# Computational Statistics - Lab 03

*Annalena Erhard (anner218) and Maximilian Pfundstein (maxpf364)*

*2019-01-26*

## Contents



## 1 Question 1: Cluster Sampling

An opinion pool is assumed to be performed in several locations of Sweden by sending interviewers to this location. Of course, it is unreasonable from the financial point of view to visit each city. Instead, a decision was done to use random sampling without replacement with the probabilities proportional to the number of inhabitants of the city to select 20 cities. Explore the file `population.xls`. Note that names in bold are counties, not cities.

**Task:** Import necessary information to R.

| Municipality | Population |
|---|---|
| Botkyrka | 81195 |
| Danderyd | 31150 |
| Ekerö | 25095 |
| Haninge | 76237 |
| Huddinge | 95798 |
| Järfälla | 65295 |

**Task:** Use a uniform random number generator to create a function that selects 1 city from the whole list by the probability scheme offered above (do not use standard sampling functions present in R).

```
# Implementation Max
seed = 12345

get_city_by_urn_wo = function(city_pool) {

  # We take the cumulative sum and then runif from 1 to max(cumulative sum).
  # This way we respect the proportions. As we need every intermediate result,
  # we use a loop
  cumulative_pop_sum = 0

  for (i in 1:nrow(city_pool)) {
    cumulative_pop_sum = cumulative_pop_sum + city_pool$Population[i]
    city_pool$CumSum[i] = cumulative_pop_sum
  }

  # Now we get a random value between 1 to max(cumulative sum). As larger muni-
```

```
  # cipalities have larger ranges, this works as expected
  selection =
    floor(runif(n = 1, min = 1, max = city_pool$CumSum[nrow(city_pool)]))

  # Return the first city which has a greater CumSum than the selection
  return(city_pool[city_pool$CumSum > selection,][1, c(1, 2)])
}
```

**Task:** Use the function you have created in step 2 as follows:

a. Apply it to the list of all cities and select one city
b. Remove this city from the list
c. Apply this function again to the updated list of the cities
d. Remove this city from the list
e. ... and so on until you get exactly 20 cities.

**Answer:** We will combine all of these steps in one function. We're lazy.

```
get_n_cities = function(data, n) {

  # Create a copy to not touch the original data.
  city_pool = data
  selected_cities = data.frame()

  # As long as we don't have n samples, get one and remove it from the pool,
  # as we sample without replacement
  while(nrow(selected_cities) < n) {
    selected_city = get_city_by_urn_wo(city_pool)
    selected_cities = rbind(selected_cities, selected_city)
    city_pool = city_pool[!rownames(city_pool) %in% rownames(selected_cities),]
  }

  return(selected_cities)
}

sample = get_n_cities(data, 20)
```

**Task:** Run the program. Which cities were selected? What can you say about the size of the selected cities?

**Answer:**

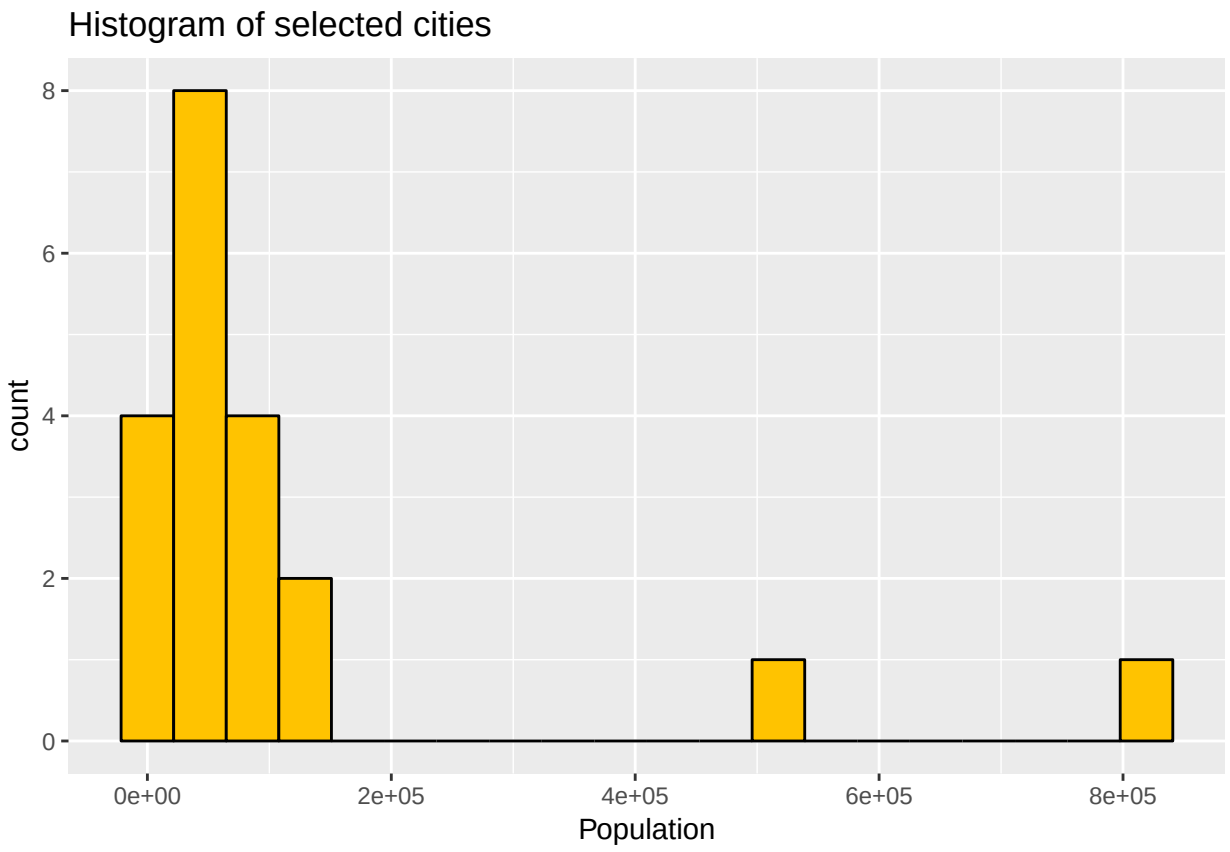
The following cities were selected:

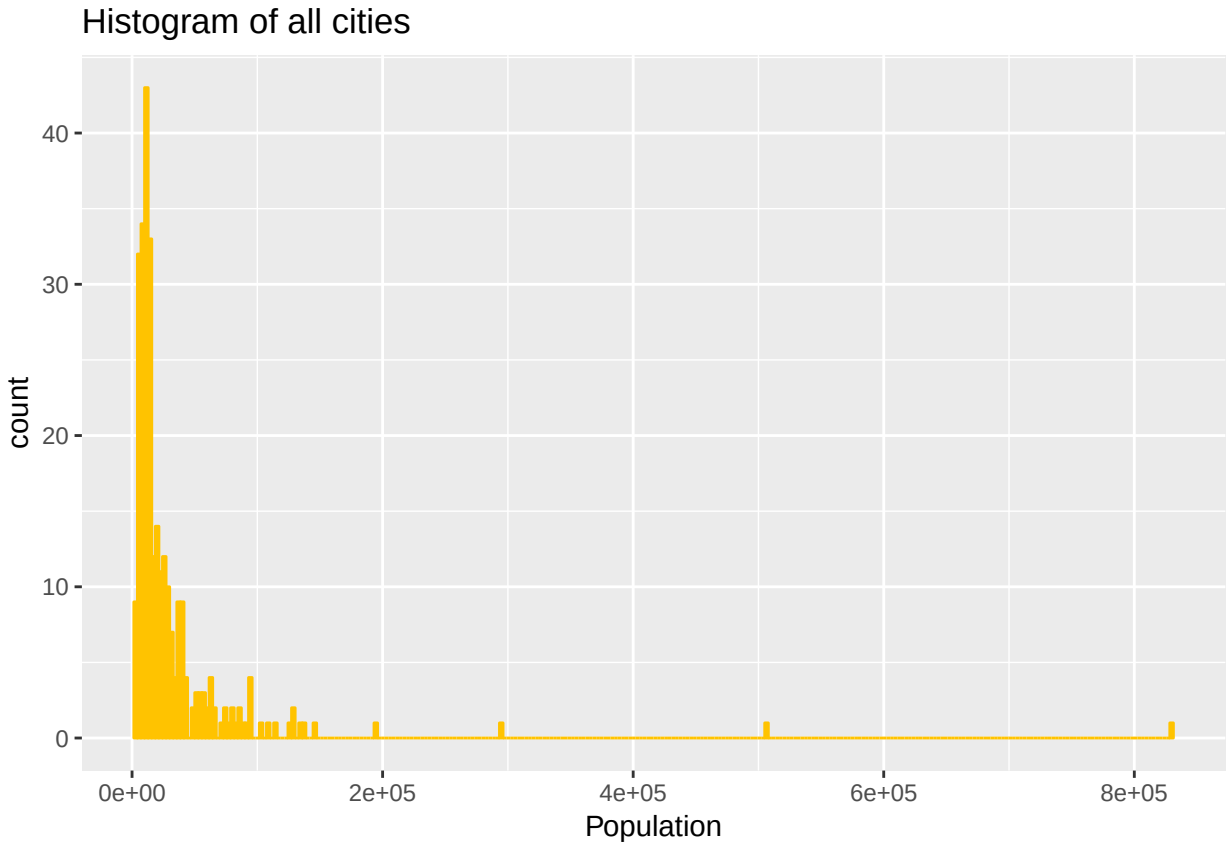
```
##      Municipality Population
## 47      Linköping     144690
## 124         Ystad      28109
## 214 Hallstahammar      15127
## 16      Stockholm     829417
## 15          Solna      66909
## 101   Helsingborg     128359
## 285         Luleå      73950
## 90        Gotland      57221
## 246     Söderhamn      25759
## 1        Botkyrka      81195
## 208    Lindesberg      23029
## 146      Göteborg     507330
## 238         Gävle      94352
```

```
## 13        Sigtuna      39219
## 7         Lidingö      43445
## 103       Höganäs      24480
## 261     Östersund      59136
## 99          Båstad      14269
## 84     Mörbylånga      13834
## 252          Ånge      10148
```

It can be seen, that mostly cities with a population greater than the mean (`32009.25`) were drawn.

**Task :** Plot one histogram showing the size of all cities of the country. Plot another histogram showing the size of the 20 selected cities. Conclusions?






**Answer:** We see that both histograms have a similar shape. This means that our sampling function is taking the size of the city into account and we created a similar distribution with a smaller subset. That is exaclty what we needed to perform the opinion pool.

## 2 Question 2: Different Distributions

The double exponential (Laplace) distribution is given by formula

$$DE(\mu, \alpha) = \frac{\alpha}{2} e^{-\alpha|x-\mu|}$$

**Task**: Write a code generating double exponential distribution $DE(0,1)$ from $Unif(0,1)$ by using the inverse CDF method. Explain how you obtained that code step by step. Generate `10000` random numbers from this distribution, plot the histogram and comment whether the result looks reasonable.

**Answer:** What we are going to do is we sample from `unif(0,1)` and take the results and put it into the quantile function of $DE(0,1)$. As we take the definition from Laplace Distribution (Wikipedia) for the quantile function we have to keep in mind that $\alpha = \frac{1}{b}$. Let's first define our quantile function with $\mu = 0$, $\alpha = 1$ and thus $b = \frac{1}{1} = 1$ as well:

$$Q(p) = F^{-1}(p) = \mu - b * sgn(p - 0.5) * ln(1 - 2|p - 0.5|)$$

With the above defined variables we obtain:

$$Q(p|\mu, b) = Q(p|0,1) = -sgn(p - 0.5) * ln(1 - 2|p - 0.5|)$$

Where `sgn` is the Sign Function (Wikipedia).

Let's implement those two functions as a start:

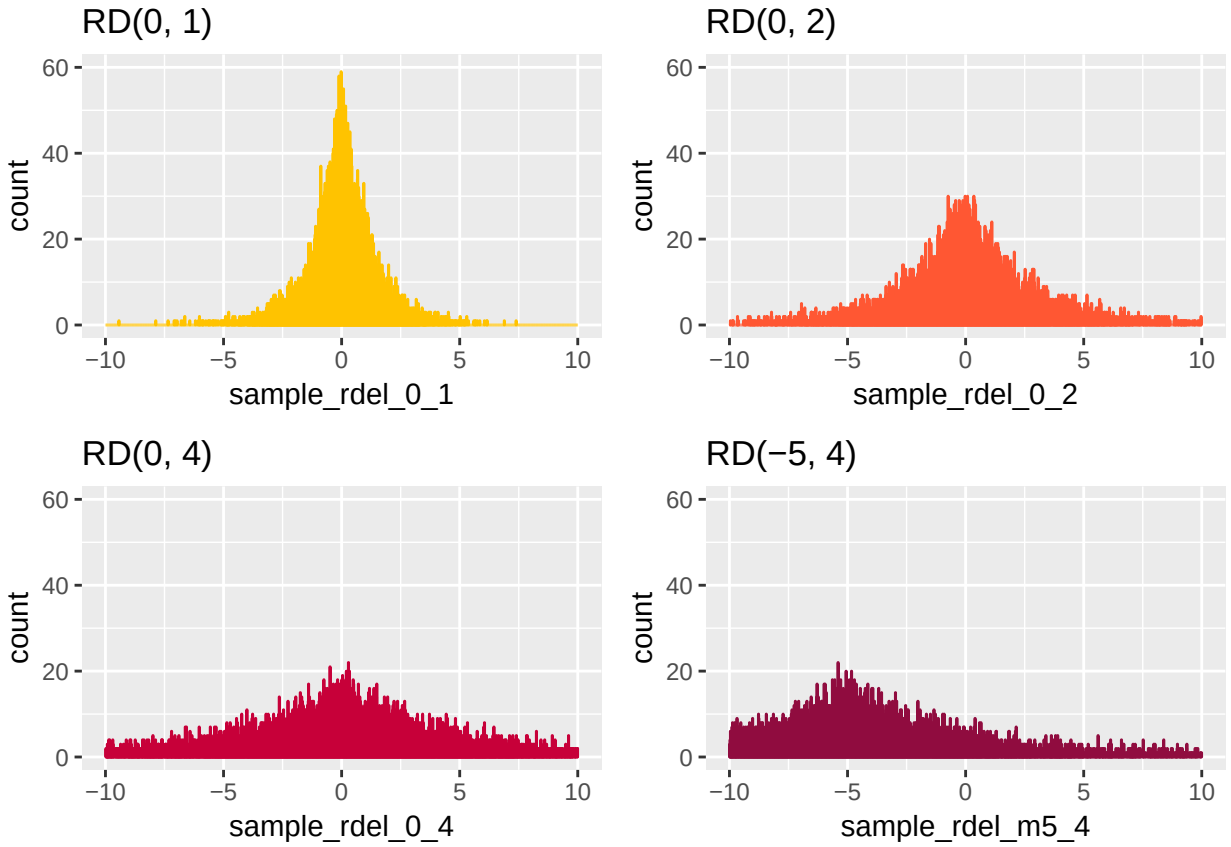
```
sgn = function(x) {
  if (x < 0) return(-1)
  if (x > 0) return( 1)
  return(0)
}

q_double_exponential_d = function(p, mu = 0, b = 1) {
  if (p < 0 | p > 1) stop("p must be in range (0, 1)")
  return(mu - b * sgn(p-0.5) * log(1 - 2 * abs(p - 0.5)))
}
```

Next we implement a function for drawing n times.

```
rdel = function(n = 1, mu = 0, b = 1) {
  quantiles = replicate(n, runif(n = 1, min = 0, max = 1))
  rdels = sapply(X = quantiles, FUN = q_double_exponential_d, mu = mu, b = b)
  return(rdels)
}
```

Let's look how the plot for `10000` random numbers from this distribution looks like:




The results look reasonable when compared to the original density function as the shape is what we would expect.

**Task**: Use the Acceptance/rejection method with $DE(0, 1)$ as a majorizing density to generate $\mathcal{N}(0, 1)$

variables. Explain step by step how this was done. How did you choose constant `c` in this method? Generate `2000` random numbers $\mathcal{N}(0,1)$ using your code and plot the histogram. Compute the average rejection rate `R` in the acceptance/rejection procedure. What is the expected rejection rate `ER` and how close is it to `R`? Generate `2000` numbers from $\mathcal{N}(0,1)$ using standard `rnorm()` procedure, plot the histogram and compare the obtained two histograms.

# 3 Source Code

```
knitr::opts_chunk$set(echo = TRUE, cache = FALSE, include = TRUE, eval = TRUE)
library(knitr)
library(readxl)
library(ggplot2)
library(gridExtra)

# reading the data in ------------------------------------
data = read.csv("population.csv", sep = ";", encoding = "latin1")

kable(head(data))


# Implementation Max
seed = 12345

get_city_by_urn_wo = function(city_pool) {

  # We take the cumulative sum and then runif from 1 to max(cumulative sum).
  # This way we respect the proportions. As we need every intermediate result,
  # we use a loop
  cumulative_pop_sum = 0

  for (i in 1:nrow(city_pool)) {
    cumulative_pop_sum = cumulative_pop_sum + city_pool$Population[i]
    city_pool$CumSum[i] = cumulative_pop_sum
  }

  # Now we get a random value between 1 to max(cumulative sum). As larger muni-
  # cipalities have larger ranges, this works as expected
  selection =
    floor(runif(n = 1, min = 1, max = city_pool$CumSum[nrow(city_pool)]))

  # Return the first city which has a greater CumSum than the selection
  return(city_pool[city_pool$CumSum > selection,][1, c(1, 2)])
}


# Comments:
# - Why n if the function is supposed to select one?
# - This does not take into account that cities with a bigger population are more
#   likely to be selected
# - round() somestimes rounds downwards, sometimes upwards, for correct partitioning
#   floor() or ceil() is better
# - Setting the seed for each drawing will result in the same number every time
```

```
#   if the external seed supplier does not change it!

sample_wo_replacement = function(n, data, seed){
  samples = numeric()
  i = 1
  while (length(samples) < n) {
    set.seed(seed)
     a = round(runif(n = 1, min = 1, max = nrow(data)))
     while (a %in% samples) {
       set.seed(seed)
       a = round(runif(n = 1, min = 1, max = nrow(data)))
     }
     samples[i] = a
     i = i+1
  }
  return(samples)
}



get_n_cities = function(data, n) {

  # Create a copy to not touch the original data.
  city_pool = data
  selected_cities = data.frame()

  # As long as we don't have n samples, get one and remove it from the pool,
  # as we sample without replacement
  while(nrow(selected_cities) < n) {
    selected_city = get_city_by_urn_wo(city_pool)
    selected_cities = rbind(selected_cities, selected_city)
    city_pool = city_pool[!rownames(city_pool) %in% rownames(selected_cities),]
  }

  return(selected_cities)
}

sample = get_n_cities(data, 20)


city_pool = data
selected_cities = data.frame()

selected_city_id = sample_wo_replacement(1, all_cities, 12345)


selected_cities = rbind(city_pool[selected_city_id,])
city_pool = all_cities[-selected_city,]


while (length(selected_cities) < 21) {
  selected_city_id = sample_wo_replacement(1, all_cities, 12345)
  all_cities = all_cities[-selected_city,]
```

```
  selected_cities = c(selected_cities, selected_city)
}

print(selected_cities)


sample


ggplot(sample)+
  geom_histogram(aes(x = Population), bins = nrow(sample), color = "black", fill = "#FFC300") +
  ggtitle("Histogram of selected cities")

ggplot(data)+
  geom_histogram(aes(x = Population), bins = nrow(data), color = "#FFC300", fill = "#FFC300") +
  ggtitle("Histogram of all cities")


sgn = function(x) {
  if (x < 0) return(-1)
  if (x > 0) return( 1)
  return(0)
}

q_double_exponential_d = function(p, mu = 0, b = 1) {
  if (p < 0 | p > 1) stop("p must be in range (0, 1)")
  return(mu - b * sgn(p-0.5) * log(1 - 2 * abs(p - 0.5)))
}


rdel = function(n = 1, mu = 0, b = 1) {
  quantiles = replicate(n, runif(n = 1, min = 0, max = 1))
  rdels = sapply(X = quantiles, FUN = q_double_exponential_d, mu = mu, b = b)
  return(rdels)
}


sample_rdel_0_1 = rdel(10000, mu = 0, b = 1)
sample_rdel_0_2 = rdel(10000, mu = 0, b = 2)
sample_rdel_0_4 = rdel(10000, mu = 0, b = 4)
sample_rdel_m5_4 = rdel(10000, mu = -5, b = 4)

df = data.frame(sample_rdel_0_1, sample_rdel_0_2,
                sample_rdel_0_4, sample_rdel_m5_4)


p1 = ggplot(df) +
  geom_histogram(aes(x = sample_rdel_0_1),
                 color = "#FFC300", fill = "#FFC300", binwidth = 0.01) +
  xlim(-10, 10) +
  ylim(0, 60) +
  ggtitle("RD(0, 1)")
```

```r
p2 = ggplot(df) +
  geom_histogram(aes(x = sample_rdel_0_2),
                 color = "#FF5733", fill = "#FF5733", binwidth = 0.01) +
  xlim(-10, 10) +
  ylim(0, 60) +
  ggtitle("RD(0, 2)")

p3 = ggplot(df) +
  geom_histogram(aes(x = sample_rdel_0_4),
                 color = "#C70039", fill = "#C70039", binwidth = 0.01) +
  xlim(-10, 10) +
  ylim(0, 60) +
  ggtitle("RD(0, 4)")

p4 = ggplot(df) +
  geom_histogram(aes(x = sample_rdel_m5_4),
                 color = "#900C3F", fill = "#900C3F", binwidth = 0.01) +
  xlim(-10, 10) +
  ylim(0, 60) +
  ggtitle("RD(-5, 4)")

grid.arrange(p1, p2, p3, p4, nrow = 2)
```