Stochastic Processes: Assignment 1

Group 1: Javier Esteban Aragoneses, Mauricio Marcos Fajgenbaun, Danyu Zhang, Daniel Alonso

November 27th, 2020

Importing libraries

Problem 1

```python
import numpy as np
from copy import deepcopy
import pandas as pd
from nltk.corpus import words
import matplotlib.pyplot as plt
import threading as th

freq =  np.loadtxt('./data/Englishcharacters.txt', usecols=range(27))

def transf_minmax(table,a,b):
    return (b-a)*((table - np.min(table))/(np.max(table) - np.min(table))) + a

def transf_log(table):
    return np.log(table + 1)

freq = transf_log(freq)

with open('./data/messages.txt') as f:
    message = f.readlines()[9].replace('\n','')

def decode(message, freqs, iters):
    # messages per iteration
    msg_iters = {}
    # identity function
    letters = ["a", "b", "c", "d",
               "e", "f", "g", "h",
               "i", "j", "k", "l",
               "m", "n", "o", "p",
               "q", "r", "s", "t",
               "u", "v", "w", "x",
               "y", "z"]
    init_letters = deepcopy(letters)
    cd = {l:l for l,l in zip(init_letters,letters)}
    cd.update({' ':' '})
    def f(c):
        return cd[c]
```

```python
# mapping letters to cols
fvals = np.array([x for x in range(len(letters))])
cd_map = {l:v for l,v in zip(letters,fvals)}
cd_map.update({' ':26})
def f_map(c):
    return cd_map[c]

# score function
def score(fun):
    p = 0
    for i in range(1,len(msg)):
        p = p + freq[f_map(fun(msg[i-1])),f_map(fun(msg[i]))]
    return p

# msg to list
msg = list(message)

# letters to modify the cipher
letters_n = deepcopy(letters)

for i in range(iters):

    # randomly choose 2 numbers and replace the 2 chosen
    # vals in a copy of letters
    ch1 = np.random.randint(1,len(letters))
    ch2 = np.random.randint(1,len(letters))
    plc1 = deepcopy(letters_n[ch1])
    plc2 = deepcopy(letters_n[ch2])
    letters_n[ch1] = plc2
    letters_n[ch2] = plc1

    cd_n = {l:v for l,v in zip(init_letters,letters_n)}
    cd_n.update({' ':' '})

    # f asterisk
    def f_n(c):
        return cd_n[c]

    #loop with scores
    scr_f = score(f)
    scr_fn = score(f_n)
    scr = scr_fn/scr_f

    # print(f'score f*: {scr_fn}, score f: {scr_f}, ratio: {scr}')
    cond = np.random.rand() <= scr

    if cond:
        letters = deepcopy(letters_n)
        cd = {l:v for l,v in zip(init_letters,letters)}
        cd.update({' ':' '})
        for k in range(len(msg)):
            msg[k] = f(msg[k])
        msg_iters[i] = (scr,''.join([x for x in msg]))
```

```
        else:
            for k in range(len(msg)):
                msg[k] = f(msg[k])
        msg = list(message)

    # put in a dataframe
    df = {'iter':[it for it in msg_iters.keys()],
          'score':[msg[0] for msg in msg_iters.values()],
          'msg':[msg[1] for msg in msg_iters.values()]}
    return pd.DataFrame(df)

result = decode(message,freq, 50000)
plt.plot(result.sort_values('score')['score'].reset_index(drop=True))
print(result[result['score'] == max(result['score'])])
```

```
#> [1] "n bnq ndrsw awornoc aslnxs rzs dowfr ksfrswc ljicrwq rj nirzjwoms n ljwjcngowif gnllocs aworofz
```

Problem 2

(a)

Let $N(t)$ be the number of cars arriving at a parking lot by time t, according to the proposed scenario, we can model $N(t)$ as a non-homogenous Poisson process. Such process has almost the same process as any other Poisson process, however, its rate is a function of time.

$N(t), t \in [0, \infty)$ is the non-homogenous Poisson process with rate $\lambda(t)$ where:

- $N(0) = 0$
- $N(t)$ has independent increments

We define 8:00 as $t = 0$ with the following integrable function and each unit of t equals to 1 hour:

$$\lambda(t) = \begin{cases} 100 & 0 \le t \le \frac{1}{2} \\ 600t - 200 & \frac{1}{2} < t \le \frac{3}{4} \\ 400t - 50 & \frac{3}{4} < t \le 1 \\ -500t + 850 & 1 < t \le 1.5 \end{cases}$$
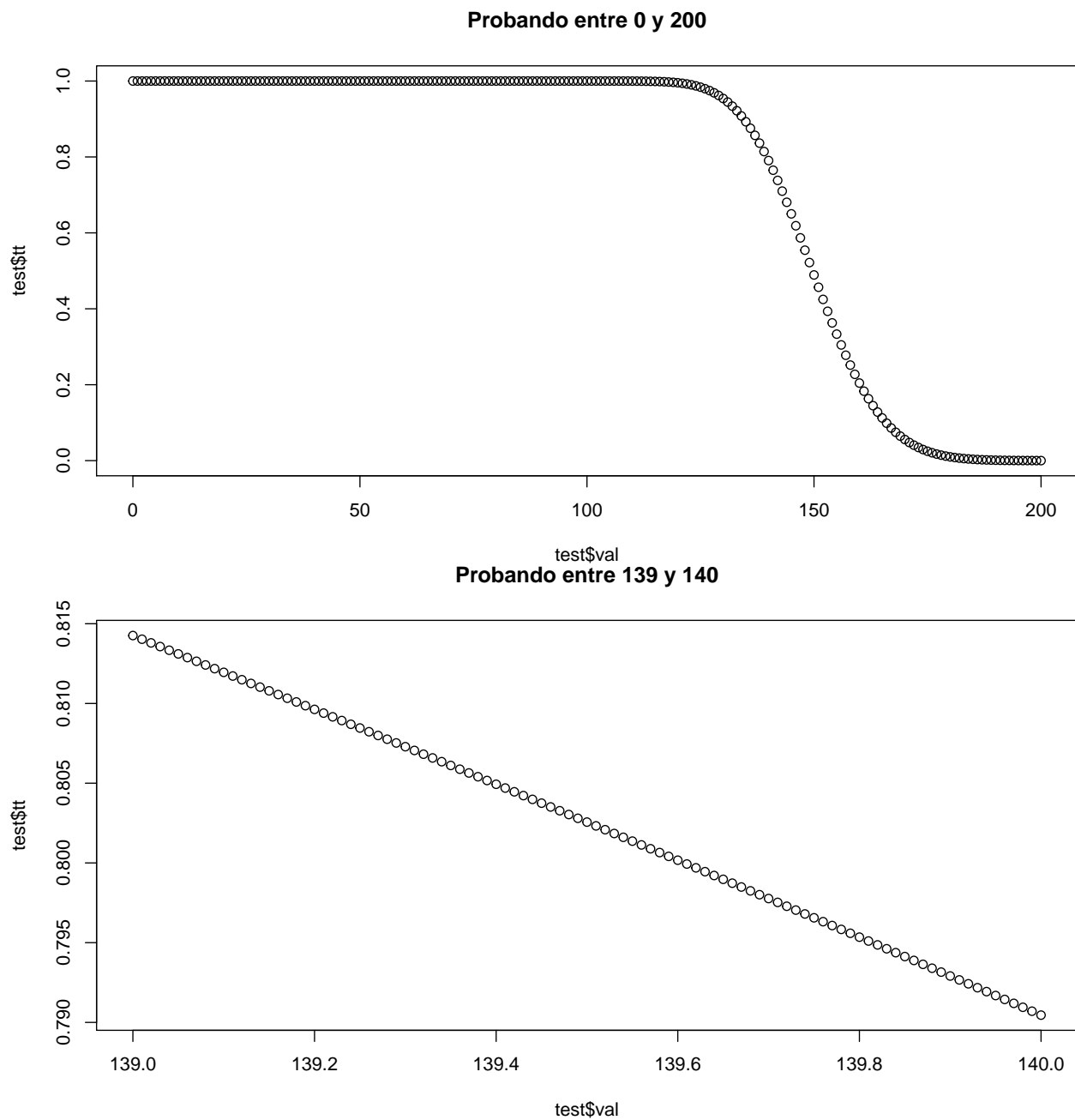
So,

$$E[N(t)] = \begin{cases} \int_0^t 100 \, dt = 100t & 0 \le t \le \frac{1}{2} \\ \int_{\frac{1}{2}}^t 600t - 200 \, dt + 50 = 300(t^2 - \frac{1}{4}) - 200(t - \frac{1}{2}) + 50 & \frac{1}{2} < t \le \frac{3}{4} \\ \int_{\frac{3}{4}}^t 400t - 50 \, dt + 93.75 = 25(8t^2 - 2t - 3) + 93.75 & \frac{3}{4} < t \le 1 \\ \int_1^t -500t + 850 \, dt + 168.75 = -50(5t^2 - 17t + 12) + 168.75 & 1 < t \le 1.5 \end{cases}$$

Given that there is a limit of 150 vehicles:

$$E[N(t)] = \begin{cases} 100t & 0 \le t \le \frac{1}{2} \\ 300(t^2 - \frac{1}{4}) - 200(t - \frac{1}{2}) + 50 & \frac{1}{2} < t \le \frac{3}{4} \\ 25(8t^2 - 2t - 3) + 93.75 & \frac{3}{4} < t < 0.94468 \\ 150 & t \ge 0.94468 \end{cases}$$

3

(b)

Probando entre 0 y 200

Probando entre 139 y 140

Luego de hacer las pruebas para $\lambda(t)$ obtenemos lo siguiente:

```
lambda = 139.6
t = 0.91232
# 8:44 AM
```

Por lo que $t = 0.91232$ horas (aproximadamente a las 8:54 de la mañana).

(c)

The following function simulates a non-homogenous poisson process from a homogenous poisson process:

```r
non_hom_poisson <- function(fun,l,a,b,start=0) {
    # This function generates a non-homogenous poisson
    # process from a homogenous poisson process
    # PARAMS:
    # fun:   if the non-homogenous poisson process has
    #        multiple functions per time subinterval
    #        this parameters represents such function
    # l:     lambda for the homogenous poisson process
    # a:     lower bound for the time subinterval
    # b:     upper bound for the time subinterval
    # start: this parameter is used to keep track of
    #        the process count.

    # We generate the homogenous poisson process
    # arrival times
    val <- rpois(1,l*(b-a))
    intervals <- (b-a) * sort(runif(val)) + a

    # Non-homogenous poisson process
    evs <- length(intervals) # lenght of arrival times
    nh_val <- 0 + start # start of the event count
    nh_intervals <- c() # arrival times for the NHPP
    for (i in 1:evs) {
        if (runif(1) < fun(intervals[i])/l) {
            # only including intervals from the HPP which
            # match with fun(intervals[i])/l probability
            nh_intervals <- c(nh_intervals, intervals[i])
            nh_val <- nh_val+1 # adding one to the event count
        }
    }
    nh_events <- seq(1+start,nh_val,1) # events since the previous group
    return(list(arrival_times=nh_intervals, events=nh_events))
}
```

```r
simulation <- function(iters, functions, lambdas, ints) {
    # This function simulates from the NHPP
    # iters:      number of iterations to plot and add to the list of
    #             dataframes
    # functions: list of functions corresponding to the lambda function
    # lambdas:    list of lambdas for each subinterval
    # ints:       lists of vectors of 2 elements each containing the intervals
    #             that correspond to each element of lambdas and functions lists
    p <- list()
    for (i in 1:iters) {
        maximum <- 0 # start for the next NHPP simulation to continue count
        arr_times <- c() # arrival times
        events <- c() # event counts
        for (k in 1:4) {
            int <- non_hom_poisson(lambda_funs[[k]],lambdas[[k]],
                                   ints[[k]][1],ints[[k]][2],
                                   start=maximum)
            maximum <- max(int$events) # remembering last event count
            arr_times <- c(arr_times, int$arrival_times)
            events <- c(events, int$events)
        }
        p[[i]] <- data.frame(arrival_times=arr_times, events=events)
        # plots
        if (i == 1) {plot(arr_times, events, cex=0.5, pch='.',
                          col=randomColor(), xlim=c(0,1.5),
                          ylim=c(0,150))}
        lines(arr_times, events, col=randomColor())
    }
    return(p)
}
```

```r
data <- simulation(1000, lambda_funs, lambdas, ints)
```
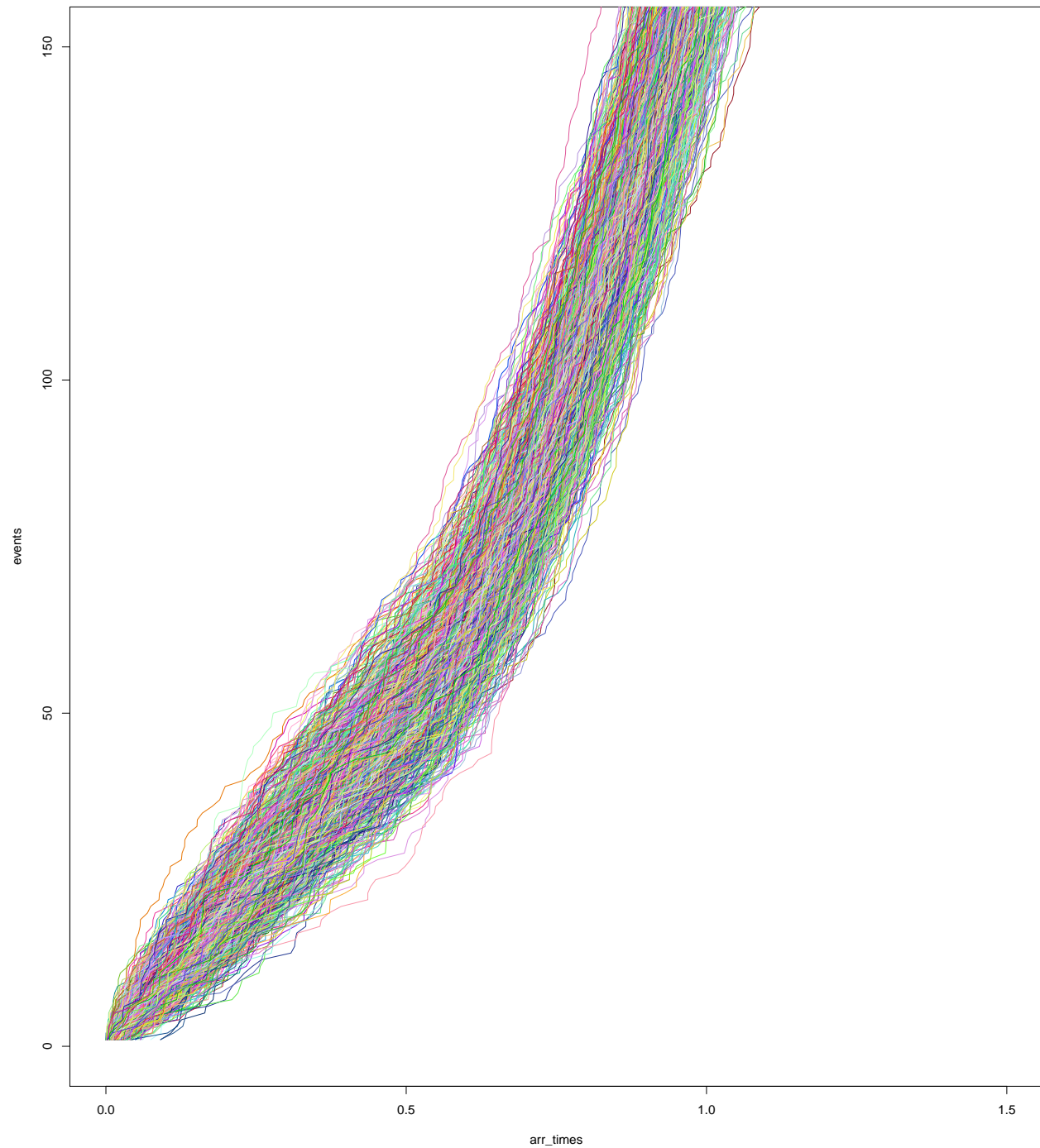


```r
ratio <- 0
for (i in 1:length(data)) {
    df <- data.frame(data[[i]])
    cnt <- df %>% filter(arrival_times < 0.91232 & events >= 150) %>% dplyr::count()
    if (cnt[1] >= 1) {
        ratio <- ratio + 1
    }
}
```

```
ratio/1000
#> [1] 0.204
```

Problem 3

(a)

Our infinitesimal generator is the following:

$$Q = \begin{pmatrix} -\lambda & \lambda & 0 & 0 & \dots \\ \mu & -(\lambda + \mu) & \lambda & 0 & \dots \\ 0 & 2\mu & -(2\mu + \lambda) & \lambda & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

(b)

We solve the following system:

$$\begin{cases} \sum_{i=1}^{\infty} \pi_i = 1 \\ \lambda \pi_0 = \mu \pi_1 \\ \lambda \pi_1 = 2\mu \pi_2 \\ \vdots \\ \lambda \pi_{n-1} = 2\mu \pi_n \\ \vdots \end{cases}$$

First we have:

$\pi_1 = \frac{\lambda \pi_0}{\mu}$

$\pi_2 = \frac{\lambda^2 \pi_0}{2\mu^2}$

$\pi_3 = \frac{\lambda^3 \pi_0}{2^2 \mu^3}$

$\vdots$

$\pi_n = \frac{\lambda^n \pi_0}{2^{n-1} \mu^n}$

$\vdots$

Then:

$\sum_{i=0}^{\infty} \pi_i = \pi_0 + \frac{\lambda \pi_0}{\mu} + \frac{\lambda^2 \pi_0}{2\mu^2} + \dots + \frac{\lambda^n \pi_0}{2^{n-1} \mu^n} + \dots = 1$

And so factoring π_0 we get:

$\pi_0(1 + \frac{\lambda}{\mu} + \frac{\lambda^2}{2\mu^2} + \dots) = \pi_0(1 + \sum_{i=1}^{\infty} \frac{1}{2^{i-1}} (\frac{\lambda}{\mu})^i)$

Then multiplying $\frac{2}{2}$ to the summation:

$\pi_0(1 + \frac{2}{2} \sum_{i=1}^{\infty} \frac{1}{2^{i-1}} (\frac{\lambda}{\mu})^i)$

$= \pi_0(2 \sum_{i=0}^{\infty} (\frac{\lambda}{2\mu})^i - 1)$

$\pi_0(2(\frac{1}{1 - \frac{\lambda}{2\mu}}) - 1) = 1$

$\pi_0 = \frac{1}{2(\frac{1}{1 - \frac{\lambda}{2\mu}}) - 1}$

$$\vdots$$

$$\pi_n = \frac{\lambda^n}{2^{n-1}\mu_n\pi_0} \frac{1}{2(\frac{1}{1-\frac{\lambda}{2\mu}})-1}$$

finally:

$$\pi_n = \frac{1}{2^{n-1}}(\frac{\lambda}{\mu})^n\pi_0$$

The infinite sum converges when $|\frac{\lambda}{2\mu}| < 1$ in which case the stationary distribution P exists.

Then:

$$L = \sum_{n=0}^{\infty} \pi_n * n$$

Using the following sum:

$$\sum_{i=0}^{n-1} ia^i = \frac{a-na^n+(n-1)a^{n+1}}{(1-a)^2}$$

As n approaches infinity:

$$\sum_{i=0}^{\infty} ia^i = \frac{a}{(1-a)^2}$$

We get the following:

$$L = \pi_0[1 + \sum_{n=0}^{infty}(\frac{\lambda}{\mu})^n * \frac{n}{2^{n-1}}]$$

$$L = \pi_0[1 + 2\sum_{n=0}^{infty}(\frac{\lambda}{2\mu})^n * n]$$

$$L = \pi_0[1 + 2\frac{\frac{\lambda}{2\mu}}{(1-\frac{\lambda}{2\mu})^2}]$$

(c)

Let's consider the probabilities conditioned on the number of customers in the system that are present once our specific subject l gets into the system.

If there are no other customers when l gets into the system, there is no chance of overtaking.

$$P(N^{OV} = 0|N^{PR} = 0) = 1$$

With N^{OV} being the number of customers that l overtakes and N^{PR} the number of customers present in the system (queing) when l gets in the system.

If $N^{PR} = 1$, then l can overtake only 1 customer, if the time it takes to be served is shorter than the time it takes the other customers to be served. Because of the memoryless property we can assert the following:

$$P(N^{OV} = 0|N^{PR} = 1) = \frac{\mu}{\mu+\mu} = \frac{1}{2}$$

Actually, in general:

$$P(N^{OV} = k|N^{PR} = n) = \frac{1}{n+1}, \; n \le c - 1, \; x = 0, 1$$

As in this case c=2, our l subject can't overtake more than one customer.

Now, if $n \ge c$, that is, l has to get in queue and wait to be served. When l gets served, there is also one more customer getting served. Because, again, of the memoryless property.

$$P(N^{OV} = k|N^{PR} = n) = \frac{1}{c}, \; n = c, \; k = 0, 1$$

In our case, it does not matter how many customers are in the system, the probability of overtaking, conditioned to the number of customers already in the system, is $\frac{1}{2}$.

Now, using Bayes' theorem and the total probability rule, we can find the probability of l overtaking another customer.

$P(A|B) = \frac{P(A \cap B)}{P(B)}$

$\frac{1}{2} \sum_{i=1}^{\infty} \pi_i = \frac{1}{2} \sum_{i=1}^{\infty} (\frac{1}{2^{i-1}})(\frac{\lambda}{\mu})^i \pi_0$

$= \sum_{i=1}^{\infty} (\frac{\lambda}{2\mu})^i \pi_0 = (\sum_{i=0}^{\infty} ((\frac{\lambda}{2\mu})^i) - 1)\pi_0$

$= (\frac{1}{1-\frac{\lambda}{2\mu}} - 1)\pi_0 = \frac{1}{2(\frac{1}{1-\frac{\lambda}{2\mu}}-1)}(\frac{1}{1-\frac{\lambda}{\mu}} - 1)$

$= \frac{1}{1-\frac{\lambda}{2\mu}} - 1 = \frac{1-(1-\frac{\lambda}{2\mu})}{1-\frac{\lambda}{2\mu}} = \frac{\frac{\lambda}{2\mu}}{1-\frac{\lambda}{2\mu}}$

$= 2(\frac{1}{1-\frac{\lambda}{2\mu}}) - 1 = \frac{2}{1-\frac{\lambda}{2\mu}} - 1 = \frac{2-(1-\frac{\lambda}{2\mu})}{1-\frac{\lambda}{2\mu}}$

$= \frac{1+\frac{\lambda}{2\mu}}{1-\frac{\lambda}{2\mu}}$

Then:

$\frac{\frac{\lambda}{2\mu}}{1\frac{\lambda}{2\mu}} = \frac{\frac{\lambda}{2\mu}}{\frac{2\mu+\lambda}{2\mu}} = \frac{\lambda}{2\mu+\lambda}$

So then we get:

$P(N^{OV} = k) = \frac{\lambda}{2\mu+\lambda}, \ k = c - 1 = 1$

(d)

We define the following function to simulate the queueing system:

```r
q <- function(customers, l, m) {
    exp_at <- rexp(n,l)
    ArrivalTimes <- rep(0,n)
    ArrivalTimes[1] <- exp_at[1]
    for (i in 2:customers) {
        ArrivalTimes[i] <- ArrivalTimes[i-1] + exp_at[i]
    }
    ServiceTimes <- rexp(customers, m)
    ExitTimes <- rep(0,customers)
    ExitTimes[1:2] <- ServiceTimes[1:2]
    for (i in 3:customers) {
        SortedTimes <- sort(ExitTimes[1:(i-1)], decreasing=T)
        if (ArrivalTimes[i] < SortedTimes[2]) {ExitTimes[i] <- SortedTimes[2] + ServiceTimes[i]}
         else {ExitTimes[i] <- ArrivalTimes[i] + ServiceTimes[i]}
    }
    times <- data.frame(arrival=ArrivalTimes,
                        exit=ExitTimes,
                        service=ServiceTimes)
    return(times)
}
```

(e)

We define the following function to calculate the overtaking probability:

```r
d <- function(customers, ot_prob, queue, ExitTimes) {
    for (i in queue:(customers - queue)) {
        ot_prob <- ot_prob + sum(ExitTimes[i] < ExitTimes[1:(i-1)])
```

```
    }
    r <- customers-2*queue
    ot_prob <- ot_prob/r
    return(ot_prob)
}
```