Universidad Carlos III de Madrid

MSc in Statistics for Data Science

Second Assignment

Stochastic Processes

Authors:

David de la Fuente López

Diego Perán Vacas

Katherine Fazioli

Adrian White

11th ***January 2021***

Exercise 1

The objective of this exercise is to use the Metropolis-Hastings algorithm, implemented in R, to solve a text decryption problem. This is a very similar approach to the example proposed in Persi Diaconis' seminal paper "The Markov Chain Monte Carlo Revolution," which advocates for the use of MCMC in a variety of applications. Indeed, the strategic use of a simple random walk through the function space of a substitution cipher yields promising results for basic encryptions based on character pair frequency matrices derivable from sufficiently large texts. In his first example, Diaconis uses Tolstoy's "War and Peace" as a reference text to derive the first order transition frequency matrix, then uses this to decrypt a passage from Shakespeare's "Hamlet" using the MCMC approach. Diaconis even extends this application to more complex linguistics, where the algorithm was successful in decoding a mixed language text.

This promising research was further elaborated in Dubrow's textbook "Introduction to Stochastic Processes with R," where the work of Jane Austen was used as a reference text to decrypt an exerpt of Edgar Allen Poe. We have followed the example provided in Dubrow's textbook as well as the R scripts provided.

This method assumes the use of a simple substitution cipher for the decryption of a coded text. The Metropolis-Hastings algorithm builds a simple random walk over the function space of the cipher, where at each step a permutation of the function is proposed where two letters, chosen randomly, swap their assigned values. The acceptance function is computed and compared to a random sample from uniform distribution on [0,1] to assess whether or not we accept the proposed function, then the selected decryption function is run.

The key to our optimization problem is the score function, which we construct using our first order transition frequency matrix, M. The score is calculated as the product over all successive pairs of letters in the decoded text, $(f(c_i), f(c_{i+1}))$ of the number of occurrences of that pair in the reference text.

Thus, we have

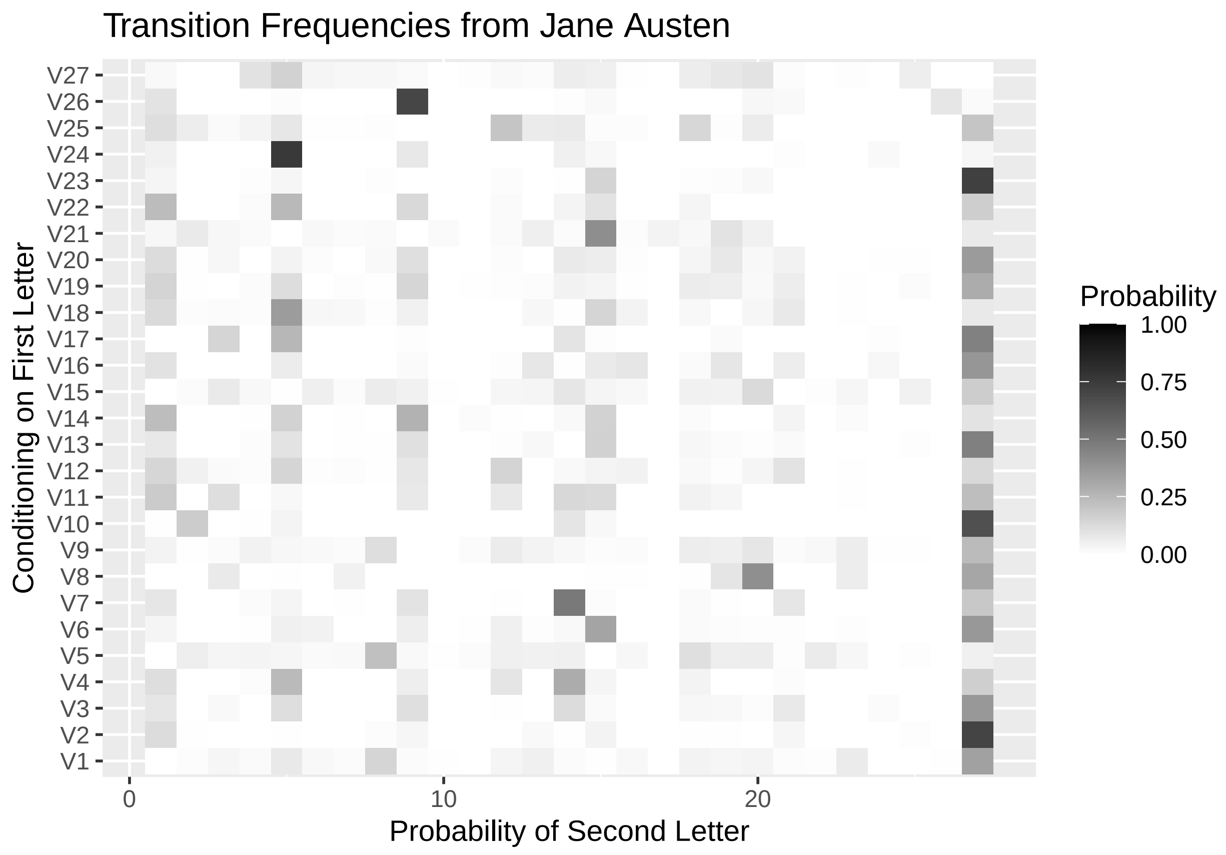
$$score(f) = \prod_{i=1}^{n} M_{f(c_i),f(c_{i+1})}$$

This score is computed at each step of the MCMC process, where the resulting decryption of the selected function is scored based on the similarity of its first order transitions to the reference text. As such, functions with high scores are good candidates for decryption.

We may summarize the steps of the Metropolis Hastings algorithm for clarity,

- Start with a preliminary guess, say f.
- Compute Pl(f).
- Change to f∗by making a random transposition of the values f assigns to two symbols.
- Compute Pl(f∗); if this is larger than Pl(f), accept f∗.
- If not, flip a Pl(f∗)/Pl(f) coin; if it comes up heads, accept f∗.
- If the coin toss comes up tails, stay at f.

We begin the implementation in R, following structure laid out by Dubrow in his example. In this exercise, the first order transition frequency matrix, M, was provided. As in Dubrow, it is based on Jane Austen's works. We load the frequency matrix and provide a preliminary visual analysis. The code is provided in the Appendix.



In this plot, we have a visual of the normalized transition frequencies for each letter pair. This gives us a sense of the structure of our first o rder t ransitions. C learly, t he d istribution o f f requencies s kews t o the left, where the majority of values are relatively low compared to the row averages. This makes sense, given that certain letters tend to pair much more frequently than others, otherwise this method would provide no insight, since the transitions would all be considered equal.

We can also look at the sub-distributions for each row or column to understand common letter pairs. For example, we see in tile (x = 14, y = 7) has a high probability, this refers to the letter pair transition g to n. Intuitively, we know that "gn" is a very common letter pair, seen in words like "cognition" or "ignition". By looking at column x = 1, which refers to the letter "a", or even column x = 5, which refers to letter "e", see that the distributions across the first letters are much more even with greater dispersion about the mean. This too, makes sense, since vowels tend to be much more subliminal in their placement. We see a similar dispersion in column 15 for letter "o". We may also note the scarcity of information available for certain letters, such as "j", "k", "q", "x", "y", and "z", which are generally less common.

While row normalization is useful for our visualization, we will use a log transformation of the transition matrix in order to facilitate the computation of the score function, since it requires taking the product over each letter pair, which can be expressed as a sum of logs. We will also use the score function and additional supporting functions, charIndex and ascii, based on Dubrow's work. These functions are used to reference the character values. We also have the decryption function, which takes the encrypted text and applies a decoding function, f.

These functions form the basic components of our algorithm, providing the application for the decryption function, f, and the score computation. We may now load the 6th encrypted message from the messages file. In addition, we prepare the score map and the initial decoding function, f. The code is provided in the Appendix.

Now, we run the Metropolis-Hastings algorithm over a large number of iterations and monitor the development of the results to check for promising attempts. The script includes the one step permutations of the decoding function, f, and the acceptance function, where the difference is scores is compared to a random sample from a uniform distribution over [0,1]. Since we are working with an MCMC approach, our optimization is not linear and promising results will appear throughout the process and may be reversed, possibly not to be recovered. As such, results are printed every 2000 iterations and are monitored closely. The code is provided in the appendix.

```
## [1] "2000"
## [2] "it thus required visionary researchers mike gemfand and spith to convince the coppunity sullort
## [3] "2071.23071945766"
## [1] "4000"
## [2] "it thus required kisionary researchers wive gewfand and spith to conkince the coppunity sullort
## [3] "2198.67261031563"
## [1] "6000"
## [2] "it thus required kisionary researchers wive gewfand and smith to conkince the community sullort
## [3] "2041.74999143302"
## [1] "8000"
## [2] "it thus required kisionary researchers wive gewfand and smith to conkince the community sullort
## [3] "2197.55236515269"
## [1] "10000"
## [2] "it thus required gisionary researchers wive pewfand and smith to congince the community sullort
## [3] "2136.12469987458"
```

We run the algorithm and track the progression of the results. This progress requires a close monitoring by the user, since the algorithm may pursue any number of random movements and, as mentioned, may reverse previous successful decryptions. Below is an example of one of the results we worked directly with to finish the decoding manually.

[2] "it thus required kisionary researchers wive gewfand and smith to conkince the community sullorted by lalers that demonstrated through a series of allwications that the method pas easy to understand easy to imlwement and lracticaw" [3] "2041.74999143302"
[1] "8000"

From this example, we see a nearly complete decoding of the message and it is quite recognizable. We can use this to begin working on a manual decryption and extrapolate the cipher function analytically by extrapolating the remaining encodings. This analytic work was successful and provided us with a plausible answer:

"it thus required visionary research like gelfand and smith to convince the community supported by papers that demonstrated through a series of applications that the method was easy to understand easy to implement and practical"

It must be noted that these clear results were not immediately available. The algorithm had to be rerun a number of times before a legible answer was avaible in the results. Moreover, the non linear nature of the decryption process makes it difficult to pinpoint exactly which decoding iteration is best. As such, this algorithm does require a certain level of supervision and post-processing analysis.

Returning to our previous insights, the hardest letter to decipher was "k", which we noted earlier is one of the data sparse characters. Meanwhile, vowels, data rich characters, were correctly deciphered by the algorithm.

We also only see “k” appear once in our text, which made it very difficult to crack. Larger sample texts may yield better results with this algorithm, since more information would be available to cross reference with the transition matrix, M.

Manual completion of the decryption was possible in this case, but is heavily reliance on the skill and knowledge of the user. This text used standard vocabulary and is well written, this will not always be the case. Moreover, this was a basic encryption. Success from manual work cannot be guaranteed or relied upon in more complex problems. However, the algorithm provided critical information, was very fast, and easy to use. May this be an attestation to its value.

Exercise 2

A)

We model the arrivals of cars to the university parking lot with the counting process $\mathbf{N} = \{N_t, t \geq 0\}$ as a non-homogeneous Poisson process. We assume the number of events at the beginning of the process is 0, meaning no cars arrive to the parking lot before it opens at 8 AM. We also assume that for the time after that parking lot is open (i.e., $t > 0$) the number of arrivals to the parking lot (i.e., N_t) have a Poisson distribution with mean:

$$\mathbb{E}[N_t] = \int_0^t \lambda(s)ds$$

Lastly, we assume the increments are independent; however, unlike a homogeneous Poisson process, the increments are not stationary as the rates of arrivals vary with time. Therefore, we determine an intensity function $\lambda(t)$ to express how the arrival rates vary with time.

We are given the following information regarding the rate of arrivals to the parking lot over time:

- Constant average of 100 per hour from 8 to 8:30 AM
- Increases linearly to 250 per hour between 8:30 and 8:45 AM
- Increases linearly to 350 per hour between 8:45 and 9 AM
- Decreases linearly to 100 per hour between 9 and 9:30AM

We can express this given information with the following intensity function $\lambda(t)$. Note time 0 corresponds to 8 AM. The code to obtain the equations is provided in the Appendix.

$$\lambda(t) = \begin{cases} 100, & 0 \leq t \leq 0.5 \\ 600t - 200, & 0.5 < t \leq 0.75 \\ 400t - \ \ 50, & 0.75 < t \leq 1 \\ 850 - 500t, & 1 < t \leq 1.5 \end{cases}$$

As stated earlier, the expected number of arrivals by time t is:

$$\mathbb{E}[N_t] = \int_0^t \lambda(s)ds$$

In order to determine the expectation at any t, we integrate up until that t. For example, we can find the expectation for arrivals by 9:30 AM. The code for evaluating the integrals can be found in the Appendix.

$$\int_0^{0.5} 100\ ds + \int_{0.5}^{0.75} 600s - 200\ ds + \int_{0.75}^{1} 400s - 50\ ds + \int_{1}^{1.5} 850 - 500s\ ds$$

$$50 + 43.75 + 75 + 112.5 = 281.25$$

We see the expected number of of cars by 9:30 AM:

$$\mathbb{E}[N_{1.5}] = 281.25$$

.

We can also see the expected number of cars by 8:30 AM:

$$\mathbb{E}[N_{0.5}] = 50$$

by 8:45 AM:

$$\mathbb{E}[N_{0.75}] = 93.75$$

and by 9 AM:

$$\mathbb{E}[N_1] = 168.75$$

B)

Now, we will determine what time we should arrive at the parking lot in order to find an empty spot 80% of the days. We are given that there are 150 total spots in the parking lot. We assume no cars are in the parking lot before it opens at 8 AM and no cars leave over the 1.5 hour period.

In this step, we will estimate the time without simulation. We do so by examining Poisson distributions at various time points over our time interval of interest. Each distribution has mean $\mathbb{E}[N_t]$ for $t > 0$. For the various distributions, we calculate the probability that there are 149 or less arrivals, which means there is at least one spot open. We select the distribution with mean $\mathbb{E}[N_t]$ such that the probability of 149 or less arrivals is 80%. The corresponding t for this distribution informs us of the appropriate time to arrive at the parking lot to find an empty spot 80% of the time.

First, from the proceeding step a), we can then say the number of arrivals by 8:30, 8:45, 9, and 9:30 AM follow Poisson distributions with means 50, 93.75, 168.75, and 281.25, respectively. From this, we can get a sense of whether or not the parking lot is likely to be full at each of these time points. This helps up close in on a window of time we should further explore, rather than searching through the entire 1.5 hour period.

The code to calculate the following probabilities is provided in the Appendix. Briefly, we find the probabilities of having 149 or less arrivals from the respective cumulative distribution functions. We can see that the probability that there are 149 or less arrivals by 8:30 and 8:45 AM is nearly 1, while the probability by 9 AM is 0.0670. Therefore, we can focus on the window of time between 8:45 and 9 AM in the next steps.

Table 1: Probability of parking lot not full at cutpoints

8:30	1.000
8:45	1.000
9	0.067
9:30	0.000

Now, we will determine the specific time in the window between 8:45 and 9 AM we should arrive to the parking lot. The code to calculate the following is provided in the Appendix. Briefly, we loop through this window of time to determine for which t does the distribution with mean $\mathbb{E}[N_t]$ show a probability of 149 or fewer arrivals of approximately 80%.

We see that this time would be the 10th minute after but including 8:45, which lands us at 8:54 AM. Therefore, we should arrive at the parking lot by 8:54 AM to have at least a 80% chance of finding a empty spot. The specific probability is 0.8800, so we are a bit conservative in our estimate. However, if we arrived by 8:55 AM, we would have a slightly less than 80% chance of finding an empty spot. Specifically, the probability is 0.7658.

Now, we can close in even further and obtain a more precise estimate of when we should arrive to the parking lot. We just saw that we should arrive by 8:54 AM to have at least an 80% chance of finding a spot, with the specific probability being 0.8800. Although this is conservative, if we arrive by 8:55 AM, we will have under a 80% chance, with a probability of 0.7658. To obtain a more precise estimate, we will loop through the the seconds between 8:54 and 8:55. The code for obtaining the following is also provided in the Appendix. We find that the more precise time we should arrive to the parking lot is by 8:54:44 AM with an associated probability of 0.8009.

C)

Now, we write a function to simulate a non-homogeneous process. We do so by simulating a homogeneous Poisson process over the time interval [0,t] and obtain the arrival times S_n where n is the number of events. For the homogeneous Poisson process, we set our λ as the maximum of our intensity function $\lambda(t)$, i.e.,

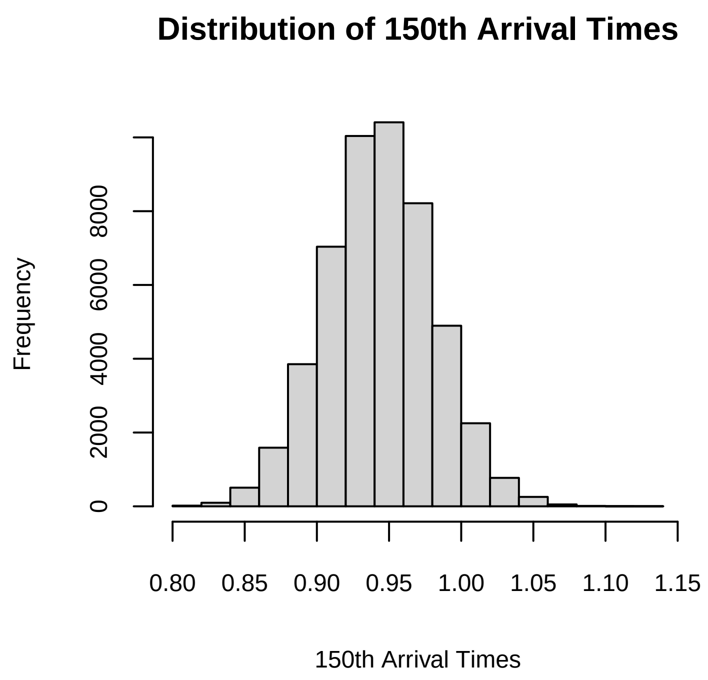
$$\lambda_m = max_{t \geq 0} \lambda(t)$$

The maximum occurs when $t = 1$ and $\lambda(t) = 350$, which corresponds to when function increases linearly to 350 by 9 AM.

However, this overestimates the number of events as we let λ be the maximum of all possible values of $\lambda(t)$. So, we have to decide which events to keep. For for each arrival time S_n, we keep the corresponding events with probability $\frac{\lambda(S_n)}{\lambda_m}$, known as the acceptance probability. The code and description is provided in the Appendix. Briefly, we simulate a homogeneous Poisson process with $\lambda = 350$. We store and sort our arrival times. Then, in order to choose which events to keep, we sample from 0=reject and 1=accept with the probability of drawing 1=accept as the with acceptance probability for each arrival time.

D)

Now, we will use the function we created to simulate our process and confirm our results we obtained without simulation. We simulate 1000 realizations and repeat 50 times. We save the 150th arrival time each time. We're interested in examining the distribution of the 150th arrival times over these simulations as this arrival time corresponds to when the parking lot will be full. We see the distribution of the 150th arrivals is normally distributed with a mean of 8:56:37 AM. Now, we want to determine the time such that the probability of the 150th arrival time being greater than it is 80%. In other words, we find the time such that for 80% of the instances, we arrive earlier than the 150th arrival time. With simulation, we find this time to be 8:54:44 AM. This is the time we found without simulation as well, and are able to verify our results.



Exercise 3

A)

Kind of process

As it has been defined in the statement, $\mathbf{X} = \{X_t, t \geq 0\}$ is a process describing the number of customers in the shoping system (both in the cashiers and the waiting line). In other words, X_t determines the number of customers at time t.

It is said that the number of arrivals to the waiting line follow a Poisson distribution with rate $\lambda \cdot t$ (i.e., it is a Poisson process). By the definition of a Poisson process, we know that the time between arrivals are distributed following an exponential distribution of rate λ. Let us summarize this concept:

$$N_t \sim P(\lambda \cdot t) \ \ \textit{is the number of arrivals to the line}$$

$$T_i \sim exp(\lambda) \ \ \textit{are the arrival times to the line}$$

In the other hand, the times to be served (i.e. to check-out once one has reached the cashier) are said to be distributed as exponential random variables with rate μ.

$$T_i' \sim exp(\lambda) \ \ \textit{are the exit times}$$

With these ideas, we are ready to give a theoretical model to the embedded discrete Markov Chain. Let us define X_n as the number of customers in the n-th interaction (i.e. we forget temporarily about time, and we consider only steps in which the chain makes a transition (acquire or lose one customer)). This chain could be modeled as follows:

$$X_n = \begin{cases} X_{n-1}+1 & \text{if } T_{n-1} < T'_{n-1} \\ X_{n-1}-1 & \text{if } T_{n-1} > T'_{n-1} \end{cases}$$

Since the variables T_n, T'_n and X_n are independent; and the number of customers in the state n only depends on what happened in the previous step (the arrival and exit times and the number of customers there were in the system), we can affirm the embedded process is a Markov Chain.

Now, we can add the concept of the time to the chain. Instead of being interested on the number of customers at each step of the chain, in this case we are interested on the number of customers there are in the system at each time t, where t is continuous. However, conceptually, this does not change the previous reasoning. The probability of being in state j at time t (with $t = s + u$ where s is the time at which the chain made the last change and $0 \leq u < s$) only depends on the state the chain was in at the previous step (i.e. only depends on X_s) and on the minimum between the arrival and the exit times of that transition. Mathematically:

$$P(X_{t+s} = j | X_s = i, X_u = k; 0 \leq u < s) = P(X_{t+s} = j | X_s = i)$$

Thus, the chain verifies the Markov property, and it is defined over the continuous quantity of time. Therefore, we can affirm this is a Continuous Time Markov Chain (CTMC).

We wanted to describe the process from scratch in order to understand it completely. However, the evidence of this process being a continuous time Markov chain could have been simply considered saying that the duration of every state follows an exponential distribution (with rate λ or μ whether the arrival time is greater than the check-out time or not). This is what is typically called a continuous-time Markov chain (the chain is in a state for a certain time and then changes to other state $\rightarrow$ variations between an state and the same one are not considered as changes).

State space

We are said the queue could accommodate an unlimited number of waiting customers. The number of people in the system will be the number of customers waiting in the line and the number of clients being attended. Therefore, the states of the process $\mathbf{X} = \{X_t, t \geq 0\}$ are $\mathbb{N} \cup \{0\}$.

Infinitesimal generator

We have followed this reasoning to form the infinitesimal generator:

- If there are no customers, it has necessarily to change to 1 customer. Then, the element [0,1] of the matrix will be λ. As the rows sum up to 0, the diagonal element [0,0] is $-\lambda$.
- If there is 1 costumer, it could change to 2 or 0. The element [1,2] will be the arrival rate (λ) and the element [1,0] will be the check-out rate (μ). Then, the diagonal [1,1] is $-\lambda - \mu$.
- If there are 2 costumers, it could change to 3 or to 1. The element [2,3] will be λ and the element [2,1] will be 2μ (since there are two cashiers). The diagonal element [2,2] will be $-\lambda - 2\mu$.
- From now on, the rows are constructed equivalently.

Then, our infinitesimal generator, represented as Q, is given by the following expression:

$$
Q = \begin{array}{c c}
\begin{array}{cccccc} \mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \cdots \end{array} & \\
\begin{bmatrix}
-\lambda & \lambda & 0 & 0 & 0 & \cdots \\
\mu & -\lambda-\mu & \lambda & 0 & 0 & \cdots \\
0 & 2\mu & -\lambda-2\mu & \lambda & 0 & \cdots \\
0 & 0 & 2\mu & -\lambda-2\mu & \lambda & \cdots \\
0 & 0 & 0 & 2\mu & -\lambda-2\mu & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{bmatrix} & \begin{array}{c} \mathbf{0} \\ \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \\ \mathbf{4} \\ \vdots \end{array}
\end{array}
$$

B)

Stationary distribution

In order to compute the stationary distribution of this continuous-time Markov chain we consider the proposition that says that if **Q** (an infinitesimal generator) and **Π** (a probability distribution) are in **detailed balance**; then **Π** is the **stationary distribution** of the chain with infinitesimal generator **Q**.

Therefore, we apply the **local balance equations** and we know that implies π is going to be the stationary distribution of the chain.

The local balance equations are:

$$\pi_i \cdot q_{ij} = \pi_j \cdot q_{ji} \qquad \forall i, j \in S$$

Let us start computing those equations:

$$\pi_0 \cdot q_{01} = \pi_1 \cdot q_{10} \rightarrow \pi_0 \cdot \lambda = \pi_1 \cdot \mu \rightarrow \pi_1 = \frac{\lambda}{\mu}\pi_0$$

$$\pi_1 \cdot q_{12} = \pi_2 \cdot q_{21} \rightarrow \pi_1 \cdot \lambda = \pi_2 \cdot 2\mu \rightarrow \pi_2 = \frac{\lambda^2}{2\mu^2}\pi_0$$

$$\pi_2 \cdot q_{23} = \pi_3 \cdot q_{32} \rightarrow \pi_2 \cdot \lambda = \pi_3 \cdot 2\mu \rightarrow \pi_3 = \frac{\lambda^3}{4\mu^3}\pi_0$$

$$\pi_3 \cdot q_{34} = \pi_4 \cdot q_{43} \rightarrow \pi_3 \cdot \lambda = \pi_4 \cdot 2\mu \rightarrow \pi_4 = \frac{\lambda^4}{8\mu^4}\pi_0$$

And so on and so forth. Then we can compute a general expression for the i-th element of the stationary distribution.

$$\pi_i = \frac{\lambda^i}{2^{i-1}\mu^i}\pi_0 \qquad \forall i > 0 \qquad and \;\; \pi_i = \pi_0 \qquad if \;\; i = 0$$

At this point, we have an expression for every element of the stationary distribution in terms of π_0. However, π_0 should be calculated. For this purpose, we are using the property of the stationary distribution as a probability distribution, that implies the sum of all the elements has to be equal to 1.

$$\sum_{i=0}^{\infty} \pi_i = 1$$

Let us define know the quantity $\frac{\lambda}{2\mu} = p$ for convenience in the computations. In the following procedures, we are taking into account the properties of a geometric sum.

$$\pi_0 + 2p\pi_0 + 2p^2\pi_0 + 2p^3\pi_0 + \cdots = 1 \Rightarrow$$

$$\pi_0\left(1 + 2p\sum_{k=0}^{\infty}(p)^k\right) = 1 \Rightarrow$$

$$\pi_0\left(1 + \frac{2p}{1-p}\right) = 1 \Rightarrow$$

$$\pi_0\left(\frac{1+p}{1-p}\right) = 1 \Rightarrow \boxed{\pi_0 = \left(\frac{1-p}{1+p}\right)}$$

At this point, we can give an expression for the stationary distribution of our continuous-time Markov chain.

$$\boxed{\pi_i = 2p^i\left(\frac{1-p}{1+p}\right) \qquad \forall i > 0 \qquad and\ \ \pi_i = \left(\frac{1-p}{1+p}\right) \qquad if\ \ i = 0}$$

Expected number of people in the system

We are asked to compute the expected number of people in the system (including both the waiting line and the checking-out customers). This is equivalent to calculating the expected value of our random variable **X** for a given probability distribution. We can assume this distribution is in fact the stationary one, since we are said to compute the expectation in the long-run. Therefore, the i-th element of the stationary distribution gives us the probability of finding i costumers in the supermarket. Thus, the expectation could be computed as follows:

$$\mathbb{E}[\mathbf{X}] = \sum_{i=0}^{\infty} i \cdot \pi_i$$

$$\mathbb{E}[\mathbf{X}] = \sum_{i=1}^{\infty} i \cdot 2p^i \cdot \frac{1-p}{1+p}$$

$$\mathbb{E}[\mathbf{X}] = 2\left(\frac{1-p}{1+p}\right) \cdot \sum_{i=1}^{\infty} i \cdot p^i$$

Let us focus on the sum term of the previous expression. We can rewrite it as follows:

$$p\sum_{i=1}^{\infty} i \cdot p^{i-1}$$

Under this form, it is clear that the sum is the derivative of the sum of the elements p^i:

$$p \cdot \left(\sum_{i=1}^{\infty} p^i\right)'$$

And the sum between brackets is a geometric sum. Therefore, we can write:

$$p \cdot \left(\frac{1}{1-p}\right)' = p \cdot \left(\frac{1}{(1-p)^2}\right) = \frac{p}{(1-p)^2}$$

Going back to the expression of the expectation, we have:

$$\boxed{\mathbb{E}\left[\mathbf{X}\right] = 2\left(\frac{1-p}{1+p}\right)\frac{p}{(1-p)^2} = \frac{2p}{(1+p)(1-p)}}$$

This is the **expected number of people in the system in the long-run**.

C)

Let us call "tagged" the customer we will focus on (on which we will calculate the probability of overtaking).

As this system has no memory, we start at the moment when "tagged" has to go to a cashier. At that time our client can be found in two scenarios:

- The system is empty (π_0) and therefore the other cashier is empty too. In this case, it is clear that overtaking is impossible.
- The system is not empty and therefore the other cashier is busy. Taking into account the memoryless property of the exponential distribution, at that moment, the probability that the "tagged" passes the client of the other cashier is $\frac{1}{2}$ because both processes follow the same distribution. Let us explain this further; the probability of the other client to spend "x" amount of time given that he/she has already spent "y" amount of time (while "tagged" was waiting) is the probability that he/she spends "x-y" paying. This probability is the same that the "tagged" client spends "x-y" amount of time checking-out (this feature, memoryless, causes both processes to be identically distributed).

When we say that the exponential distribution has no memory, we mean that, for example, if I am already in the cashier, the time that I still have in the system does not vary depending on the time that I have already spent in it. This fact collides with our intuition, because if we imagine ourselves in a supermarket, it seems obvious that the fact of having been waiting for a long time gives me a better chance of leaving earlier.

Let OT be the number of overtaking and S_i the Service time for the i-th customer

$$\boxed{P(OT>0) = P(S_{tag} < S_{tag-1}) \cdot P(\textbf{system not empty}) = \frac{1}{2}(1-\pi_0) = \frac{1}{2}\frac{p}{1+p}}$$

D)

We have created a function that computes a vector with the times customers leave the supermarket. The arguments to pass to this function are the arrival rate (how many customers arrive to the queue per unit time); the check-out time (how many customers leave the cashier per unit time); and the number of customers we are considering in our supermarket.

First, the function computes a vector ("arrival_times") with the arrival times for every customer in the supermarket. However, it is a Poisson process, so the time at which a client will arrive the queue is the arrival time of that client plus the times of all the previous customers. This means that the first client arrives in a time distributed exponentially; then the second arrives in a period distributed exponentially, but starting to count when the first one arrived. Therefore, we have to compute the cumulative sum of the arrival times and that new vector will be the real vector of arrival times (i.e. all components of the vector with the same reference system).

Second, we compute a vector of check-out times ("co_times"). This vector has as many components as number of clients we wanted to consider. Each of the components is exponentially distributed with a the check-out rate. Those are the times each client need to pay.

Third, we create an empty vector called “total_times” where we will save the exit time of each client (starting to count when the first client arrived).

Fourth, we compute the exit times. In order to do this, we use a for loop. We are starting by the 3^{rd} client that arrived to the queue since the first two have a special behavior. At the beginning there were no clients paying, so the first two will have total times equal to their arrival times plus their check-out times. Then the third could face two different situations. First, he/she could find an empty cashier (at least one). Second, he/she could find all cashiers occupied. For modeling this process, we are creating a temporary vector sorting the total times of all previous clients to a specific one (let us denote it as the i-th costumer). This vector contains the (i-1)-th total times sorted in decreasing order (i.e. the two slowest are the first two elements). Then, the i-th client could face two situations. First, the previous second slowest client has longer total time than his arrival time, which means the two cashiers are occupied (only the second slowest because there are two cashiers and it is enough to have one free). Second, the previous second slowest client has shorter total time than his arrival time, which means that cashier is empty. In the first case, the total time of the i-th client will be the total time of the previous second slowest client plus the i-th paying time. In the second case, the total time will be his arrival time plus his check-out time.

Eventually, we create a matrix and save in the first column the arrival times; in the second one, the check-out times; and in the last one, the total times.

The code of the function is provided in the Appendix.

E)

Estimation of the expected number of people in the system in the long-run

We are using the previous function to simulate the times for 10000 customers. In the following table, we have included the times for the first six customers. We already see that the sixth client will overtake the fifth one.

Table 2: Times for the first 6 customers

Arrival times	**Co times**	**Total times**
0.0072	0.0296	0.0367
0.1865	0.0433	0.2298
0.2227	0.0337	0.2563
0.2561	0.0079	0.2640
0.2822	0.3486	0.6308
0.3128	0.0116	0.3245

However, we are firstly interested on estimating the expected number of people in the system in the long-run. For this purpose, we create a vector of times, that will be from 0 to the maximum total time of a client, by steps of 0.25 hours. Then, at each step of this vector, we will compute the number of people and introduce it in another vector (number_people). We are considering that a customer is in the system at a particular time moment (t) when his/her arrival time is smaller than t (i.e. he/she has arrived) and his/her total time is greater than t (i.e. he/she has not left the system). Then, at each moment of time, we compute the number of people in the system.

At this point, we have a vector with the number of customers in the system at each moment of time. In order to study the long-run behavior, we are considering the last seventh part of the times (i.e. if we have a time vector of 1603 components, we are considering only the last 229 of them). Then we compute the mean of the last seventh part of the vector including the number of persons, and that will be the estimated expected number of customers in the long-run.

In order to get a better estimation, we are repeating this process ten times, saving the expected number of persons in the long-run for each trial; and eventually computing the mean of all 10 values. We **obtain the following estimated expected number of customers in the system in the long-run**.

```
## The expected number of people in the system in the long-run is = 4.424
```

Going back to the part B) of the exercise, we got the following theoretical formula for the expected number of people in the long run. In this case, we have $p = \frac{24}{30} = \frac{4}{5}$:

$$\mathbb{E}[\mathbf{X}] = \frac{2p}{(1+p)(1-p)} = \frac{40}{9} = 4.\widehat{4}$$

Comparing the theoretical result with the estimated expected value; we can affirm we got satisfactory results.

Estimation of the probability of overtaking

In the previous section, we had calculated the theoretical probability that a customer will overtake another customer in the system. At this point, Let's do a simulation to check if it matches the theoretically expected value.

As we already have the exit time of each client (third column of the matrix of the function), for each i-th client we check how many of the clients ahead of him (i-1) have an exit time longer that the client in question. That is, the sum of the customers that it has advanced. It should be noted that to impose we are in a steady state, we have counted only the last 5000 clients.

```
## The estimated probability of overtaking is = 0.4544
```

To finish we now compare it with the theoretical probability in terms of p. Where $\frac{\lambda}{2\mu} = \frac{4}{5}$, then:

$$P(OT > 0) = \frac{p}{1+p} = \frac{4/5}{9/5} = \frac{4}{9} = 0.\widehat{4}$$

As we can see, our simulation is very close to the theoretical value of the overtaking probability. The code we have used to estimate this probability is included in the Appendix.

Appendix

Exercise 1

```r
library(ggplot2)
library(reshape2)
library(dplyr)

# Load the transition matrix
mat <- read.table("Englishcharacters.txt",header=F)

# We can normalize a transition matrix using row normalization
transition.matrix_normalized <- as.data.frame(scale(mat + 1, center = FALSE, scale = rowSums(mat + 1)))

# Prepare the transition matrix for visualization
transition.matrix_normalized <- mutate(.data = transition.matrix_normalized, Variable = c(1:27))
```

```r
melt <- melt(transition.matrix_normalized, id.vars = "Variable")

# We visualize our first order frequency data
ggplot(data = melt, aes(x = Variable, y = variable)) + geom_tile(aes(fill = value)) +
  scale_fill_gradient(low="white",high="black",limits=c(0,1))+
  labs(x="Probability of Second Letter",y="Conditioning on First Letter", fill = "Probability",
       title = "Transition Frequencies from Jane Austen") +
  scale_y_discrete(limits = rev(levels(melt$Variable)))
```

```r
# We can also build the transition matrix using log scale
# This will work better with our score function, which requires the product of all our scores
# Using log, we can take the sum
transition.matrix_log <- as.data.frame(log(mat + 1))

# charIndex takes in a character and returns its 'char value'
# defined as a=1, b=2, ..., z=26, space=27
# this matches the array created by read.table
charIndex <- function(char)
{
  aValue <- ascii(char)
  if (aValue == 32)
  {
    # return 27 if a space
    27
  } else
  {
    #ascii sets "a" as 97, so subtract 96
    aValue - 96
  }
}

# ascii(char) returns the numerical ascii value for char
ascii <- function(char)
{
  strtoi(charToRaw(char),16L) #get 'raw' ascii value
}

# We create the score function
score <- function(code) {
  # Sum logs of frequency pairs
  p <- 0
  for (i in 1:(nchar(code)-1)) {
    p <- p + transition.matrix_log[charIndex(substr(code, i, i)), charIndex(substr(code, i+1, i+1))]
  }
  p
} # returns p = log(score(code))


# Decrypts code according to curFunc
decrypt <- function(code,curFunc)
{
  out <- code
  # for each character in the message, decode it according to the curFunc
  for (i in 1:nchar(code))
  {
    charInd <- charIndex(substr(code,i,i))
    if (charInd < 27)
    {
      # change the ith character to the character determined by the curFunc
      substr(out,i,i) <- rawToChar(as.raw(curFunc[charInd] + 96))
    }
  }
  out
}
```

```r
# We load the encrypted message from row 6 of the messages file
codemess <- read.delim("messages.txt", header = F)[6,]

# instantiate a map to hold previously computed codes' scores
map <- new.env(hash=T, parent=emptyenv())

# we begin with a basic (a->a, z->z) function for decrypting the codemess
curFunc <- 1:27
# calculate the score for curFunc and store it in the map
oldScore <- score(decrypt(codemess,curFunc))
map[[paste(curFunc, collapse='')]] <- oldScore
```

```r
# run 1000000 iterations of the Metropolis-Hastings algorithm
for (iteration in 1:1000000) {
  set.seed(iteration)
  # sample two letters to swap
  swaps <- sample(1:26,2)
  oldFunc <- curFunc

  # let curFunc be oldFunc but with two letters swapped
  curFunc[swaps[1]] <- oldFunc[swaps[2]]
  curFunc[swaps[2]] <- oldFunc[swaps[1]]

  # if we have already scored this decoding,
  # retrieve score from our map
  if (exists(paste(curFunc, collapse =''), map)){
    newScore <- map[[paste(curFunc, collapse ='')]]
  } else
    # if we have not already scored this decoding,
    # calculate it and store it in the map
  {
    newScore <- score (decrypt(codemess,curFunc))
    map[[paste(curFunc, collapse = '')]] <- newScore
  }

  # decide whether to accept curFunc or to revert to oldFunc
  if (runif(1) > exp(newScore-oldScore))
  {
    curFunc <- oldFunc
  } else
  {
    oldScore <- newScore
  }

  # print out our decryption every 100 iterations
  if ((iteration %%  2000) == 0)
  {
    print(c(iteration,decrypt(codemess,curFunc), newScore))

  }
}
```

Exercise 2

A)

Obtaining equations for intenisty function

```
line_points=function(x1,y1,x2,y2){
  m=(y2-y1)/(x2-x1)
  b=y1-m*x1
  return(list(m=m,b=b))
}
```

```
# 8:30 to 8:45, points (0.5, 100) & (0.75, 250 )
line_points(0.5, 100,0.75, 250)

# 8:45 to 9, points (0.75, 250) & (1,350)
line_points(1, 350,0.75, 250)

# 9 to 9:30 points (1, 350) & (1.5, 100)
line_points(1, 350,1.5, 100)
```

Integration

```
#f1
100*0.5

f2=function(t){600*t-200}
integrate(f2,0.5,0.75)

f3=function(t){400*t-50}
integrate(f3,0.75,1)

f4=function(t){-500*t+850}
integrate(f4,1,1.5)
```

B)

Probability of 149 or fewer arrivals at various time points

```r
options(digits=4)
# probability of <=149 by:
#830
c1=ppois(149, 50)
#845
c2=ppois(149, 93.75)
#9
c3=ppois(149, 168.75)
#930
c4=ppois(149, 281.25)
```

Estimating time to arrive to the nearest minute

```r
arriv=numeric(16)
sum_arriv=numeric(16)
prob=numeric(16)
for (i in 1:16){
  # We begin by evaluating the upper limit of the integral for each time t
    ## We loop through every minute from 8:45 to 9 AM (inclusive),
    ## corresponding to t between 45/60 or 3/4 and 60/60 or 1.
    ## We choose to loop over natural numbers (1:16) and modify the formula
    ## to arrive at the appropriate calculations (when i=1, 2, 3,...,16 ->
    ## 45/60, 47/60, 48/60,...,60/60 are the values for t ultimately
    ## included in the calculation.
  arriv[i]=200*((44+i)/60)^2-50*((44+i)/60)
  # Next, we evaluate the integral from our lower limit of 8:45 AM (3/4 or 45/60)
     ## to each upper limit t, as calculated by "arriv[i]-arriv[1]."
     ## We then add the the expectation of arrivals by 8:45 AM which was 93.75
     ## to obtain the expectations by time t overall. We then calculate
     ## the probability of 149 or less arrivals for each distribution with the
     ## expectation of arrivals as the means.
  sum_arriv[i]=93.75+arriv[i]-arriv[1]
  prob[i]=ppois(149,sum_arriv[i])
}
# Finally, we determine which distribution shows the closest probability
  ## to 80% without being under 80%.
prob_over80=prob[which(prob>=.8)]
closest_not_under=which(abs(prob_over80-.8)==min(abs(prob_over80-0.8)))
# The closest but not under probability corresponds to the 10th minute, 8:54 AM
minute=closest_not_under
# The associated probability is 0.8800
minute_prob=prob_over80[minute]
# At the 11th minute or 8:55 AM, the associated probability is 0.7658
next_minute_prob=prob[11]
```

Estimating time to arrive to the nearest second

```r
arriv_sec=numeric(61)
sum_arriv_sec=numeric(61)
prob_sec=numeric(61)
# The process is the same as described for the proceeding step, but we update with
     ## following changes:
     ## Modify code to allow for looping through seconds between 8:54 AM
     ## and 8:55 AM (inclusive)
     ## Add expectation of arrivals by 8:54 AM to the sum which 93.75+42
f3=function(t){400*t-50}
int_to54=integrate(f3,0.75,54/60)

for (i in 1:61){
  arriv_sec[i]=200*(54/60+((i-1)/3600))^2-50*(54/60+((i-1)/3600))
  sum_arriv_sec[i]=93.75+42+arriv_sec[i]-arriv_sec[1]
  prob_sec[i]=ppois(149,sum_arriv_sec[i])
}
prob_over80_sec=prob_sec[which(prob_sec>=.8)]
closest_not_under_sec=which(abs(prob_over80_sec-.8)==min(abs(prob_over80_sec-0.8)))
# The closest but not under probability corresponds to the 45th second, which
  ## corresponds to 8:54:44 AM
second=closest_not_under_sec
# At 8:54:44 AM, the associated probability is 0.8009
second_prob=prob_over80_sec[second]
```

C) Function to simulate a non-homegeneous Poisson process

```r
set.seed(1)
lambda_max=350
# Simulation of non-homogeneous Poisson process over time interval
nh_poisson_int<-function(lambda,a=0,b=1.5){
  ## Lambda is the maximum of our intensity function
  ## The function generates over time interval [a,b]
  ## We begin by simulating a homogeneous Poisson process with rate
  ## equal to lambda_max
  t=b-a
  ## We generate lambda_max*t=350*1.5=525 events
  k=rpois(1,lambda=lambda_max*t)
  ## Now, we generate 525 variables from a uniform distribution to obtain
  ## arrival times
  u=runif(k)
  ## We sort the arrival times
  S=t*sort(u)
  ## Now, we need to choose whether or not to keep the event corresponding
  ## to the arrival times. We store our acceptance probabilities in "accept".
  ## We calculate each acceptance probability by dividing the value of our
  ## intensity function at the arrival time by lambda_max. Then, we sample from
  ## 0=reject & 1=accept with acceptance probability being the probability
  ## of drawing 1=accept.
  accept=rep(0,k)
  S_c=rep(0,k)
  ## We also define the intensity function to calculate lambda for
  ## each arrival time.
  lambda_nh=function(t){
    if (t>=0 && t<=0.5){lambda_t=100}
    else if(t>0.5 && t<=0.75){lambda_t=600*t-200}
    else if(t>0.75 && t<=1){lambda_t=400*t-50}
    else {lambda_t=-500*t+850}
    return(lambda_t)}
  for (i in 1:k){
      accept[i]=sample(c(0,1),1,prob=c(1-lambda_nh(S[i])/lambda_max,
                                 lambda_nh(S[i])/lambda_max))
    ## Then, we retain the arrival events for which we drew 1=accept in the prior
    ## step. We store these accepted arrival times in a "S_c" and remove the
    ## NA's/missing values (which correspond to the events we rejected).
    if (accept[i]==1){
      S_c[i]=S[i]}
    else {S_c[i]=NA}
  }
  S_c=S_c[!is.na(S_c)]
  return(list(S_c=S_c))
}
```

D) Simulating the process

```
set.seed(1)
# We simulate 1000 realizations and repeat 50 times. We save the 150th
  ## entry from the arrival times each time.
  ## We create a vector S_150 to store 1000 realizations and S_150_c
  ## to store the repetitions.
S_150=rep(0,1000)
S_150_c=matrix(nrow=50,ncol=1000)
for (k in 1:50){
  for (l in 1:1000){
    S_150[l]=nh_poisson_int(lambda=350)$S_c[150]
    S_150_c[k,]=S_150
  }
}

## We see the distribution of the 150th arrivals is normally
  ## distributed with mean t=0.9437, which corresponds 8:56:37 AM,
  ## and standard deviation 0.0374.
hist(S_150_c, xlab="150th Arrival Times", main="Distribution of 150th Arrival Times")
mean_150=mean(S_150_c)
mean_150_convert_min=mean_150*60
mean_convert_sec=mean_150_convert_min%%1*60
sd_150=sd(S_150_c)

## Now, we find the quantile from the distribution such that the
  ## probability that the 150th arrival time is greater than
  ## it is 0.8.
  ## In other words, we find the time such that 80% of the time we
  ## arrive earlier than the 150th arrival time.
  ## Therefore, we are interested in the quantile with 0.8 the right
  ## and thus 0.2 to the left.
  ## We find the corresponding quantile is t=0.9122, which corresponds
  ## to 8:54:44 AM. This is very close to our estimate without
  ## simulation of 8:54:44 AM.
time=qnorm(0.2, mean_150, sd_150)
time_convert_min=time*60
time_convert_sec=time_convert_min%%1*60
```

Exercise 3

D) Function to simulate the system

```
LeavTim = function(lam,mu,nb,seed){
  # lam is the rate of arrivals (how many customers arrive per unit of time)
  # mu is the rate of check-outs (how many customers exit per unit of time)
  # nb is the number of customers that arrive to the waiting line
  # seed is the seed to generate the random variables

  set.seed(seed)
  arrival_times = rexp(n = nb,rate=lam) # In hours
  arrival_times = cumsum(arrival_times) # It is a Poisson process
  co_times = rexp(n = nb, rate=mu) # In hours

  total_times = rep(0,nb) # Opening the total times expended of each client

  total_times[1:2] = arrival_times[1:2] + co_times[1:2]
  # The 0 of the time affects also to the first and second costumers.
  # I.e. the time spent by the first costumer will be the time he/she spent
  # in the supermarket (the arrival time) plus the time paying (the chock-out time)
  for(i in 3:nb){
    temp_total_times = sort(total_times[1:(i-1)], decreasing = TRUE)
    if(arrival_times[i]<temp_total_times[2]){
      total_times[i] = temp_total_times[2] + co_times[i]
    }
    else{
      total_times[i] = arrival_times[i] + co_times[i]
    }
  }
  M = matrix(0,nrow=nb,ncol=3)
  M[,3] = total_times
  M[,1] = arrival_times
  M[,2] = co_times
  colnames(M) = c("Arrival times","Co times","Total times")
  return(M)
}
```

E) Estimate the expected number of customers in the long-run

```r
vect_means = rep(0,10)
for(j in 1:10){
  seed = j*j
  times = LeavTim(ratearr,rateco,number,seed)
  vect_times = seq(from = 0, to = round(max(times[,3]),0),by = 0.25)

  number_people = rep(0,length(vect_times))
  for(i in 1:length(vect_times)){
    number_people[i] = length(which(times[,1]<vect_times[i]
                                  & times[,3]>vect_times[i]))
  }
  vect_means[j] =
    mean(number_people[(length(vect_times)
                        -round(length(vect_times)/7,0)):length(vect_times)])
}

cat("The expected number of people in the system in the long-run is =",mean(vect_means))
```

E) Estimate the expected probability of overtaking in the long-run

```r
ovtk = 0
k = 5000
for(i in k:(number)){
  ovtk = ovtk + sum(rep(times[,3][i],i-1) < times[,3][1:(i-1)])
}
ovtk = ovtk/(number-k)

cat("The estimated probability of overtaking is =", ovtk)
```