Exercises from Chapter 2 of Introduction to Stochastic Processes with R by Robert P. Dobrow

Ojeda Contreras Braulio Melquisedec

2022-11-15

2.23) Simulate the first 20 letters (vowel/consonant) of the Pushkin poem Markov chain of Example 2.2

```
simulate_pushkin_poem <- function(n) {

  consonants <- c('b', 'c', 'd', 'f', 'g', 'h', 'j', 'k', 'l', 'm', 'n', 'p', 'q',
                  'r', 's', 't', 'v', 'w', 'x', 'y', 'z')
  vowels <- c('a', 'e', 'i', 'o', 'u')

  letters <- character()

  i <- 1

  init_prob_cons <- 11362 / 20000
  vv_prob <- 1104 / 8638
  cv_prob <- 7535 / 11362
  vc_prob <- 7534 / 8638
  cc_prob <- 3827 / 11362


  init_letter <- sample(c(consonants, vowels), 1,
                        prob = c(rep(init_prob_cons, length(consonants)),
                                 rep(1 - init_prob_cons, length(vowels))))
  letters[i] <- init_letter

  if (n > 1) {
    for (i in 2:n) {
      prev_letter <- letters[i - 1]

      if (prev_letter %in% vowels) {
        letter <- sample(c(consonants, vowels), 1,
                         prob = c(rep(vc_prob, length(consonants)),
                                  rep(vv_prob, length(vowels))))
      }
      else {
        letter <- sample(c(consonants, vowels), 1,
                         prob = c(rep(cc_prob, length(consonants)),
                                  rep(cv_prob, length(vowels))))
```

```
      }
      letters[i] <- letter
    }
  }
  return (letters)
}

first_20_letters <- simulate_pushkin_poem(n = 20)

print(first_20_letters)
```

```
##  [1] "g" "k" "h" "e" "k" "n" "h" "k" "a" "g" "d" "u" "j" "m" "a" "k" "h" "o" "y"
## [20] "u"
```

2.24) Simulate 50 steps of the random walk on the graph in Figure 2.1. Repeat the simulation 10 times. How many of your simulations end at vertex c? Compare with the exact long-term probability the walk visits c.

```
markov <- function(init,mat,n,labels) {
  if (missing(labels)) labels <- 1:length(init)
  simlist <- numeric(n+1)
  states <- 1:length(init)
  simlist[1] <- sample(states,1,prob=init)
  for (i in 2:(n+1))
  { simlist[i] <- sample(states,1,prob=mat[simlist[i-1],]) }
  labels[simlist]
}

mat <- matrix(c(0, 1, 0, 0, 0, 0,
                1/4, 0, 1/4, 1/4, 1/4, 0,
                0, 1/4, 0, 1/4, 1/4, 1/4,
                0, 1/4, 1/4, 0, 1/4, 1/4,
                0, 1/3, 1/3, 1/3, 0, 0,
                0, 0, 1/2, 1/2, 0, 0), ncol = 6, byrow = TRUE)

init <- c(1, 0, 0, 0, 0, 0)
n <- 50
labels <- c('a', 'b', 'c', 'd', 'e', 'f')
trials <- 100000
res <- replicate(trials, markov(mat = mat, init = init,
                                n = n, labels = labels))
last_steps <- res[51,]
end_c <- length(last_steps[last_steps == 'c'])
cat('The probability of ending at vertex c is', end_c / trials)
```

```
## The probability of ending at vertex c is 0.22271
```

2.25 The behavior of dolphins in the presence of tour boats in Patagonia, Argentina is studied in Dans et al. (2012). A Markov chain model is developed, with state space consisting of five primary dolphin activities (socializing, traveling, milling, feeding, and resting). Considering the following transition matrix is obtained, use technology to estimate the long-term distribution of dolphin activity.

```
# Consider the entries order as socializing, traveling, milling, feeding and resting
mat_dolph_bhv <- matrix(c(0.84, 0.11, 0.01, 0.04, 0.00,
                          0.03, 0.80, 0.04, 0.10, 0.03,
                          0.01, 0.15, 0.70, 0.07, 0.07,
                          0.03, 0.19, 0.02, 0.75, 0.01,
                          0.03, 0.09, 0.05, 0.00, 0.83), ncol = 5, byrow = TRUE)
```

To find the transition matrix after a huge number n of steps, we are going to consider an error of 10^{-8} order. So, if the all the differences between each entry of the n-1 steps matrix and the n steps one is least than the error, we are going to stop iterating.

```
error <- 10 ** -8
i <- 1
A_n_minus_1 <- mat_dolph_bhv
number_entries <- dim(mat_dolph_bhv)[1] * dim(mat_dolph_bhv)[2]
flag <- TRUE
while (flag) {
  A_n <- A_n_minus_1 %*% mat_dolph_bhv
  aux_mat <- A_n - A_n_minus_1
  aux_count <- length(aux_mat[abs(aux_mat) < error])
  if (aux_count == number_entries) {
    flag <- FALSE
  }
  else {
    i <- i + 1
    A_n_minus_1 <- A_n
  }
}
cat("In", i, "iterations, the matrix transition converges to the following matrix:\n")
```

```
## In 87 iterations, the matrix transition converges to the following matrix:
```

```
print(A_n)
```

```
##           [,1]      [,2]       [,3]      [,4]      [,5]
## [1,] 0.1478358 0.4149254 0.09555969 0.2163806 0.1252985
## [2,] 0.1478358 0.4149254 0.09555970 0.2163806 0.1252985
## [3,] 0.1478358 0.4149254 0.09555970 0.2163806 0.1252985
## [4,] 0.1478358 0.4149254 0.09555970 0.2163806 0.1252985
## [5,] 0.1478358 0.4149254 0.09555971 0.2163806 0.1252985
```

2.26) In computer security applications, a honeypot is a trap set on a network to detect and counteract computer hackers. Honeypot data are studied in Kimou et al. (2010) using Markov chains. The authors obtain honeypot data from a central database and observe attacks against four computer ports—80, 135, 139, and 445—over 1 year. The ports are the states of a Markov chain along with a state corresponding to no port is attacked. Weekly data are monitored, and the port most often attacked during the week is recorded.

```
# The estimated Markov transition matrix for weekly attacks has states order given by port number 80, 1
mat_attacked_ports <- matrix(c(0, 0, 0, 0, 1,
                          0, 8/13, 3/13, 1/13, 1/13,
                          1/16, 3/16, 3/8, 1/4, 1/8,
                          0, 1/11, 4/11, 5/11, 1/11,
                          0, 1/8, 1/2, 1/8, 1/4), ncol = 5, byrow = TRUE)

# And the initial distribution is
init = c(0, 0, 0, 0, 1)
```

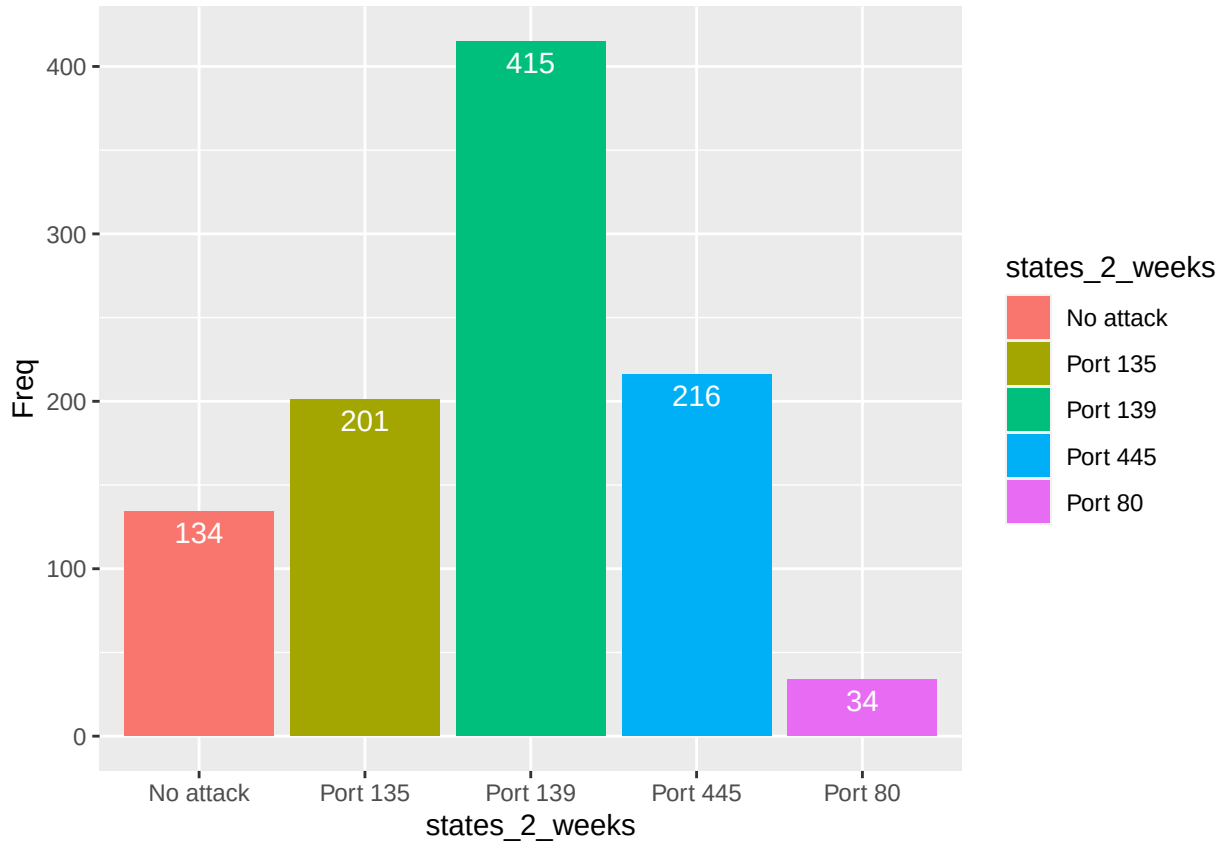
(a) Which are the least and the most likely attacked ports after 2 weeks?

```
#library(tidyverse)
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.2.2
```

```
n <- 2
labels <- c('Port 80', 'Port 135', 'Port 139', 'Port 445', 'No attack')
trials <- 1000
res <- replicate(trials, markov(mat = mat_attacked_ports,init = init,
                                n = n, labels = labels))
states_2_weeks <- res[3,]
counts <- as.data.frame(table(states_2_weeks))
ggplot(counts, aes(x=states_2_weeks, y=Freq, fill=states_2_weeks)) +
  geom_bar(stat = "identity") +
  geom_text(aes(label = Freq), vjust = 1.5, color = "white") +
  labs(cat("Attacked ports after 2 weeks (", trials, " trials )"))
```

```
## Attacked ports after 2 weeks ( 1000  trials )
```



So after two weeks, the most likely attacked port is port 139 and the least likely is port 80.

(b) Find the long-term distribution of attacked ports.

As in the previous exercise, we are going to consider an error of 10^{-8} order. So, if the all the differences between each entry of the n-1 steps matrix and the n steps one is least than the error, we are going to stop iterating.

```
error <- 10 ** -8
i <- 1
A_n_minus_1 <- mat_attacked_ports
number_entries <- dim(mat_attacked_ports)[1] * dim(mat_attacked_ports)[2]
flag <- TRUE
while (flag) {
  A_n <- A_n_minus_1 %*% mat_attacked_ports
  aux_mat <- A_n - A_n_minus_1
  aux_count <- length(aux_mat[abs(aux_mat) < error])
  if (aux_count == number_entries) {
    flag <- FALSE
  }
  else {
    i <- i + 1
    A_n_minus_1 <- A_n
  }
}
cat("In", i, "iterations, the matrix transition converges to the following matrix:\n")
```

```
## In 25 iterations, the matrix transition converges to the following matrix:
```

```
print(A_n)
```

```
##            [,1]      [,2]      [,3]      [,4]   [,5]
## [1,] 0.02146667 0.2669333 0.3434667 0.2273333 0.1408
## [2,] 0.02146667 0.2669333 0.3434667 0.2273333 0.1408
## [3,] 0.02146667 0.2669333 0.3434667 0.2273333 0.1408
## [4,] 0.02146667 0.2669333 0.3434667 0.2273333 0.1408
## [5,] 0.02146667 0.2669333 0.3434667 0.2273333 0.1408
```

2.27) See gamblersruin.R. Simulate gambler's ruin for a gambler with initial stake $2, playing a fair game.

(a) Estimate the probability that the gambler is ruined before he wins $5.

```
# gamblersruin.R
# Example 1.11

# gamble(k, n, p)
  #   k: Gambler's initial state
  #   n: Gambler plays until either $n or Ruin
  #   p: Probability of winning $1 at each play
  #   Function returns 1 if gambler is eventually ruined
  #                    returns 0 if gambler eventually wins $n

gamble <- function(k, n, p, goal) {

    stake <- k
    stake_historic <- integer()
    reach_goal <- FALSE
    i <- 1

    while (stake > 0 & stake < n) {
        bet <- sample(c(-1, 1), 1, prob = c(1 - p, p))
        stake <- stake + bet
        stake_historic[i] <- stake
        if (stake == goal)
          reach_goal = TRUE
        i <- i + 1
    }

    if (stake == 0)
       return(list(1, reach_goal))
    else
      return(list(0, reach_goal))
}

k <- 2
n <-  1000 # Let's assign a big number to represent that the rival has so much money
p <- 0.50
```

```r
goal <- 5
trials <- 1000

res <- replicate(trials, gamble(k, n, p, goal))
res_not_reach_goal <- res[2, res[1,] == 1 & res[2,] == FALSE]
prob_not_reach_goal <- length(res_not_reach_goal) / trials

cat("The probability that the gambler is ruined before reaching the goal of $",
    goal, "is", prob_not_reach_goal, "\nwith p =", p, ", i.e. a fair game")
```

```
## The probability that the gambler is ruined before reaching the goal of $ 5 is 0.577
## with p = 0.5 , i.e. a fair game
```

(b) Construct the transition matrix for the associated Markov chain. Estimate the desired probability in (a) by taking high matrix powers.

The states of this matrix are going to be the possible wealthy values before reaching the goal of 5, i.e. 0, 1, 2, 3, 4, 5 monetary units.

```r
trans_matrix_gamblers_ruin <- matrix(c(1, 0, 0, 0, 0, 0,
                                       1 - p, 0, p, 0, 0, 0,
                                       0, 1 - p, 0, p, 0, 0,
                                       0, 0, 1 - p, 0, p, 0,
                                       0, 0, 0, 1 - p, 0, p,
                                       0, 0, 0, 0, 0, 1),
                                       ncol = 6, byrow = TRUE)
print(trans_matrix_gamblers_ruin)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]  1.0  0.0  0.0  0.0  0.0  0.0
## [2,]  0.5  0.0  0.5  0.0  0.0  0.0
## [3,]  0.0  0.5  0.0  0.5  0.0  0.0
## [4,]  0.0  0.0  0.5  0.0  0.5  0.0
## [5,]  0.0  0.0  0.0  0.5  0.0  0.5
## [6,]  0.0  0.0  0.0  0.0  0.0  1.0
```

As it could be seen in previous exercises, we are going to consider an error of 10^{-8} order. So, if the all the differences between each entry of the n-1 steps matrix and the n steps one is least than the error, we are going to stop iterating.

```r
error <- 10 ** -8
i <- 1
A_n_minus_1 <- trans_matrix_gamblers_ruin
number_entries <- dim(trans_matrix_gamblers_ruin)[1] * dim(trans_matrix_gamblers_ruin)[2]
flag <- TRUE
while (flag) {
  A_n <- A_n_minus_1 %*% trans_matrix_gamblers_ruin
  aux_mat <- A_n - A_n_minus_1
  aux_count <- length(aux_mat[abs(aux_mat) < error])
  if (aux_count == number_entries) {
    flag <- FALSE
```

```
  }
  else {
    i <- i + 1
    A_n_minus_1 <- A_n
  }
}

# print(A_n)
df_trans_matrix_gamblers_ruin = data.frame(round(A_n, 2))
rownames(df_trans_matrix_gamblers_ruin) <- c("wealthy_0", "wealthy_1",
                                             "wealthy_2", "wealthy_3",
                                             "wealthy_4", "wealthy_5")
colnames(df_trans_matrix_gamblers_ruin) <- c("wealthy_0", "wealthy_1",
                                             "wealthy_2", "wealthy_3",
                                             "wealthy_4", "wealthy_5")
cat("In", i, "iterations, the matrix transition converges to the following matrix:\n")
```

```
## In 86 iterations, the matrix transition converges to the following matrix:
```

```
print(df_trans_matrix_gamblers_ruin)
```

```
##           wealthy_0 wealthy_1 wealthy_2 wealthy_3 wealthy_4 wealthy_5
## wealthy_0       1.0         0         0         0         0       0.0
## wealthy_1       0.8         0         0         0         0       0.2
## wealthy_2       0.6         0         0         0         0       0.4
## wealthy_3       0.4         0         0         0         0       0.6
## wealthy_4       0.2         0         0         0         0       0.8
## wealthy_5       0.0         0         0         0         0       1.0
```

Note that with the n-steps matrix we could see that if we start with a wealthy of 2 we would get eventually ruined with a probability of 0.6, approximately the value calculated in 2.27a) previous subsection. This also could be understood by considering the complement of reaching wealthy of 5 starting with wealthy of 2, i.e. $1 - 0.4 = 0.6$.

(c) Compare your results with the exact probability.

Remembering that the exact probability is given by the quotient of n - k divided by n, where for this case $n = 5$ (desired wealthy) and $k = 2$ (initial wealthy), we get that the exact probability is 3 / 5 = 0.6. Thus, the probabilities shown above are correct.