ECS 256 Homework 1.2

Bingxi Li, Qiwei Li, Jiaping Zhang

January 2016

1 Math Problem B

Recall the Bus Paradox in our book. The quantity of interest was Dt, the time until the next bus arrival, where t is the time we arrive at the bus stop. We wished to determine the long-run distribution of Dt. If we are dealing with continuous random variables, that long-run density is

f(w) = (1 - FL(w)) / EL

where L is the time between buses in general (not at the times you arrive). And remarkably, if we look at the time At since the last bus, we get the same distribution.

If L is a discrete random variable, the result is the same, except that instead of a density we have a probability mass function. In this problem, you will derive the above equation for this discrete case, using Markov chains, for the case of the time since the last bus.

It will be easier to think of light bulbs. As soon as one burns out, we replace it. Let Q be the length of time the current bulb has been burning, so Q = 0 at the time of a replacement. If Q = 8, say, the bulb has already lasted 8 units of time, but if it burns out at the next time step, Q immediately becomes 0 again.

1.1 Density of D_t in Discrete Case

For discrete case, the cumulative distribution function of waiting time D_t can be expressed as follows,

$$\begin{aligned} P(D_t \le \omega) &= \sum_{i=0}^{w} P(D_t = i) \\ &= \sum_{i=0}^{w} cP(L \ge i + t | L \ge t) \\ &= \sum_{i=0}^{w} cP(L \ge i) \end{aligned} \tag{1}$$

where the $\omega = Q\Delta t$ and Δt is smallest time unit. The normalization constant c can be determined as,

$$\begin{aligned} 1 = P(D_t \le \infty) &= \sum_{i=0}^{\infty} cP(L \ge i) \\ c &= \frac{1}{\sum_{i=0}^{\infty} P(L \ge i)} \end{aligned} \tag{2}$$

Using the "tail sum formula", c is therefore

$$\begin{aligned} c &= \frac{1}{\sum_{i=0}^{\infty} P(L \ge i)} \\ &= \frac{1}{EL} \end{aligned} \tag{3}$$

Combing equation 1 and 3, we have

$$\begin{aligned} P(D_t \le \omega) &= \sum_{i=0}^{w} \frac{P(L \ge i)}{EL} \\ &= \sum_{i=0}^{w} \frac{1 - F_L(w)}{EL} \end{aligned} \tag{4}$$

Therefore,

$$P(D_t = \omega) = \frac{1 - F_L(w)}{EL} \tag{5}$$

1.2 Density of A_t in Discrete Case

The derivation for A_t is similar here.

For discrete case, the cumulative distribution function of waiting time A_t can be expressed as follows,

$$\begin{aligned} P(A_t \le \omega) &= \sum_{i=0}^{w} P(A_t = i) \\ &= \sum_{i=0}^{w} cP(L \ge i) \end{aligned} \tag{6}$$

where the $\omega = Q\Delta t$ and Δt is smallest time unit. The normalization constant c can be determined as,

$$\begin{aligned} 1 = P(A_t \le \infty) &= \sum_{i=0}^{\infty} cP(L \ge i) \\ c &= \frac{1}{\sum_{i=0}^{\infty} P(L \ge i)} \end{aligned} \tag{7}$$

Using the "tail sum formula", c is therefore

$$\begin{aligned} c &= \frac{1}{\sum_{i=0}^{\infty} P(L \ge i)} \\ &= \frac{1}{EL} \end{aligned} \tag{8}$$

Combing equation 1 and 3, we have

$$\begin{aligned} P(A_t \le \omega) &= \sum_{i=0}^{w} \frac{P(L \ge i)}{EL} \\ &= \sum_{i=0}^{w} \frac{1 - F_L(w)}{EL} \end{aligned} \tag{9}$$

Therefore,

$$P(A_t = \omega) = \frac{1 - F_L(w)}{EL} \tag{10}$$

2 Math Problem C

Here you will model queuing in a call center, with the following characteristics:

- There are s servers who take calls, and a buffer size of b calls. If a call arrives when b calls are already in the system (being served or waiting on hold), that call is declined, with a message "Try again later." From a modeling point of view, these calls are simply lost.

- A proportion q of calls are complex, and must be referred to the manager. They are queued if the manager is busy. This is determined upon call arrival.
- Customers in the arrivals queue may get tired of waiting, and leave.
- Distributions are exponential, with the following rates:
  - Arrival rate α.
  - Call service rate σ for ordinary calls, μ for ones that need the manager's attention.
  - A customer's tolerance for waiting has mean $1/\omega$. If her wait in the arrivals queue exceeds this, she leaves.

2.1 Part A

Set this up as a continuous-time Markov chain, with states (i,j), where i is the number of ordinary calls currently in the system (being served or waiting) and j is the number of complex calls.

- Express the proportions of calls that (i) are declined and (ii) result in customers leaving due to impatience, in terms of π.
- Solve it using your CRAN project, for several sets of the parameter values, and evaluate the above two quantities. You'll have to relabel the subscripts, so that (0,0) becomes 1, (1,0) becomes and so on.

Notations

The mathematical notations for this question are listed as follows,

- Number of servers: s
- Buffer size: b
- Proportion of complex calls: q
- Arrival rate: α
- Ordinary calls service rate: σ
- Complex calls service rate: μ
- Customer waiting tolerance: $\frac{1}{\omega}$

The stationary distribution of states with different numbers of ordinary and complex calls is notated as $\pi = [\pi_{(0,0)}, \pi_{(1,0)}, \ldots, \pi_{(i,j)}, \ldots, \pi_{(b,0)}, \pi_{(0,b)}]$. Note that there are totally $\frac{(b+1)(b+2)}{2}$ pairs(states) in π because the sum of two integer numbers should be less or equal to b.

2.1.1 Proportions of Declined and Leaving Calls

Proportion of Declined Calls

The call will be declined when the total number of i and j is equal to b. So the proportion of declined calls, $P_{declined}$,

$$P_{declined} = \sum_{i=0}^{b} \pi_{(i,b-i)} \tag{11}$$

Proportion of Leaving Calls

The customers who will leave are those who are still in queues in the long run. The number of potentially leaving people is $i+j-s-1$, where the $s+1$ is the service capacity of s servers and one manager.

$$P_{leaving} = \sum_{i+j>(s+1)}^{b} \frac{i+j-s-1}{i+j} \pi_{(i,j)} \tag{12}$$

2.1.2 Derivation Results with Different Parameter Sets

To analytically derive the stationary distribution of states $\pi_{(i,j)}$, we need to have the infinitesimal generator Q. The Q can be determined transition rate equation. To have this equation, we firstly define an indicator function.

$$I(x) \;=\; \begin{cases} 1 & if \;\; x\; is\; satisfied \\ 0 & if \;\; x\; is\; unsatisfied \end{cases} \tag{13}$$

For the state $\pi_{(i,j)}$, its leaving rate is,

$$\begin{aligned} R_{leaving} =& \pi_{(i,j)} \cdot \{\alpha I(0 \le i+j \le b-1) \;+\; \sigma[sI(s \le i \le b) \;+\; iI(0 \le i \le s-1)] \\ & \;+\; \mu I(1 \le j \le b) \;+\; \omega[(j-1)I(2 \le j \le b) \;+\; (i-s)I(s+1 \le i \le b)]\} \end{aligned} \tag{14}$$

Its coming rates will be determined by four states $\pi_{(i-1,j)}, \pi_{(i,j-1)}, \pi_{(i+1,j)}, \pi_{(i,j+1)}$. For the four states, the sum of ordinary and complex calls should be no larger than b in order to appear on Markov Chain.

$$\begin{aligned} R_{coming} =& \pi_{(i+1,j)} \cdot [(i+1-s)wI(s \le i \le b-1) + s\sigma I(s \le i \le b-1) + (i+1)\sigma I(0 \le i \le s-1)] + \\ & \pi_{(i,j+1)} \cdot [j\omega I[1 \le j \le b-1] + \mu I(0 \le j \le b-1)] + \\ & \pi_{(i-1,j)} \cdot [\alpha(1-q)I(1 \le i \le b)] + \\ & \pi_{(i,j-1)} \cdot [\alpha q I(1 \le j \le b)] \end{aligned} \tag{15}$$

The equation 14 and 15 can be combined as,

$$R_{leaving} = R_{coming} \tag{16}$$

By reformulating equation 16, we have

$$R_{leaving} - R_{coming} = 0 \tag{17}$$

The infinitesimal generator Q can then be determined from equation 17. Using our derived equation and our simulation code from the next section, we get the following table of result.

Table 1: Testing Results for Different Parameter Sets

case	$s = 1$ $b = 1$ $q = 0.5$ $\alpha = 0.4$ $\sigma = 0.3$ $\mu = 0.2$ $\omega = 0.1$	$s = 2$ $b = 4$ $q = 0.1$ $\alpha = 0.4$ $\sigma = 0.3$ $\mu = 0.2$ $\omega = 0.25$	$s = 3$ $b = 5$ $q = 0.2$ $\alpha = 0.4$ $\sigma = 0.3$ $\mu = 0.2$ $\omega = 0.2$
Derivation	P(declined)=0.625 P(leaving) = 0	P(declined)=0.064 P(leaving) = 0.016	P(declined)=0.061 P(leaving) = 0.012
Simulation	P(declined)=0.623 P(leaving) = 0	P(declined)=0.143 P(leaving) = 0.190	P(declined)=0.124 P(leaving) = 0.119

We are well aware that our derivation result and simulation result do not match. After long hours of investigating, we conclude that the simulation is comprehensive and our derivation is missing cases. For example, in our derivation, when a complex call comes in, no matter if the server is available to pick it up, the call goes straight into the manager queue understand the constrain of buffer size. In our simulation, this situation is well taken care of. The key factor that prevents us from getting the derivation comprehensive is that the event of a complex call going to the manager queue through a server does not cost time. We could not figure out a way to include this in the rate. We would like to learn.

2.2 Part B

We will show our simulation result by using our simulation code in the code section and the following parameter. We did not use the DES package because we want to get things right. We do not want to start coding a complex simulation using a general framework. By using our code, each state of the call center is returned for debugging. We are very confident that the simulation is correct. With the below input,

- Number of servers: s = 2
- Buffer size: b = 4
- Proportion of complex calls: q = 0.1
- Arrival rate: $\alpha = 0.4$
- Ordinary calls service rate: $\sigma = 0.3$
- Complex calls service rate: $\mu = 0.2$
- Customer waiting tolerance: $\frac{1}{\omega} = \frac{1}{0.25}$

Let us see our simulation results:

Simulation runs:	500	1000	1500	2000	2500	3000	3500	4000
P(declined):	0.1420118	0.1260997	0.1053678	0.1322437	0.1413673	0.1369335	0.1377049	0.130186
P(left):	0.1656805	0.1935484	0.1968191	0.1946508	0.2039397	0.2054002	0.202459	0.202432

As we can see, the result converges.

3 Code

3.1 Derivation project Verification

```
generate_matrix<- function(s,b,q,alpha,sigma,mu,w)
{
   num = (b+1)*(b+2)/2
   Q <- matrix(0,nrow=num,ncol=num)
   #construct the possible pairs such that i+j<=b and record it as a state
    queue_ij <-list()
    idx = 0
    for(i in 0:b){
       for(j in 0:b){
          if(i+j<=b){
            idx = idx+1
            queue_ij[[idx]]<- idx
            names(queue_ij)[[idx]] <- paste0(j,',',i)
          }
       }
    }

    #There are 15 states so we have 15 equations
    for(state_str in names(queue_ij)){
        state_name = strsplit(state_str,',')
        i = as.numeric(state_name[[1]][1])
        j = as.numeric(state_name[[1]][2])

        #state_i will range from 1 to 15 sequentially in the list of states
        state_i = queue_ij[state_str][[1]][1]
        r_leaving = alpha*ifelse(i+j>=0 && i+j<= b-1,1,0) + sigma*(s*ifelse(i>=s && i<= b,1,0) +
        i*ifelse(i>=0 && i<=s-1,1,0)) +
        mu*ifelse(j>=1 && j<=b,1,0) +
        w*((j-1)*ifelse(j>=2 && j<=b,1,0) +
        (i-s)*ifelse(s+1<=i && i<=b,1,0))
```

```
      Q[state_i, state_i] = r_leaving

      #We will define the values on [state_i, state_j_?]
      state_j_str_1 = paste0(i+1,',',j) #from pi(i+1,j)
      if(state_j_str_1 %in% names(queue_ij)){
        state_j_1 = queue_ij[state_j_str_1][[1]][1] #retrieve the actual state from mapping
        #calculate the value to put on [state_i, state_j_1], possibly 0
        value_1 = -((i+1-s)*w*ifelse(s<=i && i<=b-1,1,0)+
        s*sigma*ifelse(s<=i && i<=b-1,1,0)+
        (i+1)*sigma*ifelse(0<=i && i<=s-1,1,0))
        #print(value_1)
        Q[state_i,state_j_1] = value_1
      }

      state_j_str_2 = paste0(i,',',j+1) #from pi(i,j+1)
      if(state_j_str_2 %in% names(queue_ij)){
        state_j_2 = queue_ij[state_j_str_2][[1]][1]
        value_2 = -(j*w*ifelse(i<=j && j<=b-1,1,0)+mu*ifelse(0<=j && j<=b-1,1,0))
        #print(value_2)
        Q[state_i,state_j_2] = value_2
      }

      state_j_str_3 = paste0(i-1,',',j) #from pi(i-1,j)
      if(state_j_str_3 %in% names(queue_ij)){
        state_j_3 = queue_ij[state_j_str_3][[1]][1]
        value_3 = -alpha*(1-q)*ifelse(1<=i && i<=b,1,0)
        #print(value_3)
        Q[state_i,state_j_3] = value_3
      }

      state_j_str_4 = paste0(i,',',j-1) #from pi(i,j-1)
      if(state_j_str_4 %in% names(queue_ij)){
        state_j_4 = queue_ij[state_j_str_4][[1]][1]
        value_4 = -alpha*q*ifelse(1<=i && i<=b,1,0)
        #print(value_4)
        Q[state_i,state_j_4] = value_4
      }
    }
  return(list(RateMatrix=Q, StateMap=queue_ij))
}

#Case 1
s = 2
b = 4
q = 0.1
alpha = 0.4
sigma = 0.3
mu = 0.2
w = 0.25
result = generate_matrix(s,b,q,alpha,sigma,mu,w)
Q = result$RateMatrix
state_map = result$StateMap
mc.test1 = mc.create(Q,discrete = F, infinite = F, use_ratematrix=T)
pi = getStationaryDistribution(mc.test1)
p_declinded = 0
for(i in 0:b){
  j = b-i
  state_str = paste0(i,',',j)
  state = state_map[state_str][[1]][1]
  p_declinded = p_declinded + pi[state]
}
p_declinded
# 0.0642546

#Proportion of Leaving Calls
p_leaving = 0
for(i in 0:b){
  for(j in 0:b){
    state_str = paste0(i,',',j)
    if(i+j>(s+1) && state_str%in% names(state_map)){
      #print(state_str)
      state = state_map[state_str][[1]][1]
      p_leaving = p_leaving + (i+j-s-1)/(i+j)*pi[state]
    }
  }
}
p_leaving
# 0.01606365
```

3.2 Simulation

```
callCenter = function(eventLimit,number_of_server,buffer_size,complex_prob,arriveRate,
    ordServiceRate,compServiceRate,tiredRate){
  wantToKeep =  c("currentTime", "currentEvent",
                "numOrdCall", "numCompCall",
                "serverUsed", "managerUsed",
                "holdLine", "compLine","totalInSystem",
                "servedOrdCall", "servedCompCall",
                "totalCall", "totalDecline", "totalLeft")

  arriveList = cumsum(rexp(eventLimit, rate= arriveRate))
  tiredList = cumsum(rexp(eventLimit, rate = tiredRate))
  ordServiceList = cumsum(rexp(eventLimit, rate = ordServiceRate))
  compServiceList = cumsum(rexp(eventLimit, rate = compServiceRate))
  allEventList = rbind(cbind(arriveList, rep('arrive', length(arriveList))),
                      cbind(tiredList, rep('tired', length(tiredList))),
                      cbind(ordServiceList, rep('ordFinish', length(ordServiceList))),
                      cbind(compServiceList, rep('compFinish', length(compServiceList))))
  allEventList = allEventList[order(as.numeric(allEventList[,1])), ]
  holdLine = character()
  index=1

  info = as.data.frame(matrix(NA, nrow=100, ncol=length(wantToKeep)))
  colnames(info) = wantToKeep
  info[1, ] = rep(0, ncol(info))
  for(i in 1:nrow(allEventList)){
    info[index+1, ] = info[index, ]
    index = index + 1
    thisTime = allEventList[i, 1]
    thisEvent = allEventList[i, 2]
    info[index, 'currentTime'] = round(as.numeric(thisTime),4)
    info[index, 'currentEvent'] = thisEvent

    if(thisEvent == 'arrive'){
      if(info[index, 'totalInSystem'] >= buffer_size){
        info[index, 'totalDecline'] = info[index, 'totalDecline'] + 1
        next
      }
      info[index, 'totalInSystem'] = info[index, 'totalInSystem'] + 1

      callType = sample(c('ord','comp'), 1, prob = c(1-complex_prob, complex_prob))
      if(callType == 'ord')
        info[index, 'numOrdCall'] = info[index, 'numOrdCall'] + 1
      if(callType == 'comp')
        info[index, 'numCompCall'] = info[index, 'numCompCall'] + 1

      if(info[index, 'serverUsed'] >= number_of_server){
        info[index, 'holdLine'] = info[index, 'holdLine'] + 1
        holdLine = append(holdLine, callType)
        next
      }
      if(info[index, 'serverUsed'] < number_of_server){
        if(callType == 'ord')
          info[index, 'serverUsed'] = info[index, 'serverUsed'] + 1
        if(callType == 'comp'){
          if(info[index, 'managerUsed'] == 1)
            info[index, 'compLine'] = info[index, 'compLine'] +1
          if(info[index, 'managerUsed'] == 0)
            info[index, 'managerUsed'] = 1
        }
        next
      }
      print("arrive LIMBO")
    }

    if(thisEvent == 'tired'){
      if(info[index, 'holdLine']==0)
        next
      if(info[index, 'holdLine']>0){
        if(length(holdLine) <= 0)
          stop("holdLine number and object do not match")
        personIndex = sample(length(holdLine), 1)
        person = holdLine[personIndex]
        holdLine = holdLine[-personIndex]
        info[index, 'holdLine'] = info[index, 'holdLine'] - 1
        info[index, 'totalInSystem'] = info[index, 'totalInSystem'] - 1
        info[index, 'totalLeft'] = info[index, 'totalLeft'] + 1
        if(person == 'ord')
          info[index, 'numOrdCall'] = info[index, 'numOrdCall'] -1
        if(person == 'comp')
          info[index, 'numCompCall'] = info[index, 'numCompCall'] -1
        next
      }
      print("tired LIMBO")
    }

    if(thisEvent == 'ordFinish'){
      if(info[index, "serverUsed"]==0)
        next

      info[index, "numOrdCall"] = info[index, "numOrdCall"] -1
      info[index, "serverUsed"] = info[index, "serverUsed"] -1
      info[index, "totalInSystem"] =  info[index, "totalInSystem"] -1
      info[index, "servedOrdCall"] = info[index, "servedOrdCall"] +1
      info[index, "totalCall"] = info[index, "totalCall"] +1

      while(info[index, 'serverUsed'] < number_of_server && info[index, "holdLine"] > 0){
        info[index, "holdLine"] = info[index, "holdLine"] -1
        person = holdLine[1]
        holdLine = holdLine[-1]

        if(person == 'ord')
          info[index, "serverUsed"] = info[index, "serverUsed"] +1
        if(person == 'comp'){
          if(info[index, 'managerUsed']==1){
            info[index, "compLine"] = info[index, "compLine"] +1
          } else {
            info[index, "managerUsed"] = 1
          }
        }
      }
    }

    if(thisEvent == 'compFinish'){
      if(info[index, 'managerUsed'] == 0)
        next
      info[index, "numCompCall"] = info[index, "numCompCall"] -1
      info[index, "totalInSystem"] =  info[index, "totalInSystem"] -1
      info[index, 'servedCompCall'] = info[index, 'servedCompCall']+1
      info[index, "totalCall"] = info[index, "totalCall"] +1
      info[index, 'managerUsed'] = 0
      if(info[index, 'compLine'] == 0)
        next
      if(info[index, 'compLine'] > 0){
        info[index, 'compLine'] = info[index, 'compLine'] -1
        info[index, 'managerUsed'] <- 1
        next
      }
      print("compFinish LIMBO")
    }
  }
  info = info[1:floor(eventLimit/2), ]
  total_called = sum(info$currentEvent == 'arrive')
  total_declined = max(info$totalDecline)
  total_leave = max(info$totalLeft)
  cat("p_declined: ")
  print(total_declined/total_called)
  cat("p_leave: ")
  print(total_leave/total_called)
  return(info)
}
```