time[i+1] <- T1_Service_Begin_Time[i+1] +
                                                 Service_Time[i+1] - 1
                 }
               else
                 {
                   T2_Service_Begin_Time[i+1] <- T2_Service_End_Time[i]+1
                   T2_Service_End_Time[i+1] <- T2_Service_Begin_Time[i+1] +
                                                 Service_Time[i+1] - 1

                 }
              }
          }
        }


Customer_Waiting_Time <-
        ifelse(T1_Service_Begin_Time == Call_Received_Time |
        T2_Service_Begin_Time == Call_Received_Time, 0,
        ifelse(T1_Service_Begin_Time > T2_Service_Begin_Time,
        T2_Service_Begin_Time,T1_Service_Begin_Time) -
        Call_Received_Time)

Total_System_Time <- Customer_Waiting_Time + Service_Time

t <- which(T1_Service_Begin_Time == 0 & T1_Service_End_Time == 0)
Taxi_1_Idle_Time <- vector(length=length(Inter_Arrival_Calls))

Temp_1 <- T1_Service_Begin_Time[-t]

Temp_2 <- T1_Service_End_Time[-t]

Temp <-  Temp_1[-1] - Temp_2[-length(Temp_2)]

Taxi_1_Idle_Time <- vector(length=length(Inter_Arrival_Calls))
Taxi_1_Idle_Time[t] <- NA
Taxi_1_Idle_Time[c(-t,-1)] <- Temp - 1
Taxi_1_Idle_Time[1] <- Call_Received_Time[1] - 1

t <- which(T2_Service_Begin_Time == 0 & T2_Service_End_Time == 0)
t1 <- min(which(t[-1] - t[-length(t)] != 1))

Taxi_2_Idle_Time <- vector(length=length(Inter_Arrival_Calls))

Taxi_2_Idle_Time[1:t1] <- NA
```

```
            if((t1+1) < length(Taxi_2_Idle_Time))
              {
              Temp_1 <- T2_Service_Begin_Time[(t1+1):length(T2_Service_Begin_Time)]


              Temp_2 <- T2_Service_End_Time[(t1+1):length(T2_Service_Begin_Time)]

              t2 <- which(Temp_1 == 0 & Temp_2 == 0)
              Temp_1_1 <- Temp_1[-t2]
              Temp_2_2 <- Temp_2[-t2]






            Taxi_2_Idle_Time[t2+t1] <- NA
            Taxi_2_Idle_Time[(c(-(t2+t1),-(1:t1),-(t1+1)))] <-
                            (Temp_1_1[-1] - Temp_2_2[-length(Temp_2_2)] - 1)
            Taxi_2_Idle_Time[t1+1] <- Call_Received_Time[t1+1] - 1

              }


        Day <- rep(a,length(Inter_Arrival_Calls))


T1_Service_Begin_Time[which(T1_Service_Begin_Time==0)] <- NA
T2_Service_Begin_Time[which(T2_Service_Begin_Time==0)] <- NA

T1_Service_End_Time[which(T1_Service_End_Time==0)] <- NA
T2_Service_End_Time[which(T2_Service_End_Time==0)] <- NA


df <- rbind(df,
            data.frame(Day,
                       Call_Received_Time,
                       Inter_Arrival_Calls,
                       Service_Time,
                       T1_Service_Begin_Time,
                       T1_Service_End_Time,
                       T2_Service_Begin_Time,
                       T2_Service_End_Time
               , Taxi_1_Idle_Time
              ,Taxi_2_Idle_Time, Customer_Waiting_Time, Total_System_Time)
)

    }

    return(df)

}
```

Let us display some data.

```r
df <- Gen_Taxi_Calls_2(5)
pander(head(df), split.table = 80, style = 'rmarkdown')
```

Table 12: Table continues below

Day	Call_Received_Time	Inter_Arrival_Calls	Service_Time
1	25	25	15
1	50	25	25
1	70	20	5
1	85	15	15
1	110	25	5
1	125	15	25

Table 13: Table continues below

T1_Service_Begin_Time	T1_Service_End_Time	T2_Service_Begin_Time
25	39	NA
50	74	NA
NA	NA	70
85	99	NA
110	114	NA
125	149	NA

Table 14: Table continues below

T2_Service_End_Time	Taxi_1_Idle_Time	Taxi_2_Idle_Time
NA	24	NA
NA	10	NA
74	NA	69
NA	10	NA
NA	10	NA
NA	10	NA

Customer_Waiting_Time	Total_System_Time
0	15
0	25
0	5
0	15
0	5
0	25

```r
l1 <- length(df$Taxi_1_Idle_Time != NA)
l2 <- length(df$Taxi_2_Idle_Time != NA)
```

Average customer waiing time = 0 minutes

Average taxi-1 idle time = 12.5789474 minutes

Average taxi-2 idle time = 18.1052632 minutes

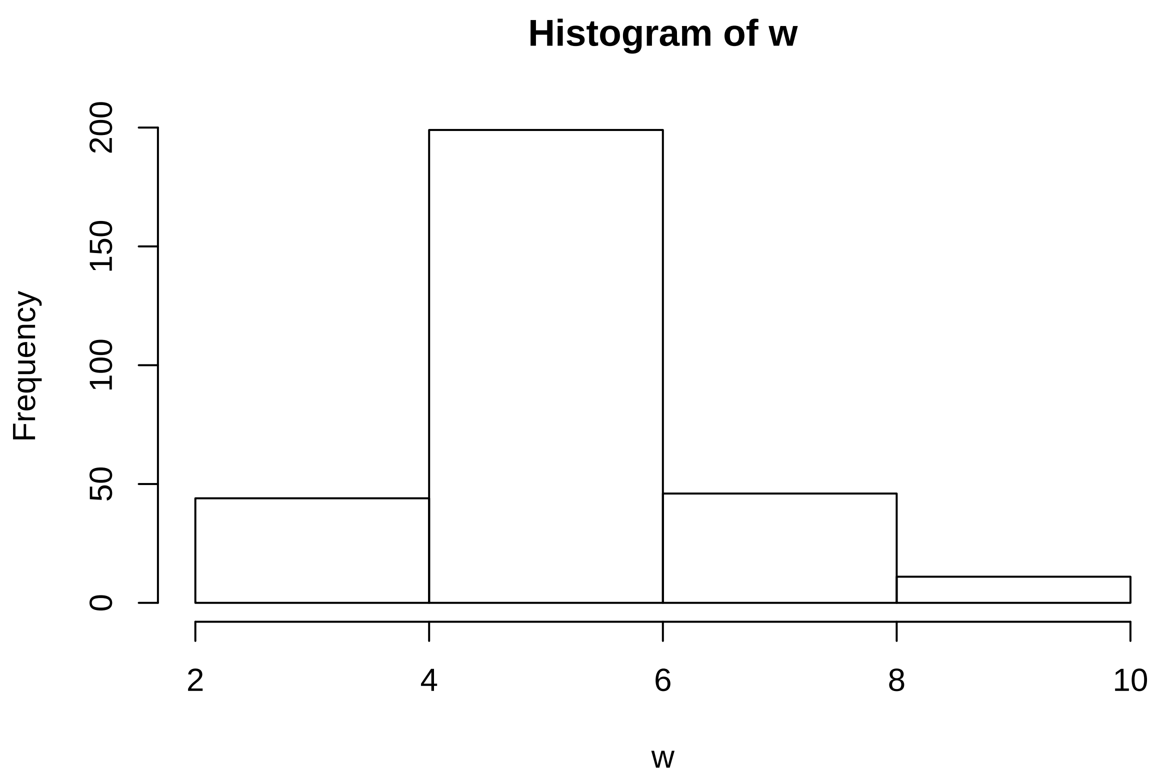
Therefore, if a new taxi is added, then the customer wait time is zero. But the average taxi-1 idle time is 12.57 minutes/day (increased from appx 5 minutes/day, for a single taxi system). It is suggested to add a new taxi, since the customer wait time has literally reached zero minutes.

Problem 2.5

The following R code will get the random numbers, as per the requirement, and plots a histogram

```
x <- rnorm(50, mean= 100, sd=10)
y <- rnorm(300, mean= 100, sd=15)
z <- rnorm(50, mean= 40, sd=8)

w <- (x + y)/z
hist(w,breaks=4)
```



Problem 2.7

In this problem, we are asked to generate 7 weeks of demand data (as per the constraints given in the problem). The following R code will generate the 35 days (or 7 weeks) of demand data. We assume that the inventory is checked daily, at the end of the day, and if the available number of units are less than or equal to

10, then an order is placed to make the inventory to 20 units. Th eorder placed can arrive immediately or within 5 days (following uniform probability distribution).

```
#Set the seed
set.seed(1)
#Get the demand data for 35 days

demand <- rnorm(35,mean=5,sd=1.5)

demand <- round(demand)



#Prepare the 1st day stock (beg_stk[] array contains the
#inventory at the beginning of the day)

beg_stk <- vector(length=35)

beg_stk[1] <- 18



#End of the day stock array

end_stk <- vector(length = 35)

end_stk[1] <- beg_stk[1] - demand[1]



#This array contains the order placement information

place_order <- vector(length=35)

place_order[1:35] <- 'N'



#This array contains the ordered quantity

ordered_qty <- rep(NA,35)



#Assign -1s for all time to arrive elements

time_to_arrive <- rep(-1,35)



#Variable to control the reorder logic

#It is assumed that:
```

```
# Every evening the stock is checked.
# If the stock is less than 20 units, then an order is placed
# The order is delivered in the morning hours
# If the delivered order places the end of day inventory less than 20,
# another order is made in the evenng. Also some orders are delivered
# on the same day. And we need to reorder the inventory, if the end of the day
# inventory falls below 20 units
reorder <- 0

#Variable to check if the order has been placed today
ordered_today <- 0
order_placed <- 0

#Set the default lead time variable to -1
lead_time <- -1

for(i in 1:34)
{
  #Set/Reset reorder variable to 0
  reorder <- 0

  #The following loop is repeated until no need to reorder
  repeat{

  ordered_today <- 0

  if(reorder == 0)
```

```
  {
end_stk[i] <- beg_stk[i] - demand[i]
}


if(end_stk[i] <= 10 & order_placed == 0)
{
  reorder <- 0
  lead_time <- sample(0:5,1)
  time_to_arrive[i] <- lead_time
  order_qty <- 20 - ifelse(end_stk[i]<=0,0,end_stk[i])
  ordered_qty[i] <- order_qty

  place_order[i] <- 'Y'

  if(lead_time == 0)
  {
    end_stk[i] <- beg_stk[i] - demand[i] + order_qty
    order_placed <- 0
    lead_time <- -1
    time_to_arrive[i] <- 0

    if(end_stk[i] <= 10)
    {
      reorder <- 1
      next
    }
```

```
  }
  else
  {
    order_placed <- 1
  }
  ordered_today <- 1

}



if(lead_time > 0 & ordered_today == 0)
{
  lead_time <- (lead_time - 1)
  time_to_arrive[i] <- lead_time
}

if(lead_time == 0 & ordered_today == 0)
{
  end_stk[i] <- beg_stk[i] - demand[i] #+ order_qty
  lead_time <- -1
  time_to_arrive[i] <- lead_time
  order_placed <- 0

  if(end_stk[i] <= 10)
  {
    reorder <- 1
    next
```

```
    }

  }



  beg_stk[i+1] <- ifelse(end_stk[i]<= 0, 0, end_stk[i])



  if(time_to_arrive[i] == 1)

  {

    beg_stk[i+1] = beg_stk[i+1] + order_qty

  }

  if(reorder == 0){break}

  }

}





#Trim the data for final display/analysis.

time_to_arrive[time_to_arrive == -1] <- NA

end_stk[35] <- beg_stk[35] - demand[35] +
  ifelse(lead_time==0,order_qty,0)

time_to_arrive[35] <- ifelse(lead_time > 0, (lead_time - 1),-1)

missed_demand <- end_stk

missed_demand[missed_demand > 0] <- NA

missed_demand <- missed_demand * -1



end_stk[end_stk<0] <- 0



#Create a data frame

df <- data.frame(Day=1:35,Begin_Stock = beg_stk,
           Demand=demand,
```

```
            End_Stock=end_stk,
            Order_Placed=place_order,
            Days_To_Arrive=time_to_arrive,
            Ordered_Qty=ordered_qty,
            Missed_Demand=missed_demand)

pander(df, split.table = 120, style = 'rmarkdown')
```

Day	Begin_Stock	Demand	End_Stock	Order_Placed	Days_To_Arrive	Ordered_Qty	Missed_Demand
1	18	4	14	N	NA	NA	NA
2	14	5	9	Y	2	11	NA
3	9	4	5	N	1	NA	NA
4	16	7	9	Y	5	11	NA
5	9	5	4	N	4	NA	NA
6	4	4	0	N	3	NA	0
7	0	6	0	N	2	NA	6
8	0	6	0	N	1	NA	6
9	11	6	5	Y	2	15	NA
10	5	5	0	N	1	NA	0
11	15	7	8	Y	2	12	NA
12	8	6	2	N	1	NA	NA
13	14	4	10	Y	2	10	NA
14	10	2	8	N	1	NA	NA
15	18	7	11	N	NA	NA	NA
16	11	5	6	Y	5	14	NA
17	6	5	1	N	4	NA	NA
18	1	6	0	N	3	NA	5
19	0	6	0	N	2	NA	6
20	0	6	0	N	1	NA	6
21	14	6	8	Y	5	12	NA
22	8	6	2	N	4	NA	NA
23	2	5	0	N	3	NA	3
24	0	2	0	N	2	NA	2
25	0	6	0	N	1	NA	6
26	12	5	7	Y	2	13	NA
27	7	5	2	N	1	NA	NA
28	15	3	12	N	NA	NA	NA
29	12	4	8	Y	4	12	NA
30	8	6	2	N	3	NA	NA
31	2	7	0	N	2	NA	5
32	0	5	0	N	1	NA	5
33	12	6	6	Y	5	14	NA
34	6	5	1	N	4	NA	NA
35	1	3	0	N	3	NA	2

The above generated data shows that, on the beginning of day 1, we have a stock of 18 units. With a demand of 4 units on day 1, the stock at the end of the day 1 is 14. Since this stock remained is greater than 10 units, no order has been placed (see the row 1 in column Order_Placed, which is set to N). On Day 2, the beginning stock is 14 units, and a demand of 5 units on day 2 made the end of the day 2 available stock as 9 units. Since the available stock is less than or equal to 10 uints, an order of 11 units were made (see Ordered_Qty on day 2). The days to get the order is 2 days. On day 9, an order of 15 units was made, and this order was delivered on day 11. But on Day 11, there was a demand of 7 units, and by day 11 end, the available stock is 8 units, and hence the order is placed again on Day 11 end of the day (although the previous order of 15 units is delivered on Day 11 morning). On day 33 end of the day, the available stock is 6 units, and an order has been placed (14 units). This order will be delivered on day 5, which crosses the number of days simulated. On day 35, we lost 2 units of demand (since we have only 1 available unit, while there was a demand of 33 units on day 35).

Let us find the demand lost due to stock unavailability.

Total demand lost due to unavailability of the stock = 52 units

Average demand lost per day = 1.4857143 units

Average demand lost per week = 10.4 units

Problem 2.8

Given that the elevator takes 3 minutes to take the material from 1st floor to second floor and unload the material. The elevator needs 1 minute to return back. Let us generate some sample data with the given probability distributions and constraints.

```
set.seed(19)

#Generate the arrival times for 3 types

A_Demand <- sample(3:7,replace=TRUE,size=30)

B_Demand <- rep(6,10)

C_Demand <- sample(2:3,size=30,prob=c(.33,.67),replace=TRUE)

#Combine the arrival data into a data frame

Arrival_Data <- rbind(data.frame(Arrival_time = cumsum(A_Demand),Item="A"),
                      data.frame(Arrival_time = cumsum(B_Demand),Item="B"),
                      data.frame(Arrival_time = cumsum(C_Demand),Item="C")
)

#Order the data as per the time
Arrival_Data <- Arrival_Data[order(Arrival_Data$Arrival_time),]

#Remove the orders which have arrived after 60 minutes
Arrival_Data <- Arrival_Data[Arrival_Data$Arrival_time<= 60,]

#Add the weight information
Weight <- vector(length=length(Arrival_Data))
Weight[Arrival_Data$Item == "A"] <- 200
Weight[Arrival_Data$Item == "B"] <- 100
Weight[Arrival_Data$Item == "C"] <- 50
Arrival_Data$Weight <- Weight

#Add a unique ID to each arrived box
id <- 1:nrow(Arrival_Data)
Arrival_Data <- cbind(as.data.frame(id),Arrival_Data)
```

```r
#Let us find the wait times of elevator, and of the boxes to be delivered
#We have to order the boxes as per their loaded time
#whenever a cumulative weight of 400 kg is reached, then the elevator
#is operated, else the elevator has to wait.
#For example, if we have 350Kg of weight already placed in the elevator,
#and if we get another 100 kg order, then the elevator will not place the
#100 kg order into the elevator. But it will wait till a 50Kg order is received.
#The waiting 100kg will be placed in the next elevator trip

#We will assume the arrived items in a box, and place the items
#into another box (or elevator), so that weight constraints are matched.


Weight <- Arrival_Data$Weight

len <- nrow(Arrival_Data)
Box_1 <- Arrival_Data
Weight_Assignment <- vector()

Type_Assignment <- vector()

Arrival_Time_Assignment <- vector()

ID_Assignment <- vector()

i <- 1

c <- 0

repeat{
  repeat{
    if(i > len) break
    else{
      if(Box_1$Weight[i] == 0)
      {
        i <- i+1
        if(i > len)
        {
          break
        }
      }
      else{
        break
      }
    }
  }
  if(i > len) break
  c <- c + Box_1$Weight[i]
  if(c < 400)
  {
    Weight_Assignment <- c(Weight_Assignment,Box_1$Weight[i])
    ID_Assignment <- c(ID_Assignment,Box_1$id[i])
    Type_Assignment <- c(Type_Assignment,Box_1$Item[i])
```

```
    Arrival_Time_Assignment <- c(Arrival_Time_Assignment,Box_1$Arrival_time[i])
    Box_1$Weight[i] <- 0
    i <- (i+1)
    next
  }

  if(c == 400)
  {
    Weight_Assignment <- c(Weight_Assignment,Box_1$Weight[i])
    Box_1$Weight[i] <- 0
    ID_Assignment <- c(ID_Assignment,Box_1$id[i])
    Type_Assignment <- c(Type_Assignment,Box_1$Item[i])
    Arrival_Time_Assignment <- c(Arrival_Time_Assignment,Box_1$Arrival_time[i])

    c <- 0

    i <- 1

    next

  }

  if(c > 400)
  {
    c <- (c - Box_1$Weight[i])
    i <- (i+1)
    if(i > len) {break}
    else {next}
  }
}

if(any(Box_1$Weight > 0))
{
  left_overs <- which(Box_1$Weight>0)
  Weight_Assignment <- c(Weight_Assignment,Box_1$Weight[left_overs])
  Box_1$Weight[left_overs] <- 0
  ID_Assignment <- c(ID_Assignment,Box_1$id[left_overs])
  Type_Assignment <- c(Type_Assignment,Box_1$Item[left_overs])
  Arrival_Time_Assignment <- c(Arrival_Time_Assignment,Box_1$Arrival_time[left_overs])
}



#Put everything in a data frame.
df <- data.frame(ID = ID_Assignment, Item_Type = Type_Assignment,
                 Weight = Weight_Assignment, Arrival_Time=Arrival_Time_Assignment)
df$C_Wt_Sum = cumsum(df$Weight)

#Find the item wait times due to elevator unavailability or
#as the cumulative weight is less than 400Kg

load_points <- which(df$C_Wt_Sum %% 400 == 0)
temp_load <- vector()
```

```r
if(length(load_points) >= 1)
{
  temp_load <- c(rep(df$Arrival_Time[load_points[1]],load_points[1]))
  temp_elev <- c(rep(0,load_points[1]))

  for(i in 2:length(load_points))
  {
    temp_load <- c(temp_load,rep(df$Arrival_Time[load_points[i]],load_points[i]-load_points[i-1]))
    temp_elev <- c(temp_elev,rep(df$Arrival_Time[load_points[i-1]] +4,load_points[i]-load_points[i-1]))
  }

}

temp_load <- c(temp_load,rep(NA,len - length(temp_load)))
temp_elev <- c(temp_elev,rep(NA,len - length(temp_elev)))

df$Wait_Weight <- temp_load - df$Arrival_Time

df$Wait_Elev <- temp_elev - df$Arrival_Time
df$Wait_time <- ifelse(df$Wait_Weight > df$Wait_Elev,df$Wait_Weight ,df$Wait_Elev)
df$Wait_for_Elev <- ifelse(df$Wait_Elev >= df$Wait_Weight, 'Yes', 'No')
df$Wait_for_Weight <- ifelse(df$Wait_Weight >= df$Wait_Elev, 'Yes', 'No')

#Final data frame
elevator <- data.frame(ID=df$ID,
                 Type=df$Item_Type,
                 Weight=df$Weight,
                 Arr_Time=df$Arrival_Time,
                 Wait_Time = df$Wait_time,
                Wait_for_Elev = df$Wait_for_Elev,
                 Wait_for_Weight=df$Wait_for_Weight

)

elevator$System_Time <- elevator$Wait_Time+3
elevator$Wait_for_Weight[elevator$Wait_Time==0] <- "No"
elevator$Wait_for_Elev[elevator$Wait_Time==0] <- "No"
elevator$Type[elevator$Type==1] <- 'A'
elevator$Type[elevator$Type==2] <- 'B'
elevator$Type[elevator$Type==3] <- 'C'
```

```r
pander(elevator, split.table = 120, style = 'rmarkdown')
```

ID	Type	Weight	Arr_Time	Wait_Time	Wait_for_Elev	Wait_for_Weight	System_Time
1	A	200	3	3	No	Yes	6
2	C	50	3	3	No	Yes	6
3	B	100	6	0	No	No	3
4	C	50	6	0	No	No	3
5	A	200	8	4	No	Yes	7
6	C	50	9	3	No	Yes	6
7	C	50	11	1	No	Yes	4
8	B	100	12	0	No	No	3
9	C	50	13	6	No	Yes	9
10	A	200	14	5	No	Yes	8
11	C	50	15	4	No	Yes	7
13	C	50	17	2	No	Yes	5
15	C	50	19	0	No	No	3
12	A	200	17	8	No	Yes	11
14	B	100	18	7	No	Yes	10
17	C	50	22	3	No	Yes	6
20	C	50	25	0	No	No	3
16	A	200	21	9	No	Yes	12
18	B	100	24	6	No	Yes	9
21	C	50	27	3	No	Yes	6
24	C	50	30	0	No	No	3
19	A	200	25	9	Yes	No	12
22	A	200	29	5	Yes	No	8
23	B	100	30	6	No	Yes	9
25	C	50	33	3	No	Yes	6
26	A	200	34	2	No	Yes	5
28	C	50	36	0	No	No	3
27	B	100	36	6	No	Yes	9
29	C	50	39	3	No	Yes	6
30	A	200	41	1	No	Yes	4
32	C	50	42	0	No	No	3
31	B	100	42	5	No	Yes	8
33	C	50	44	3	No	Yes	6
34	A	200	47	0	No	No	3
35	C	50	47	0	No	No	3
36	B	100	48	5	No	Yes	8
37	C	50	50	3	No	Yes	6
38	A	200	52	1	No	Yes	4
39	C	50	53	0	No	No	3
40	B	100	54	3	Yes	Yes	6
41	C	50	55	2	Yes	Yes	5
42	A	200	57	0	No	No	3
43	C	50	57	0	No	No	3
44	B	100	60	NA	NA	NA	NA
45	C	50	60	NA	NA	NA	NA

Let us analyze the data generated. The variables significance of the generated data frame is given below:

ID - Represents the unique identifier for each of the items received.

Type - Item type (A, B, C)

Weight - Weight of the item. Item type A will be 200Kg, B will be 100Kg and C will be 50Kg

Arr_Time - Arrival time (clock time) of the item. Note that the items are not loaded into the elevator as per their arrival times (reason explained later)

Wait_Time - Item wait time (minutes) to get loaded into elevator

Wait_for_Elev - Is the item waiting for elevator?

Wait_for_Weight - Is the item waiting for the future items to make the cumulative weight of the items placed in the elevator as 400Kg?

System_Time - Total time (minutes) spent by the items in the system

The data is not ordered by ID or arrival times. It is ordered based on how the items are loaded into the elevator. Here are some points about the generated data that are worth noting:

The first and second rows show that the items have arrived in the 3rd minute, while 3 and 4 have arrived in the 6th minute. The first 2 items are placed in the elevator, and they have to wait till the cumulative weight becomes 400Kg. Hence the first 2 rows have "Yes" under the Wait_for_Weight column. The wait time for the first 2 items is 3 minutes. For 3rd and 4th items, there is no wait time, since they were placed directly into the elevator the moment they arrived, since the cumulative weight of the first 4 items is 400Kg.

The item with ID-12 was not directly placed into the elevator, although it arrived at 17th minute, since the elevator has already filled with items ID-9, 10, 11, whose cumulative weight is 300. If item with ID-12 is placed into elevator, then the cumulative weight will become 500Kg, which is not allowed. Hence the items with ID-13, and 14 (although they arrived later than ID-12) are placed along with ID-9,10,11. The ID-12 item is placed in the elevator in the next elevator's trip.

The items with ID-19 and 22 have to wait for the elevator to arrive (see the column Wait_for_Elev set to Yes for these two IDs). Items with ID-40 and 41 have waited for both the future weights and elevator.

Since we stopped the simulation after getting 1 hour worth of data, the last 2 observations are having NA values (and this represents that they are placed in the elevator but the cumulative weight has not become 400Kg).

Here are the answers to the questions asked in the problem:

Q. What is the average transit time of box type A?

A. The average transit time of type A box is 6.9166667 minutes

Q. What is the average waiting time of box type B?

A. The average waiting time of type B box is 4.2222222 minutes

Q. How many boxes of material C have made the trip in 1 hour?

A. 22 (The last row should not be counted, since it will be transported in the next hour, outside the simulation window)