DJILLALI LIABES UNIVERSITY OF SIDI BEL ABBES
FACULTY OF EXACT SCIENCES
DEPARTMENT OF COMPUTER SCIENCES

Module : Modélisation et Simulation
1ST YEAR OF MASTER'S DEGEREE IN
NETWORKS,INFORMATION SYSTEMS & SECURITY(RSSI)
2021/2022

Chane de Markov en temps continu et File d'attente

Students:
HADJAZI M.Hisham
AMUER Wassim
Group: 01 / RSSI

Instructor:
Dr. S.BENBEKRITI

A paper submitted in fulfilment of the requirements for the
Modélisation et Simulation TP-03

December 27, 2021

Contents

List of Figures ii

1 Solutions of Fiche TP-03 1

1.1 Exercise 1 2

1.1.1 On considère la chaîne à temps continu sur l'espace 1,2,3, de générateur infinitésimal A. 2

Construire sous R un générateur infinitésimal 3x3. 2

En diagonalisant A, calculer e^A. 5

Trouver la loi stationnaire de A 6

1.2 Exercise 2 7

1.2.1 Écrire une fonction MMs qui mesure les performances d'une file d'attente M/M/s. 7

Components of a Queueing System 7

1.2.2 Donnez des valeurs a λ, μ, n, s et executez le programme. . . . 8

A R code 9

List of Figures

1.1 generator matrix 3x3 2
1.2 transition matrix 3x3 3
1.3 transition states plot function 4
1.4 transition states 4
1.5 diag function. 5
1.6 expm function. 6
1.7 Steady State 6
1.8 Performance Function 8

Chapter 1

Solutions of Fiche TP-03

Notes regarding this solution :
This solution and the executions of the code in it was done in the following machine :

- *PC* : Lenovo IdeaPad S210 8GBl
- *OS* : Linux Mint 20.2 Cinnamon Kernel v.5.4.0-88
- *IDE* : RStudio 2021.09.0 Build 351
- *R Version* : 3.6.1 (2019-07-05)

This TP was very informative to us specially the second part that relates to the topic of **Queing Theory**, as we researched the topic we understood the variety of different systems from M/M/1 , M/M/s , M/G/1 etc. we learned how to simulate the systems to learn its limits. the beauty of simulations is not only the ease of calculations but the visual presentation of information in graphs and other forms helps better understand the system in hand.

1.1 Exercise 1

1.1.1 On considère la chaîne à temps continu sur l'espace 1,2,3, de générateur infinitésimal A.

Construire sous R un générateur infinitésimal 3x3.

We will be naming our three states space to 1,2,3 and creating a 3x3 matrix with the following values in R.

$$\begin{bmatrix} -2 & 1 & 1 \\ \frac{1}{2} & -1 & \frac{1}{2} \\ 0 & \frac{1}{3} & \frac{-1}{3} \end{bmatrix}$$

```
stateNames <- c("1","2","3")
GeneratorMatrix <- matrix(c(-2,1,1,1/2,-1,1/2,0,1/3,-1/3),
                           nrow=3, byrow=TRUE)
row.names(GeneratorMatrix) <- stateNames; colnames(GeneratorMatrix) <- stateNames
round(GeneratorMatrix,3)
```

The execution of the code gives the following generator matrix :

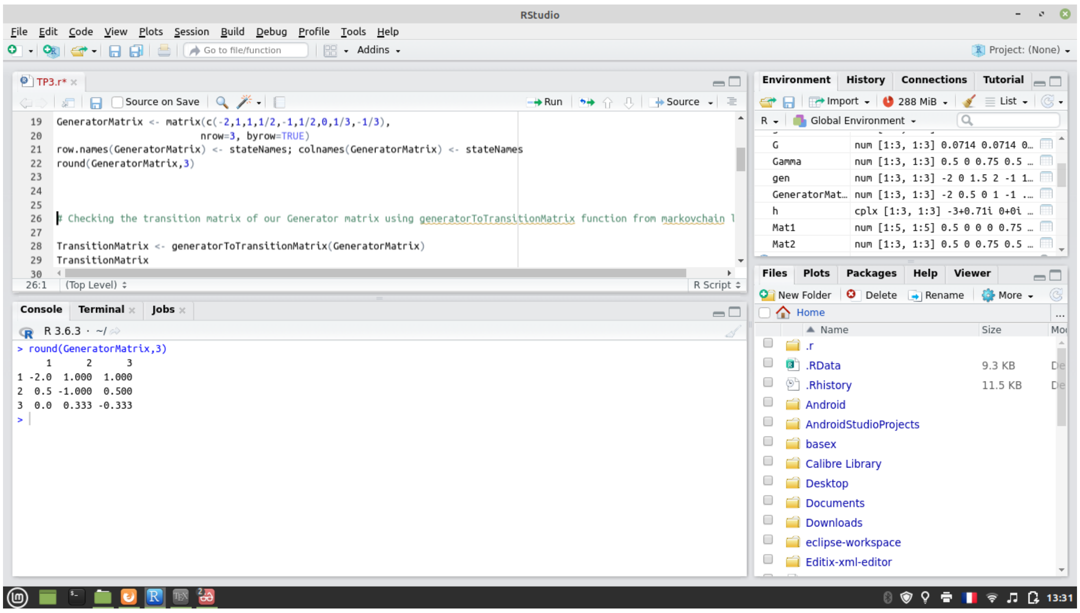

FIGURE 1.1: generator matrix 3x3

We can check the transition matrix of our generator matrix using a handy function called **generatorToTransitionMatrix()**.

```
TransitionMatrix <- generatorToTransitionMatrix(GeneratorMatrix)
TransitionMatrix
```

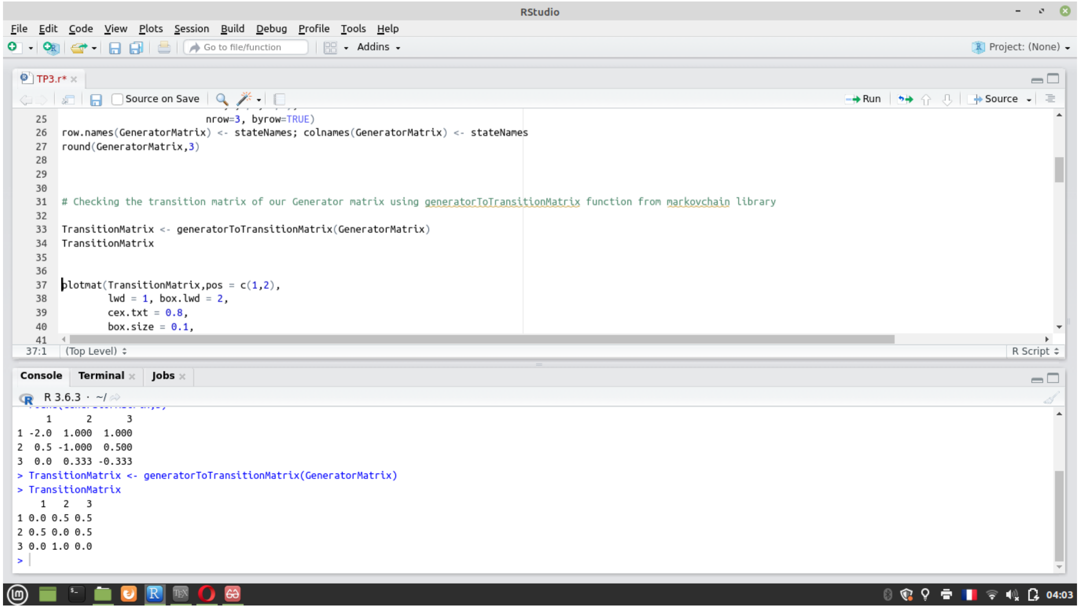

FIGURE 1.2: transition matrix 3x3

Then we obtain the following Transition Matrix :

$$\begin{bmatrix} 0 & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & \frac{1}{3} & 0 \end{bmatrix}$$

To plot the Transition Matrix :

```
plotmat(TransitionMatrix,pos = c(1,2),
        lwd = 1, box.lwd = 2,
        cex.txt = 0.8,
        box.size = 0.1,
        box.type = "circle",
        box.prop = 0.5,
        box.col = "light yellow",
        arr.length=.1,
        arr.width=.1,
        self.cex = .4,
        self.shifty = -.01,
        self.shiftx = .13,
        main = "")
```

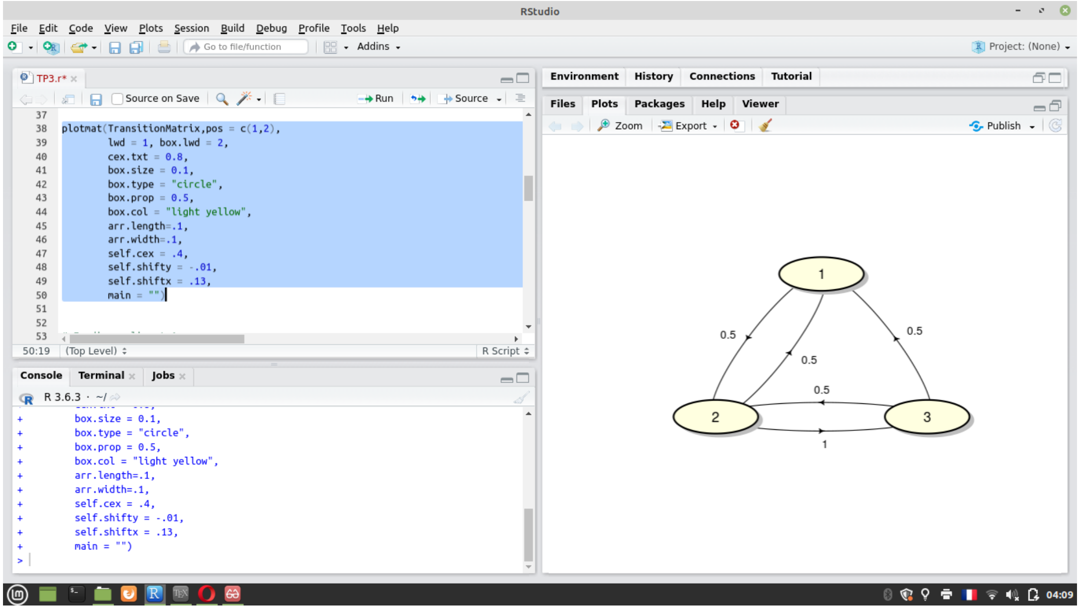

FIGURE 1.3: transition states plot function

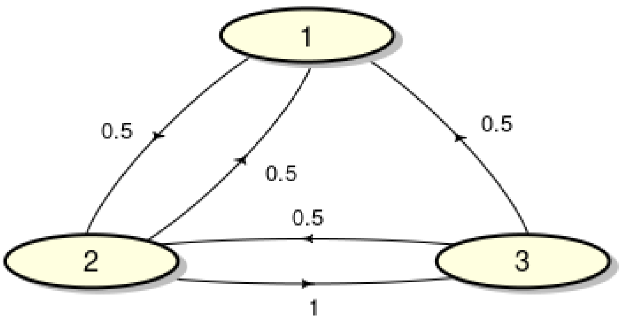

FIGURE 1.4: transition states

En diagonalisant A, calculer e^A.

We can create a function that calculates the eigenvalues and eigenvectors of our matrix to check if our matrix can be diagnosable or not.

```
diagflag = function(m,tol=1e-10){
  x = eigen(m)$vectors
  y = min(abs(eigen(x)$values))
  return(y>tol)
}

diagflag(GeneratorMatrix)
```

The execution of the code gives us **true** which means it is a diagnosable matrix.

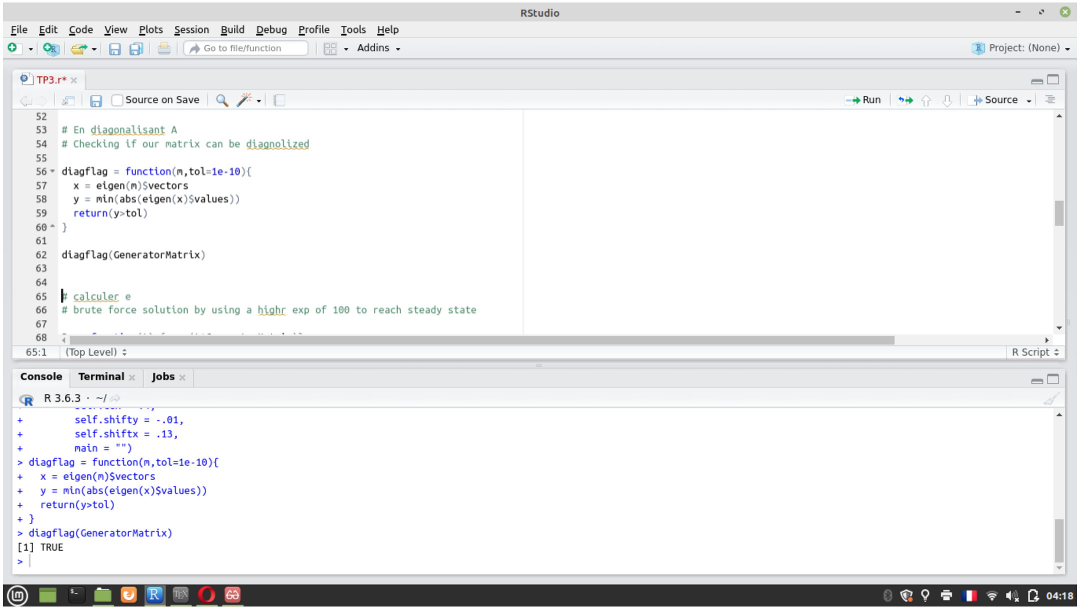

FIGURE 1.5: diag function.

To Calculate the exponent of a matrix we can use the function **expm** if we chose a high number we will reach the steady state by brute force for example the G^{100} will return the steady state matrix

$$\begin{bmatrix} 0.071 & 0.286 & 0.643 \\ 0.071 & 0.286 & 0.643 \\ 0.071 & 0.286 & 0.643 \end{bmatrix}$$

```
P <- function(t) {expm(t*GeneratorMatrix)}
expMatrix <- P(100)
round(expMatrix,3)
```

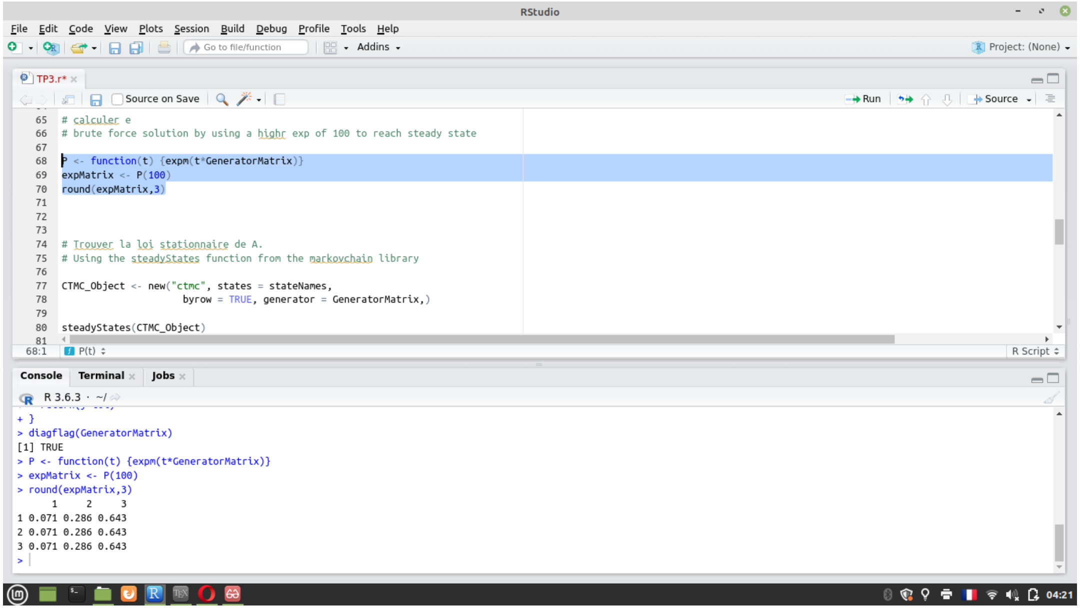

FIGURE 1.6: expm function.

Trouver la loi stationnaire de A

The stationary or stable state can be can be obtained like we did by brute force with expm function or by using a built in function in the markovchain library called **steadyStates()**. to use it we need to convert our generator matrix to a ctmc s4 object.

```
CTMC_Object <- new("ctmc", states = stateNames,
                   byrow = TRUE, generator = GeneratorMatrix,)

steadyStates(CTMC_Object)
```

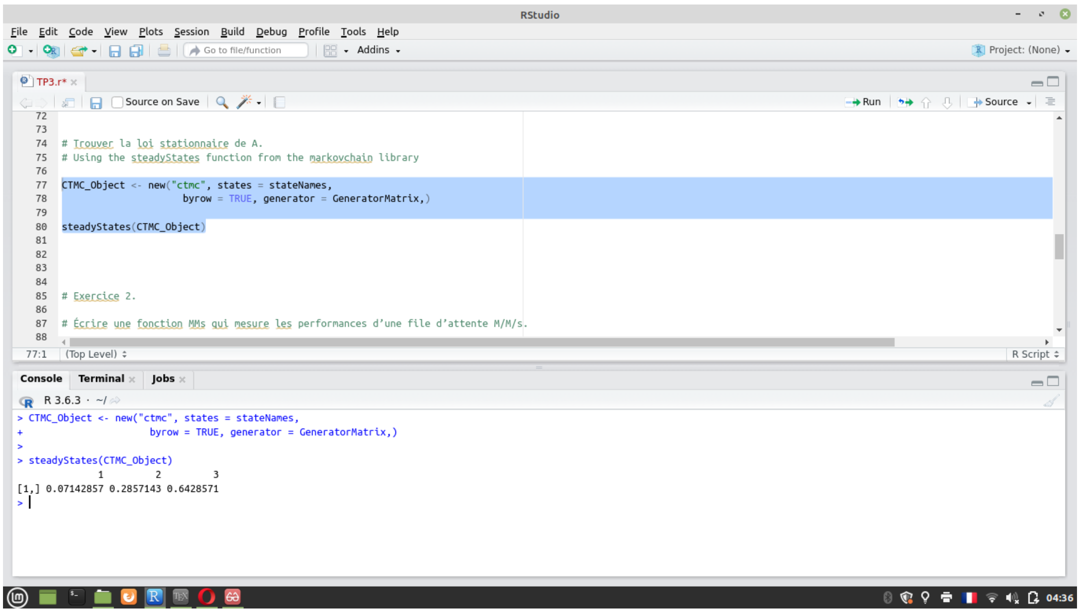

FIGURE 1.7: Steady State

1.2 Exercise 2

1.2.1 Écrire une fonction MMs qui mesure les performances d'une file d'attente M/M/s.

Queueing theory is the mathematical study of waiting in lines, or queues. Queueing theory, along with simulation, are the most widely used operations-research and management-science techniques. Its main objective is to build a model to predict queue lengths and waiting times to make effective business decisions related to resources' management and allocation to provide a given service.

Components of a Queueing System

A queueing system is characterized by three components: arrival process, service mechanism, and queue discipline.

- **Arrival process:** describes how the customers arrive to the system, and the distribution of the customers' arrival
- **Service mechanism:** is articulated by the number of servers, and whether each server has its own queue or there is one queue feeding all servers, and the distribution of customer's service times
- **Queue discipline:** refers to the rule that a server uses to choose the next customer from the queue when the server completes the service of the current customer (e.g. FIFO: first-in, first-out; LIFO: last-in, first-out; priority-based; random selection)

The M/M/s system for multiple servers is very similar to the M/M/1 in principle. we used the formulas provided in our course slides to create the function rather than using any available functions since it is very easy to create one by just applying the formulas. but we must note that there are functions from libraries that do this with more details and extra functionality like graphing. two good examples are **library(queueing)** and **library(simmer)**.

```
rm(list=ls())

mms_performance <- function(lambda, mu, n, s){
  Utilization <- (lambda/mu)/n
  Average_num_of_customers_in_the_system <- lambda/(mu-lambda)
  Average_num_of_customers_in_the_queue <- Utilization * Average_num_of_customers_in_the_system
  Average_time_a_customer_spends_in_the_system <- n / (mu-lambda)
  Average_time_a_customer_spends_in_the_queue <- Average_num_of_customers_in_the_queue/lambda

  X <- data.frame(Utilization, Average_num_of_customers_in_the_system, Average_num_of_customers_in_the_queue, Average_time_a_customer_spends_in_the_system, Average_time_a_customer_spends_in_the_queue)

  names(X)<-c('Utilization','Average_num_of_customers_in_the_system','Average_num_of_customers_in_the_queue','Average_time_a_customer_spends_in_the_system','Average_time_a_customer_spends_in_the_queue')
  return(X)
```

```
}
mms_performance(15, 20, 2)
```

1.2.2 Donnez des valeurs a λ, μ, n, s et executez le programme.

λ (lambda): average number or arrivals per time period = 15
μ (mu): average number of customers served per time period = 20
n (n): number of servers = 2
s (Time): Duration = Infinity

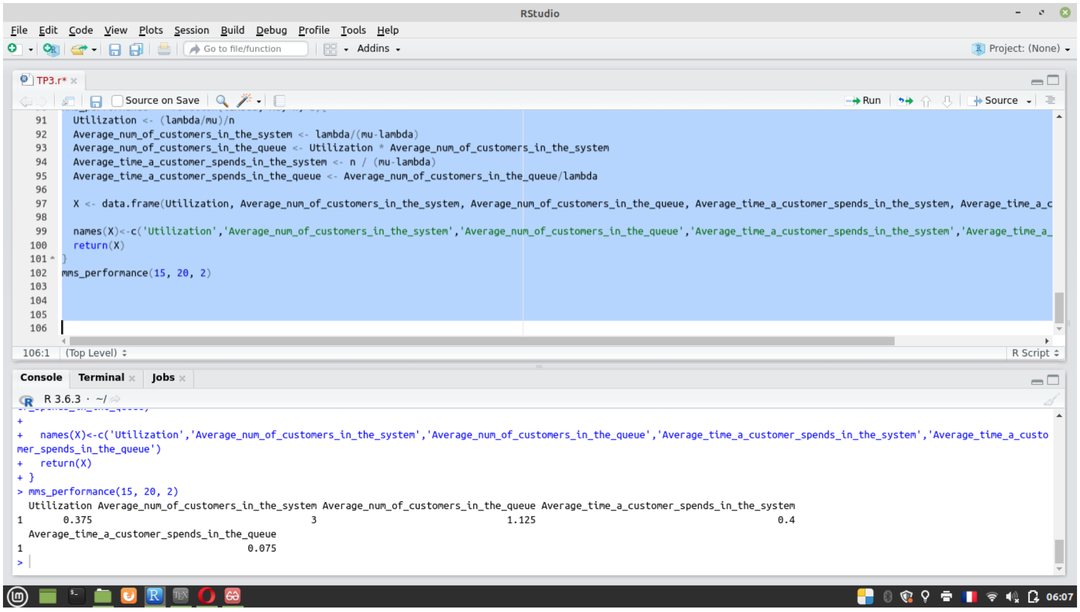

FIGURE 1.8: Performance Function

Our results are per hour, so as we see :

- **Utilization per server :** 37.5%
- **Average number of customers in the system :** 3 customer per hour.
- **Average number of customers in the queue :** 1.125 customer per hour.
- **Average time a customer spends in the system :** 0.4 hour or 24 min.
- **Average time a customer spends in the queue :** 0.075 hour or 4 minutes and 30 seconds.

Appendix A

R code

```
library(markovchain)
library(ctmcd)
library(diagram)
library(pracma)
library(Matrix)
library(expm)





# Exercice 1.
# On considere la chaine a temps continu sur l espace {1,2,3}, de generateur infinitesimal A.


stateNames <- c("1","2","3")


# Construire sous R un generateur infinitesimal 3x3.

GeneratorMatrix <- matrix(c(-2,1,1
                            ,1/2,-1,1/2
                            ,0,1/3,-1/3),
                         nrow=3, byrow=TRUE)
row.names(GeneratorMatrix) <- stateNames; colnames(GeneratorMatrix) <- stateNames
round(GeneratorMatrix,3)



# Checking the transition matrix of our Generator matrix using generatorToTransitionMatrix function from markovchain library

TransitionMatrix <- generatorToTransitionMatrix(GeneratorMatrix)
TransitionMatrix

# we can plot or transition matrix

plotmat(TransitionMatrix,pos = c(1,2),
        lwd = 1, box.lwd = 2,
        cex.txt = 0.8,
        box.size = 0.1,
        box.type = "circle",
        box.prop = 0.5,
```

```r
        box.col = "light yellow",
        arr.length=.1,
        arr.width=.1,
        self.cex = .4,
        self.shifty = -.01,
        self.shiftx = .13,
        main = "")


# En diagonalisant A
# Checking if our matrix can be diagnolized

diagflag = function(m,tol=1e-10){
  x = eigen(m)$vectors
  y = min(abs(eigen(x)$values))
  return(y>tol)
}

diagflag(GeneratorMatrix)


# calculer e
# brute force solution by using a highr exp of 100 to reach steady state

P <- function(t) {expm(t*GeneratorMatrix)}
expMatrix <- P(100)
round(expMatrix,3)



# Trouver la loi stationnaire de A.
# Using the steadyStates function from the markovchain library

CTMC_Object <- new("ctmc", states = stateNames,
                     byrow = TRUE, generator = GeneratorMatrix,)

steadyStates(CTMC_Object)




# Exercice 2.

# Ecrire une fonction MMs qui mesure les performances d une file d attente M/M/s.


rm(list=ls())

mms_performance <- function(lambda, mu, n, s){
  Utilization <- (lambda/mu)/n
  Average_num_of_customers_in_the_system <- lambda/(mu-lambda)
  Average_num_of_customers_in_the_queue <- Utilization * Average_num_of_customers_in_the_system
  Average_time_a_customer_spends_in_the_system <- n / (mu-lambda)
```

```
  Average_time_a_customer_spends_in_the_queue <- Average_num_of_customers_in_the_queue/lambda

  X <- data.frame(Utilization, Average_num_of_customers_in_the_system, Average_num_of_customers_in_the_queue, Average_time_a_customer_spends_in_the_system, Average_time_a_customer_spends_in_the_queue)

  names(X)<-c('Utilization','Average_num_of_customers_in_the_system','Average_num_of_customers_in_the_queue','Average_time_a_customer_spends_in_the_system','Average_time_a_customer_spends_in_the_queue')
  return(X)
}
mms_performance(15, 20, 2)
```

Bibliography

[1] Robert P. Dobrow. *Introduction to stochastic processes with R*. Hoboken, New Jersey: Wiley, 2016. ISBN: 9781118740651.

[2] Mor Harchol-Balter. *Performance modeling and design of computer systems: queueing theory in action*. Cambridge: Cambridge University Press, 2013. ISBN: 9781107027503.

[3] Owen Jones, Robert Maillardet, and Andrew Robinson. *Introduction to scientific programming and simulation using R*. Second edition. The R series. OCLC: ocn885074437. Boca Raton, FL: CRC Press, Taylor & Francis Group, 2014. ISBN: 9781466569997 9781466570016.

[4] Roberto Salazar. *Queueing Models with R*. July 2021. URL: https://towardsdatascience.com/queueing-models-with-r-a794c78e6820.