732A90: Computational Statistics

Computer lab3 - Group11

Sofie Jörgensen, Oriol Garrobé Guilera, David Hrabovszki

13 February 2020

Question 1: Cluster sampling

An opinion poll is assumed to be performed in several locations of Sweden by sending interviewers to this location. But since it is unreasonable from the financial point of view to visit each city, we use random sampling without replacement to select 20 cities, where the the probability of a city getting selected is proportional to its number of inhabitants.

1.

The data containing the cities and populations is found in `population.csv`, which we import into R.

2.

We create a function `sampler()` that selects 1 city from the whole list using a uniform random number generator. This function essentially performs weighted random sampling on our data, where the weight corresponding to each city is its population.

Our implementation is based on the pseudo-code written by Peter Kelly [1]. The function first generates an integer random number from the uniform distribution between the range [1,Total Population]. Then, it iterates through all the cities and updates the random number by subtracting the population of the current city from it. After that, but in the same iteration step, it compares this value to 0. If it is smaller than or equal to 0, then the function returns the current city and the loop breaks. If it is larger, then it continues into the next iteration step and updates the value with the subtraction of the next city's population from it until it finds a city, where this value is <= 0.

Since the generated random number gets lower and lower with every iteration, it will eventually return a result in every case. More populated cities get selected more often, because the larger the weight, the more likely to be larger than a randomly generated number in the range [1,Total Population], which means that their difference is <= 0.

3.

In this step, we use our `sampler()` function to randomly select 20 cities from the list without replacement. The function ensures that more populated cities get selected with a higher probability.

The method is as follows:

(a) Apply `sampler()` to the list of all cities and select one city

(b) Remove this city from the list

(c) Apply `sampler()` again to the updated list of the cities

(d) Remove this city from the list

(e) ... and so on until we get exactly 20 cities.

[1] https://medium.com/@peterkellyonline/weighted-random-selection-3ff222917eb6

4.

Using `sampler()` function and the sampling method described in step 3, we obtain a list (Table 1) of selected cities.

Table 1: Selected cities

	Municipality	Population
5	Huddinge	95798
8	Nacka	88085
16	Stockholm	829417
19	Tyresö	42602
51	Söderköping	14042
80	Hultsfred	13855
107	Kristianstad	78788
112	Malmö	293909
154	Lidköping	37989
155	Lilla Edet	12773
173	Tjörn	14961
177	Uddevalla	51518
178	Ulricehamn	22753
221	Västerås	135936
230	Mora	20146
247	Härnösand	24675
249	Sollefteå	20442
266	Nordmaling	7205
278	Arvidsjaur	6622
285	Luleå	73950

The average population of the 20 selected cities is 94273, while the average of all cities is 32209. This indicates that our random selection contains more populated cities with a higher probability.

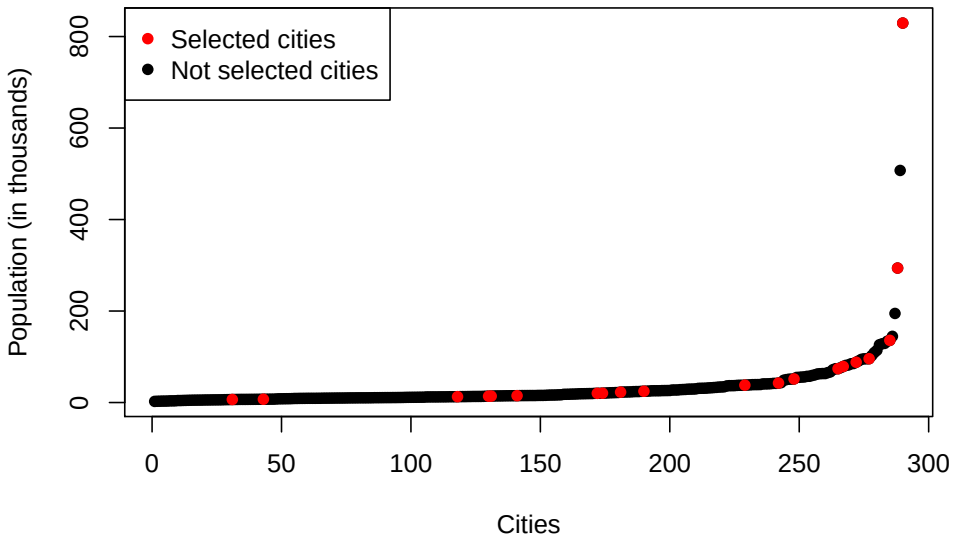

Figure 1: Weighted random selection of cities in Sweden

Figure 1 visualizes the cities as points in ascending order by their populations. The red points represent cities that were selected by our sampling method, the black ones represent the ones that were not. We can observe that more populated cities make up the majority of our selection, as they were more likely to be selected. In fact, out of the 20 most populated cities, we picked 5, while we chose 0 out of the 20 least populated ones.

5.

Figure 2 shows the histogram of the size of all Swedish cities, while figure 3 is the histogram of the size of the 20 selected cities. We can observe that both histograms look similar, this is due to the randomness of the sampling method. We can also see that cities with large population are very rare in the first case, but much more frequent in the second one. This is because our `sample()` function selects cities with large populations with a higher probability. Cities with smaller populations are still in majority, since there were a lot more of them originally, but their proportion is much smaller than before sampling.

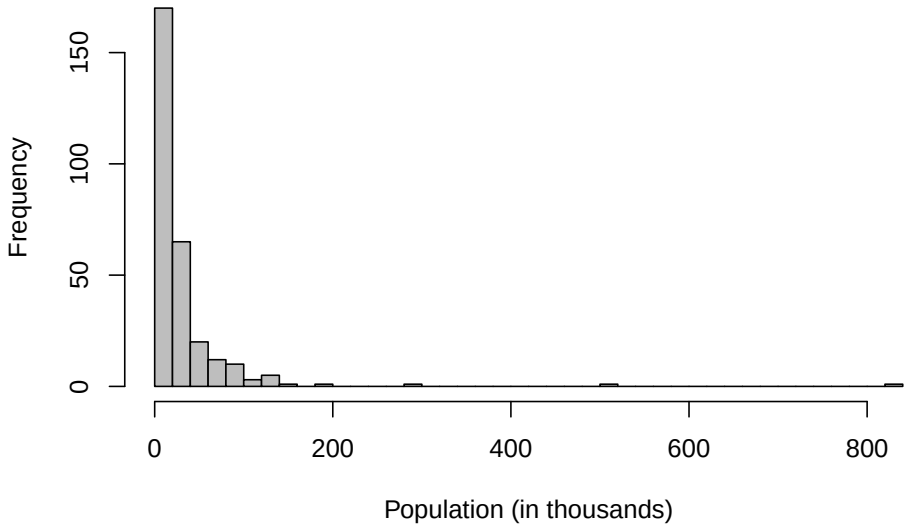

Figure 2: Histogram of the sizes of cities in Sweden

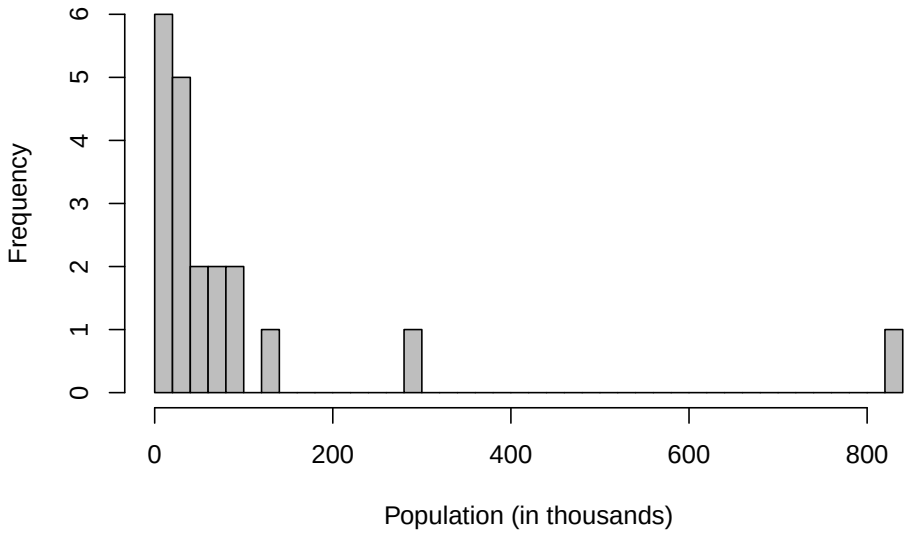

Figure 3: Histogram of the sizes of the selected cities

Question 2: Different distributions

In this question we are given the double exponential (Laplace) distribution with the location parameter μ and the scale parameter α, where the formula is given by

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

1.

The idea behind the inverse cumulative distribution function (CDF) method is that if we can generate $U \sim \text{Unif}(0, 1)$, then we can generate $X \sim F_X$ as $X = F_X^{-1}(U)$ provided we can calculate $F_X^{-1}(U)$.

The aim of this task is to generate random numbers from a double exponential or Laplace distribution with the location parameter $\mu = 0$ and the scale paramenter $\alpha = 1$, that is $DE(0, 1)$, from a uniform distribution Unif$(0, 1)$ by using the inverse CDF method. For this, we want to generate 10000 random numbers, i.e. $X \sim DE(0, 1)$, and plot a histogram of the results.

The probability density function (PDF) of a standard double exponential $DE(0, 1)$ is given by

$$f_X(x) = \frac{1}{2}\exp(-|x|).$$

Implying that the CDF is,

$$F_X(x) = \int_{-\infty}^{x} f_X(s)\ ds = \int_{-\infty}^{x} \frac{1}{2}\exp(-|s|)\ ds.$$

Since an absolute value is included in the PDF, the value of x can be positive as well as negative. Thus we divide this problem into two cases $x < 0$ and $x \geq 0$, and we compute the following integrals:

For $x < 0$, the integral is computed as

$$F_X(x) = \int_{-\infty}^{x} \frac{1}{2}\exp(s)\ ds = \frac{\exp(x)}{2}.$$

For $x \geq 0$,

$$F_X(x) = \int_{-\infty}^{0} \frac{1}{2}\exp(s)\ ds + \int_{0}^{x} \frac{1}{2}\exp(-s)\ ds = 1 - \frac{\exp(-x)}{2}.$$

Therefore,

$$F_X(x) = \begin{cases} \frac{\exp(x)}{2} & \text{if } x < 0, \\ 1 - \frac{\exp(-x)}{2} & \text{if } x \geq 0. \end{cases}$$

Now we can obtain F_X^{-1} by taking the inverse of the CDF, by using the following steps:

For $x < 0$,

$$y = \frac{\exp(x)}{2} \iff \exp(x) = 2y \iff x = \ln(2y),$$

and for $x \geq 0$,

$$y = 1 - \frac{\exp(-x)}{2} \iff \exp(-x) = 2 - 2y \iff x = -\ln(2 - 2y).$$

Hence, if $U \sim \text{Unif}(0,1)$, then for $x < 0$, we have that $\ln(2U) = X \sim DE(0,1)$, and for $x \geq 0$, we have $-\ln(2-2U) = X \sim DE(0,1)$.

Studying this results it is clear that, for negative values $x < 0$, $\ln(2u) < 0 \iff u < \frac{1}{2}$. Also, for $x \geq 0$, $-\ln(2-2u) \geq 0 \iff u \geq \frac{1}{2}$.

With all this information it is possible to code a generator for the double exponential distribution $DE(0,1)$ from $U(0,1)$ by using the inverse CDF method. We create the function `my_DE` accoring to the approach as described (see Appendix).

From this point, using `my_DE` we can generate 10000 random numbers from this distribution and plot the histogram, figure 4.

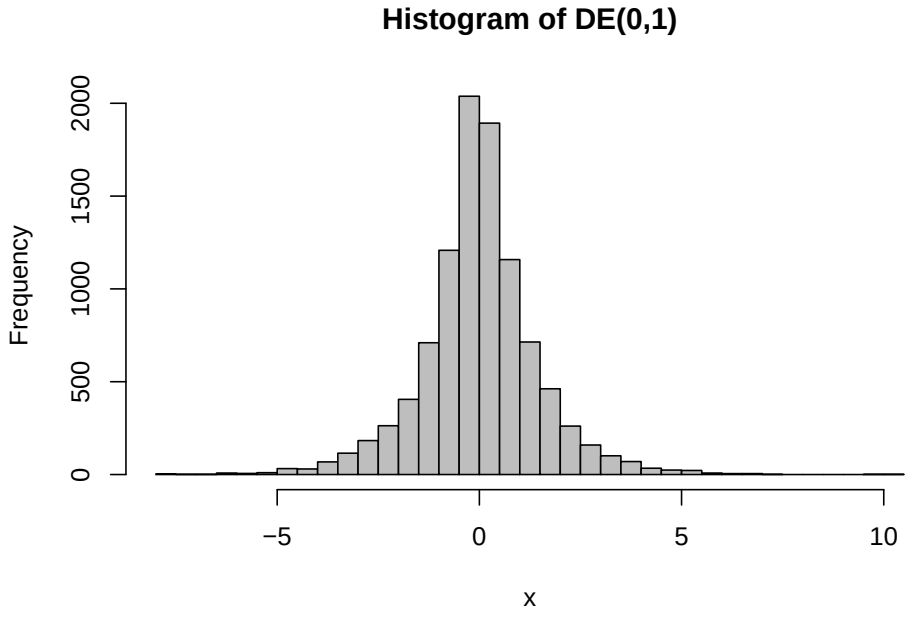

Figure 4: Histogram of 10000 generated random numbers from DE(0,1) using the inverse cdf method.

Looking at figure 4 we can see that it looks like a Laplace distribution centered around 0. Therefore the results obtained using the generator function `my_DE` look reasonable.

We can also use the function `rlaplace` from the package `rmutil` that generates random numbers from a standard laplace distribution and plot an histogram, figure 5, in order to compare to the previous plot.

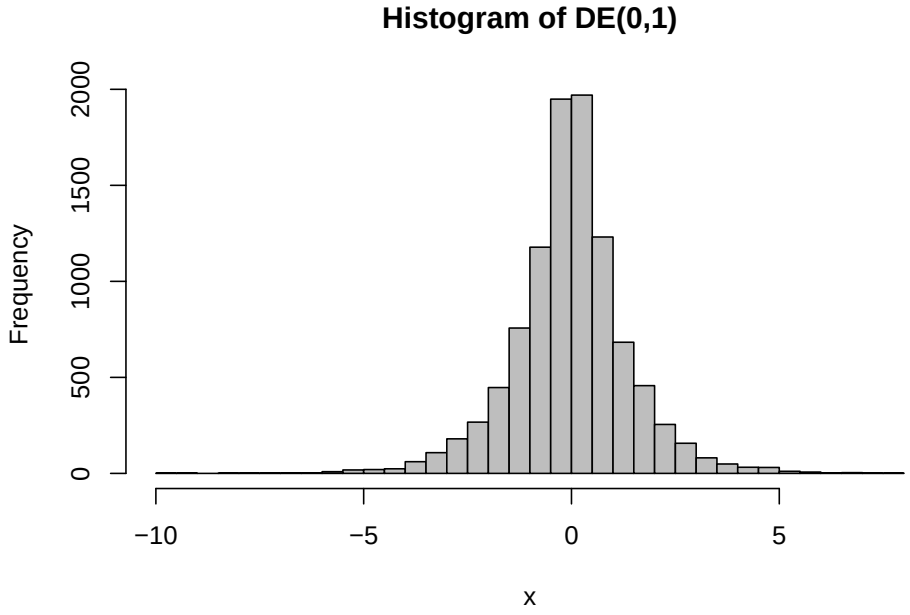

Figure 5: Histogram of 10000 generated random numbers from DE(0,1) using rlaplace function

As we can see both histograms look alike, confirming the previous statements and giving reliability to the results.

2.

Now we are going to apply the Acceptance/Rejection method by using the double exponential distribution obtained from Question 2.1, in order to generate standard normal variables. In other words, $DE(0,1)$ is the majorizing density f_Y and $N(0,1)$ is the target density f_X. The implementation of the Acceptance/Rejection method is taken from the example in the course material provided in the lecture. From Figure 4 we can observe that the shape of the distribution looks quite similar to a standard normal distribution since it is centered around zero and have a similar spread, thus the choice of the $DE(0,1)$ can be considered as reasonable.

The idea is to generate a random number from $DE(0,1)$, i.e. $Y \sim f_Y$ and a random number from $U \sim \text{Unif}(0,1)$. Then if $U \leq \frac{f_X(Y)}{c f_Y(Y)}$ holds, then we accept the generated number Y, otherwise it is rejected. The rejection is controlled by the majorizing constant c, and a value of c closer to 1 will imply fewer rejections. This means that if $c = 1$, then f_Y and f_X are the same density function. Thus we wish to pick a majorizing constant c, close to 1 such that it fulfills the requirement that $c \cdot f_Y(x) \geq f_X(x)$ for all x. We test a sequence of numbers of c from 1 to 5, and obtained 1.32 as the smallest c such that the requirement is still fulfilled for any x.

Thereafter we generate 2000 standard normal random numbers with this setting. The result is presented in a histogram in Figure 7. In the Acceptance/Rejection method, we compute the number of rejections R with `num.reject` and we obtain $R = 0.317$ for $c = 1.32$. The number of rejections plus the single draw, when the random number is accepted, must equal the total number of draws. Since the total number of draws is Geometric distributed with mean c, the expected rejection rate ER is given by $c - 1 = 1.32 - 1 = 0.32$. By computing the difference $ER - R = 0.003$ we can conclude that the expected rejection rate ER and the average rejection rate R are very close to each other. Then we generate 2000 standard normal random numbers, but directly from the standard normal distribution by using `rnorm`. Also, the two histograms looks similar when comparing Figure 7 with Figure **??** for 2000 random numbers. Hence we are satisfied with our implemented random generator.

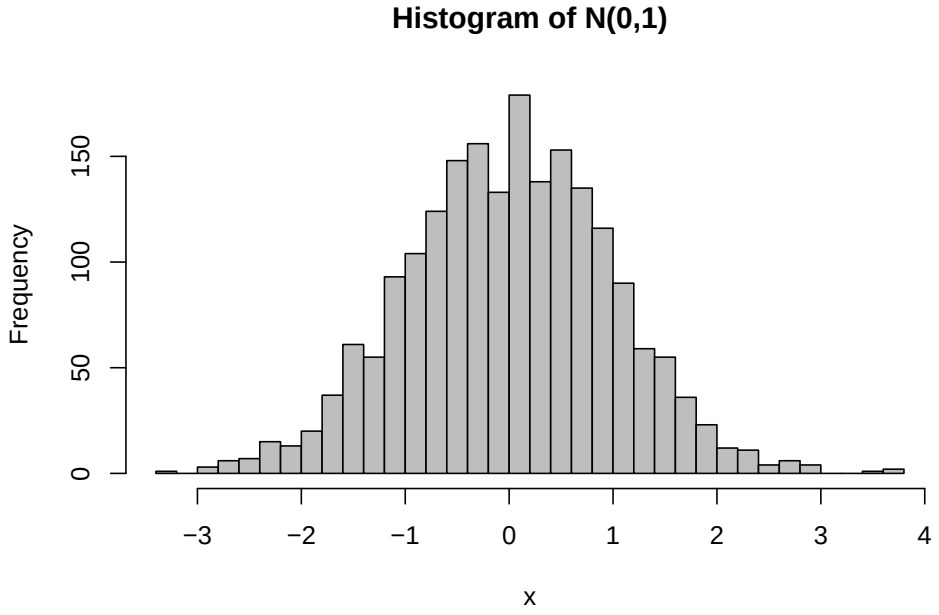

Figure 6: Histogram of 2000 generated random numbers from N(0,1) using the Acceptance/Rejection method

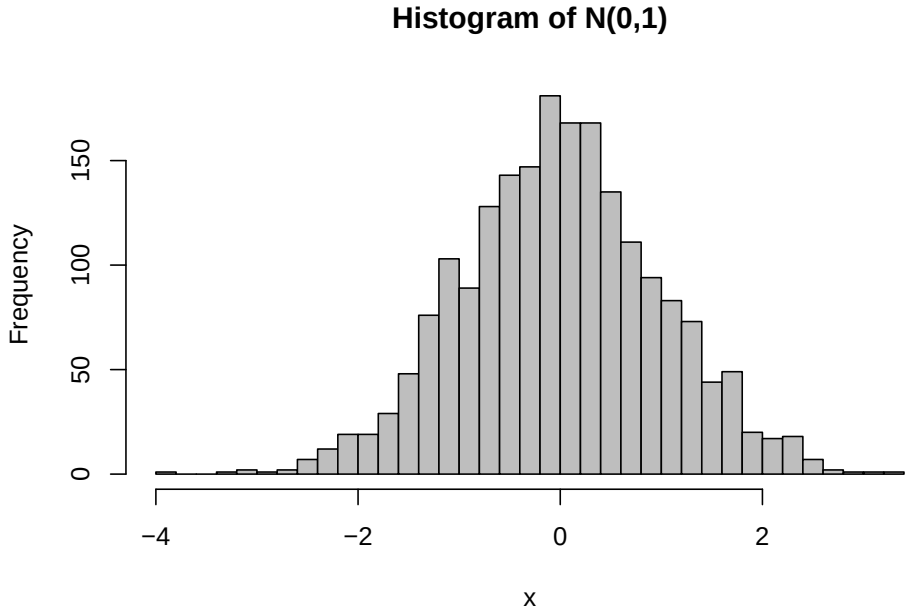

Figure 7: Histogram of 2000 generated random numbers from N(0,1) by using rnorm().

Appendix

```
knitr::opts_chunk$set(echo = FALSE, fig.pos = 'H')
################################### Question 1: Cluster sampling ###################################
# R version
RNGversion('3.5.1')
#1.1
data <- read.csv2("population.csv", stringsAsFactors = FALSE)

#1.2
# Seed
set.seed(1234567890)

sampler <- function(data) {
  rand <- round(runif(1,min=1, max=sum(data$Population)),0)
  for (i in 1:dim(data)[1]) {
    rand <- rand - data[i,2]
    if (rand <= 0) {
      return(data[i,1])
    }
  }
}

#1.3
selected_cities <- vector(length = 20, mode = "character")
cities <- data

set.seed(1234567890)
for (j in 1:20) {
  selected_cities[j] <- sampler(cities)
  cities <- cities[-which(cities$Municipality==selected_cities[j]),]
}

#1.4
#the selected cities:
sample <- data[which(data$Municipality %in% selected_cities),]
knitr::kable(sample, caption = 'Selected cities')

#mean(sample$Population) #94273.3
#mean(data$Population) #32209.25

#extra
joined <- merge(data,sample, by = "Municipality", all = TRUE)

plot(joined[order(joined$Population.x),]$Population.x/1000,
     pch = 16, ylab = "Population (in thousands)", xlab = "Cities")
points(joined[order(joined$Population.x),]$Population.y/1000,
       col="red", pch = 16)
legend(x="topleft", legend = c("Selected cities", "Not selected cities"),
       col = c("red", "black"), pch = 16)
hist(data$Population/1000, breaks = 50,
     ylab = "Frequency", xlab = "Population (in thousands)", main = "", col="grey")
hist(sample$Population/1000, breaks = 50,
```

```
     ylab = "Frequency", xlab = "Population (in thousands)", main = "", col="grey")
############################## Question 2: Different distributions ##############################

# 2.1
# Random numbers from DE(0,1) from Unif(0,1) generated by using the inverse cdf method.
# Divide the two cases into different statements.
my_DE <- function(n){
  x <- c()
  for (i in 1:n){
    u <- runif(1)
    if(u < 0.5){
      x[i] <- log(2*u)
    }else{
      x[i] <- -log(2*(1-u))
    }
  }
  return(x)
}

# Generate 10000 random numbers from DE(0,1)
x <- my_DE(10000)

# Histogram of the random numbers
hist(x, breaks = 40, col = "grey", main = "Histogram of DE(0,1)")
library(rmutil)
# Histogram of  DE(0,1) by directly sample from the laplace.
hist(rlaplace(10000), breaks = 40, col = "grey", main = "Histogram of DE(0,1)", xlab = "x")

# 2.2
# Acceptance/Rejection method
# Function that generates N(0,1) by using the majorizing density DE(0,1)
# The code is obtained from "732A90_ComputationalStatisticsVT2020_Lecture03codeSlide19.R"
fgennorm <- function(c){
  x <- NA
  num.reject <- 0
  while (is.na(x)){
    y <- my_DE(1) # majorizing density
    u <- runif(1) # Unif(0,1)
    if (u <= dnorm(y)/(c*0.5*exp(-abs(y)))){x <- y}
    else{ num.reject <- num.reject + 1}
  }
  c(x, num.reject)
}

# Set seed
set.seed(12345)

# Number of generated numbers
n <- 2000
# Create a matrix that stores the random numbers and number of rejections
A <- matrix(c(rep(0,n) ,rep(0,n)), ncol = 2)

# Call the fgennorm function with c=3.12, based on the requirement.
```

```r
for (i in 1:n){
  A[i,] <- fgennorm(1.32)
}

# Histogram of 2000 generated random numbers N(0,1)
# by using the Acceptance/Rejection method
# with majorizing density DE(0,1)
hist(A[,1], breaks = 40, col = "gray", xlab = "x", main = "Histogram of N(0,1)")

# Histogram of 2000 random N(0,1) numbers
hist(rnorm(2000), breaks = 40, col = "grey", xlab = "x" , main ="Histogram of N(0,1)")

# Average rejection rate for c=1.32
mean(A[,2])

# Pick the smallest c, s.t. the requirement is still fulfilled
# Here the generated data with c=1.32 is used, but works the same for other values
nr <- A[,1]
c_values <- seq(1,5,0.01)
for (i in seq_along(c_values)){
  if(all((c_values[i]*0.5*exp(-abs(nr)) >= dnorm(nr)))){
    c_value <- c_values[i]
    break()
  }
}

# Confirmes majorizing constant
all(c_value*0.5*exp(-abs(nr)) >= dnorm(nr))
table(c_value*0.5*exp(-abs(nr)) >= dnorm(nr))
```