STATS 782 Assignment 2

Chase Robertson - crob873

1 April 2022

I have read the declaration on the cover sheet and confirm my agreement with it.

Question 1

Pascal's triangle looks like this:

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

The values in each row, apart from the edges which are always 1, are found by adding the two adjacent numbers from the previous row.

(a) Write a function called pascal() that takes an argument n and then produces the first n rows of Pascal's triangle, where n > 0. Each row of the triangle should be a numeric vector, and the final result returned by the function should be a list of such vectors.

```r
pascal <- function(n) {
  # start with peak of triangle
  prev_level = c(1)
  result <- list(prev_level)

  # for each descending level, construct based on previous
  while (length(result) < n) {
    # start a new level with 1 on left edge
    new_level = c(1)

    # fill new level with previous levels' summed neighbors
    j = 2
    while (j <= length(prev_level)) {
      new_level[j] = prev_level[j-1] + prev_level[j]
      j = j+1
    }

    # 1 on new level's right edge
    new_level[j] = 1

    # add new level as bottom of triangle thus far
    result[[length(result)+1]] = new_level
    prev_level = new_level
  }

  result
}

pascal(6)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 1 1
##
## [[3]]
## [1] 1 2 1
##
## [[4]]
## [1] 1 3 3 1
##
## [[5]]
## [1] 1 4 6 4 1
##
## [[6]]
## [1]  1  5 10 10  5  1
```

Pascal's triangle is generated from the given n by repeatedly adding a new numeric vector to a list, where each vector starts and ends with 1, with its middle numbers being the sum of the two numbers above it in the triangle.

Question 2

In this question you will implement a very simple version of 'Approximate Bayesian Computation' — a method for inferring parameters from data. It will seem like a 'guess and check' sort of algorithm that hopefully matches common sense. Suppose that there are some data values x1, x2, …, xn and that a Cauchy distribution is appropriate:

$$x1, x2, ..., xn| \sim Cauchy(, 1).$$

(a) Write a function to generate 7 values from a Cauchy distribution with location parameter ('centre') µ, given as an argument. You can use rt() for this, since a Cauchy distribution is the same thing as a t-distribution with df=1.

```
my_rcauchy <- function(centre, n=7) {
  rt(n, df=1, ncp=centre)
}
my_rcauchy(centre=5)
```

```
## [1]  4.725815  7.344699  8.713119  4.517756 23.130768 14.262407  2.481066
```

The function includes an n parameter which defaults to 7, so the function can generate sets of varying length.

(b) Suppose x = {7, 4, 10, 11, 6} is observed. The median of these values is 7, and you are going to generate possible scenarios with median close to 7, to see what µ values are compatible with this observation. Write a function that generates a possible µ value between 0 and 20 uniformly, and simulates a dataset using it, until the median is between 6.99 and 7.01, returning the value of µ that was used when that happened.

```r
munif <- function() {
  mu = Inf
  sim_med = Inf
  # loop forever until a valid mean is found
  while (sim_med < 6.99 || sim_med > 7.01) {
    # generate a cauchy set using some random mean
    mu = runif(1, 0, 20)
    sim_set = my_rcauchy(centre=mu, n=5)

    # update simulated median
    sim_med = median(sim_set)
  }
  mu
}
munif()
```

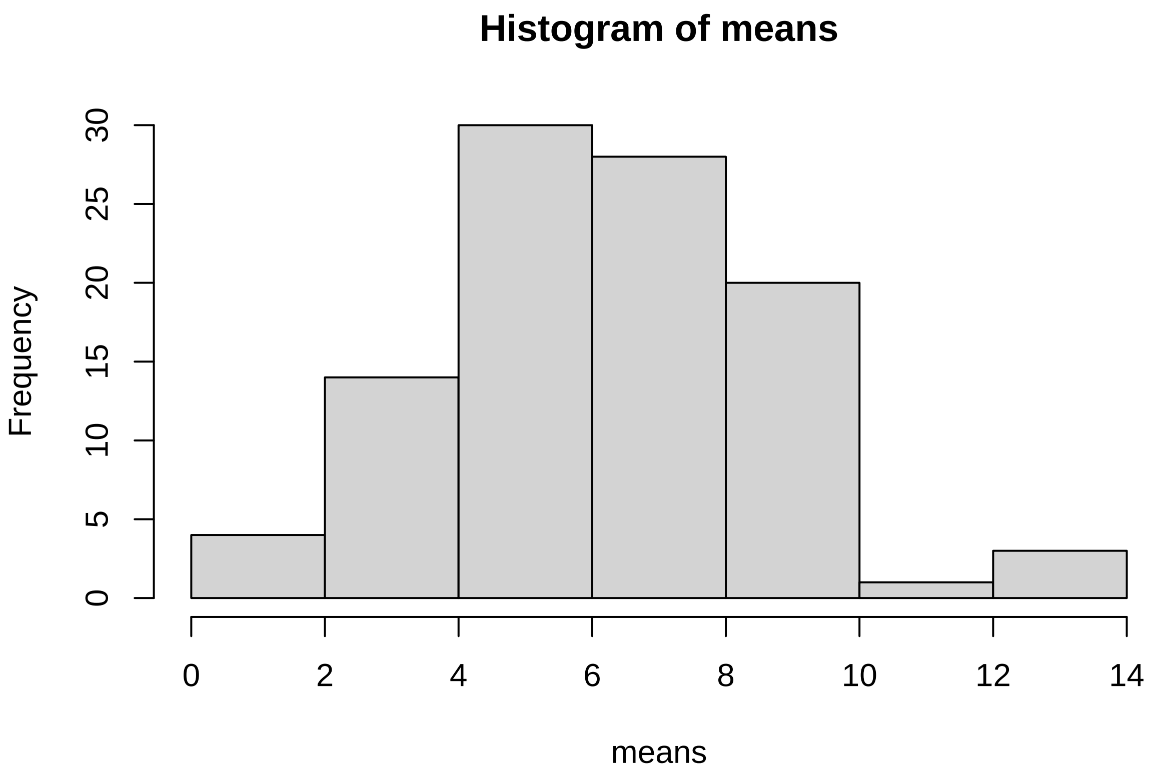
```
## [1] 5.461188
```

This function, beginning with some infinite mean and median values, loops continually until a set of 5 numbers with median of about 7 is generated from the my_cauchy function, using some uniformly random mean between 0 and 20.

(c)

Plot a histogram of 100 or more such μ-values.

```r
# run munif 100 times and store results
means = sapply(1:100, function(x){munif()})
hist(means)
```



The histogram shows 100 means which generate sets of 5 numbers from a cauchy distribution with median

7.

Question 3 [15 marks]

The 'Bradley-Terry model' is used to analyse and predict sporting events. Suppose each team in a competition has a certain ability, described by a positive real number ai (for i {1, 2, ..., N}, where N is the number of teams). If team x plays against team y, the probability that the former wins is given by

$$P(teamxwins|ax, ay) = \frac{ax}{ax + ay}$$

which is team x's fraction of the total ability involved in the match. For a tournament with many matches and many teams, let a = {a1, a2, ...aN } be the vector of unknown abilities. The probability of the particular sequence of winners that occurs (the data) is given by a product of terms. Let match i be between team xi and team yi, such that team xi is the winner.

$$P(data|a) = NmatchesYi = 1\frac{a_{x_i}}{a_{x_i} + a_{y_i}}$$

(a)

Write R code to read in the data from matches.csv and make sure the result is a data frame called data.

```r
data = read.csv("matches.csv")
str(data)
```

```
## 'data.frame':    18 obs. of  2 variables:
##  $ team_x: int  1 1 1 4 2 2 3 1 1 1 ...
##  $ team_y: int  3 2 4 1 3 4 4 3 3 2 ...
```

As expected, the data object contains a data.frame with 18 observations of two integer-type variables.

(b)

Write a function called minus_log_likelihood(), that takes a vector a = {a2, ..., aN } as input and returns the negative log likelihood. The ability of team 1 should be assumed to be 1, because only ability ratios matter. If any of the inputs are negative, the function should return Inf.

```r
minus_log_likelihood <- function(abilities) {
  # escape if any abilities are negative
  if (min(abilities) < 0) return(Inf)

  # create a matrix of the team_x vs team_y win counts
  wins = table(data)

  # ability of team 1 assumed to be 1
  abilities = c(1, abilities)

  n = length(abilities)
  lhoods = matrix(0, nrow=n, ncol=n)

  # calculate the log likelihood of each ability
  # given the wins matrix from our data
  for (i in seq_along(abilities)) {
    for (j in seq_along(abilities)) {
      lhoods[i,j] = wins[i,j]*log(abilities[i]) - wins[i,j]*log(abilities[i]+abilities[j])
    }
  }
  -sum(lhoods)
}
minus_log_likelihood(rep(1, 3))
```

```
## [1] 12.47665
```

```
minus_log_likelihood(c(2, 3, 4))
```

```
## [1] 16.60872
```

```
minus_log_likelihood(c(-2, 3, 4))
```

```
## [1] Inf
```

Given some ability ratios of teams 2, 3, and 4, all relative to team 1, this function returns the negative log likelihood of seeing the wins in our data frame. From the first two examples, it can be assumed that an even distribution of ability is more likely than team 4 being four times as "able" as team 1, etc.

(c)

Use R's optim() function to find the maximum likelihood estimate of the {ai} parameters.

```
optimal_a = optim(rep(1, 3), minus_log_likelihood)
# add ability 1 of team 1
abilities = c(1, optimal_a$par)
abilities
```

```
## [1] 1.0000000 0.1768263 0.6959820 0.1987461
```

The maximum likelihood of our team's abilities can be interpreted as a ranking: Team 1 is the "best", followed by Team 3, Team 4, and Team 2.

(d)

Assuming the point estimate from (c) is correct, find the probability that team 3 would beat team 4 in the next match.

```
p_3beats4 = abilities[3] / (abilities[3] + abilities[4])
p_3beats4
```

```
## [1] 0.7778698
```

Team 3 is likely to beat Team 4 in the next match.