Computational statistics - Lab 3

Alexander Karlsson (aleka769)
Lennart Schilling(lensc874)

2019-02-12

Contents

Question 1: Cluster sampling

Importing data

```
# importing data
data = read.csv("population.csv", sep = ";", stringsAsFactors = FALSE)
```

Creating function

Since the random sampling of the 20 cities should be performed using the probabilities proportional to the number of inhabitants of the city, these probabilite have to be calculated first. They will be added to the data frame.

To select one city from the whole list with the derived probabilities using a uniform random number generator, the cumulated percentages have to be calculated.

A uniformly sampled value (parameters have to be specified for the function) between 0 and 1 then decides for which interval of cumulated percentages the city will be chosen. If, for example, two cities represent 5% and 4% of the total population, the cumulated percentages are 5% and 9%. If now a value of for example 0.07 would be uniformely sampled, this number would be bigger then 0.05 and smaller than 0.09. Therefore, the second city would be chosen. Since larger cities are connected to larger intervals, the cities are chosen based on the probabilities proportional to the number of inhabitants. If this city was chosen, it will be removed from the list. Consequently, the probabilites proportional to the number of inhabitants of the cities and therefore the cumulated percentages have to be calculated again.

The function can be created as follows:

```
selectCities = function(n, data, rngPars) {
  # generating n uniform random numbers between 0 and 1 using rngPars
  rNumber = c()
  x = rngPars["seed"]
  for (i in 1:n) {
    x = (rngPars["a"] * x + rngPars["c"]) %% rngPars["m"]
    rNumber[i] = x / rngPars["m"]
  }
  # creating empty objects for the follwing loop
  selectedCities = c()
  loopData = data
  # running function n times
  for (i in 1:n) {
    # preparing loopData
      # adding percentage of inhabitants of cities to loopData
      loopData$Percentage = loopData$Population/sum(loopData$Population)
      # adding cumulated percentage of inhabitants of cities to data frame
      loopData$CumPercentage = cumsum(loopData$Percentage)
    # adding city from loopData based on percentage of inhabitants using random number
    selectedCities[i] =
      loopData[
        which(
          loopData$CumPercentage ==
            min(loopData[loopData$CumPercentage >= rNumber[i], ]$CumPercentage)
          ), ]$Municipality
    # removing selected city from loopData
    loopData = loopData[loopData$Municipality != selectedCities[i], ]
```

```
  }
  # returning selected cities
  return(selectedCities)
}
```

Running function

```
# defining parameters to generate unif[0,1]
rngPars = c(seed = 5, a = 2, c = 2, m = 100)
# running function
selectedCities = selectCities(n = 20,
                              data = data,
                              rngPars = rngPars)
```

Analysing result

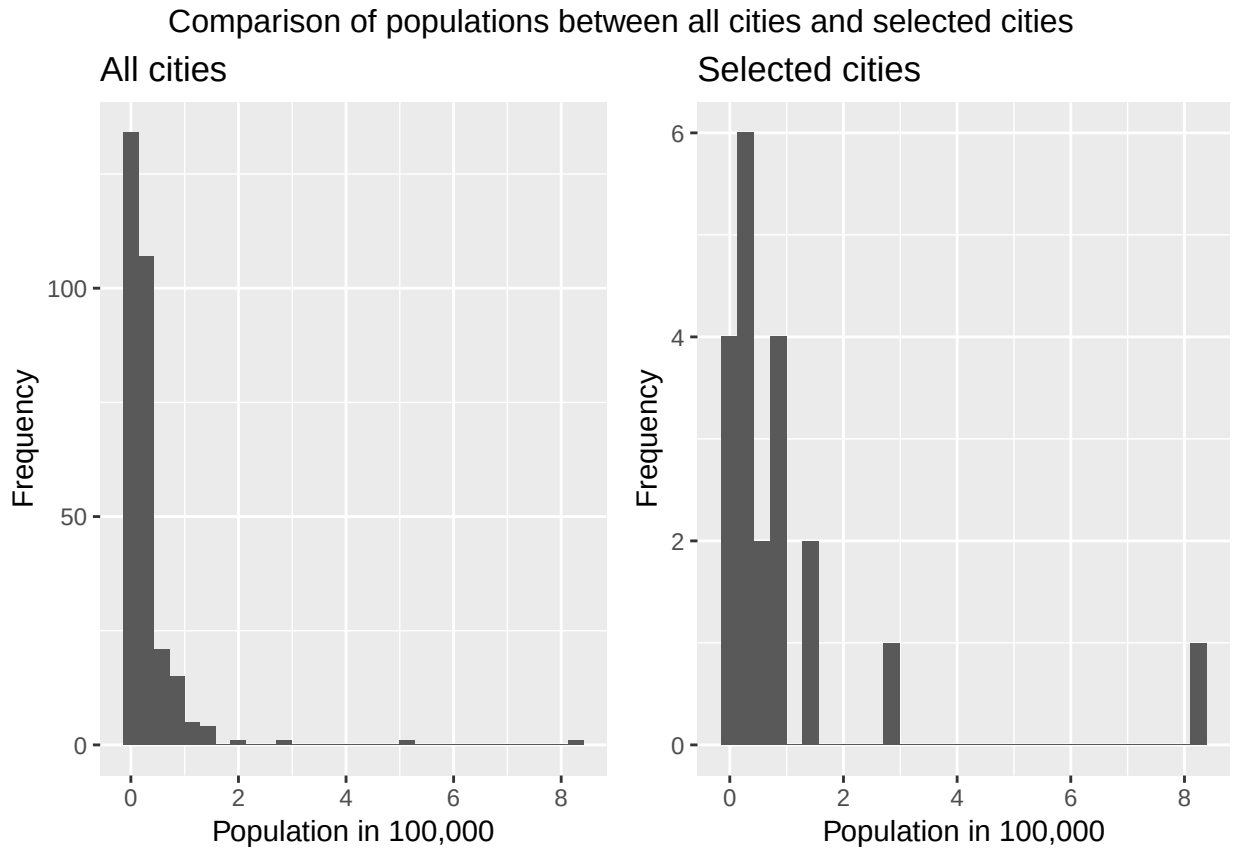
```
# listing selected cities
selectedCities = data[data$Municipality %in% selectedCities, ]
knitr::kable(selectedCities, caption = "Selected cities", row.names = FALSE)
```

Table 1: Selected cities

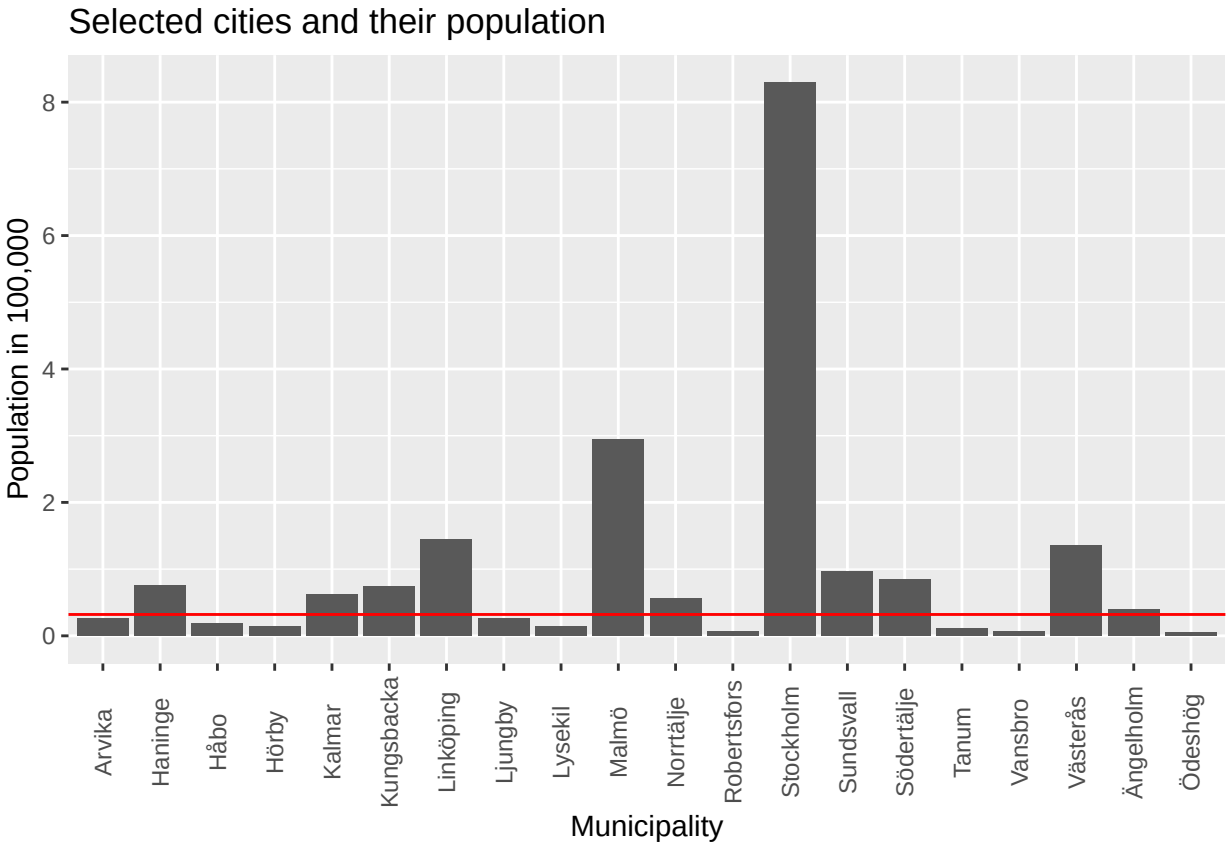
Municipality	Population
Haninge	76237
Norrtälje	55927
Stockholm	829417
Södertälje	85270
Håbo	19452
Linköping	144690
Ödeshög	5314
Ljungby	27410
Kalmar	62388
Hörby	14762
Malmö	293909
Ängelholm	39083
Kungsbacka	73938
Lysekil	14535
Tanum	12253
Arvika	26100
Västerås	135936
Vansbro	6876
Sundsvall	95533
Robertsfors	6880

```
# plotting histograms
library(ggplot2)
library(gridExtra)
plotAllCities = ggplot() +
  geom_histogram(data = data,
```

```
                aes(x = round(Population/100000, 2))) +
  ggtitle("All cities") +
  labs(x = "Population in 100,000",
       y = "Frequency")
plotSelectedCities = ggplot() +
  geom_histogram(data = selectedCities,
                 aes(x = round(Population/100000, 2))) +
  ggtitle("Selected cities") +
  labs(x = "Population in 100,000",
       y = "Frequency")
grid.arrange(plotAllCities,
             plotSelectedCities,
             ncol = 2,
             top = "Comparison of populations between all cities and selected cities")
```




```
# plotting extra plot for selected cities
ggplot() +
  geom_col(data = selectedCities,
           aes(x = Municipality,
               y = round(Population/100000, 2))) +
  geom_abline(intercept = round(mean(data$Population/100000),2),
              slope = 0,
              colour = "red") +
  ggtitle("Selected cities and their population") +
  ylab("Population in 100,000") +
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5))
```



Based on the cities which are selected the created function seems to work as desired. The returned results represent an expected result since cities with a larger population have a higher probability to be chosen. The list shows that for example the two of the three largest cities are chosen by the function (Stockholm and Malmö). The two histograms also show that of all cities, the frequency of smaller cities is much bigger than the frequency of the larger cities which is extremely small. In contrast, analysing only the selected cities, this frequency of the higher populated cities relatively increased a lot, since the bars are much higher now. This in indication for the desired performing: Even if originally the frequence of these cities with high populations is extremely small, they will be chosen by the function (sine the probability to get chosen is higher for these cities).

The plot *Selected cities and their population* summarizes the selected cities graphically and underlines that cities with a much higher population than the mean (red line) are chosen.

Question 2

Double exponential distribution

Inverse CDF method:

To obtain X as a function of Y, the inverse CDF method is used. By integrating the pdf of the double exponential distribution, the CDF is obtained. From this function, we want to isolate X on one side of the equation by $Y = F(X) \rightarrow X = G(Y)$, where $F(X)$ is the cumulative distribution function (CDF) of the double exponential distribution. The trick is that the properties of any cdf is that the range of x might differ, but all distributions have a density area that is 1, meaning that the y-axis of this cdf has the interval [0, 1]. When we solve the inverse of this cdf, we simply flip the axis, meaning that Y is in the range of [0, 1]. We can thus draw uniformly distributed values and make use of:

$$U \sim Uni(0,1) \quad \rightarrow \quad G(U) \sim f(...),$$

where f(...) is the desired distribution (in this case the double exponential distribution).

The double exponential distribution is tricky in the sense that it contains the absolute sign $|x - \mu|$, which means that we should consider two cases; $x < \mu$ and $x > \mu$ separately. Also, since the double exponential distribution is symmetric around μ, we make use of the symmetrical properties:

$$\int_{-\infty}^{\mu} f(v)dv = \frac{1}{2}$$

The pdf and theoretical cdf can be seen below.

$$\begin{aligned} \textbf{pdf}: \qquad & f(v) = \frac{\alpha}{2} e^{-\alpha|v-\mu|} \\ \textbf{cdf}: \qquad & F(v) = \int_{-\infty}^{x} f(v)dv \end{aligned}$$

To obtain the formulas for the cdf, we take the integral of the pdf according to the previously mentioned steps.

$$\begin{aligned} \mathbf{x} < \mu: \qquad & F_1(x) = \int_{-\infty}^{x} f(v)dv = \frac{1}{2}\int_{-\infty}^{x} \alpha e^{-\alpha(-v+u)}dv = \frac{1}{2}\int_{-\infty}^{x} \alpha e^{\alpha(v-u)}dv = \\ & = \Big[e^{\alpha(v-\mu)}\Big]_{-\infty}^{x} = \frac{1}{2}\Big(e^{\alpha(x-\mu)} - e^{\alpha(-\infty-\mu)}\Big) = \frac{1}{2}e^{\alpha(x-\mu)} \\ \mathbf{x} > \mu: \qquad & F_2(x) = \int_{-\infty}^{x} f(v)dv = \frac{1}{2} + \int_{\mu}^{x} f(v)dv = \frac{1}{2} + \frac{1}{2}\int_{\mu}^{x} \alpha e^{\alpha(\mu-v)}dv = \\ & = \frac{1}{2} + \frac{1}{2}\Big[-e^{\alpha\mu-\alpha v}\Big]_{\mu}^{x} = \frac{1}{2} + \frac{-1}{2}\Big(e^{\alpha\mu-\alpha x} + e^{\alpha\mu-\alpha\mu}\Big) = \frac{1}{2}\Big(2 - e^{\alpha(\mu-x)}\Big) \end{aligned}$$

Now that the cdf is defined, we can solve $x = G(y)$. Also, for the inverse CDF, there exists two solutions, one for $x > \mu$ and one for $x < \mu$:

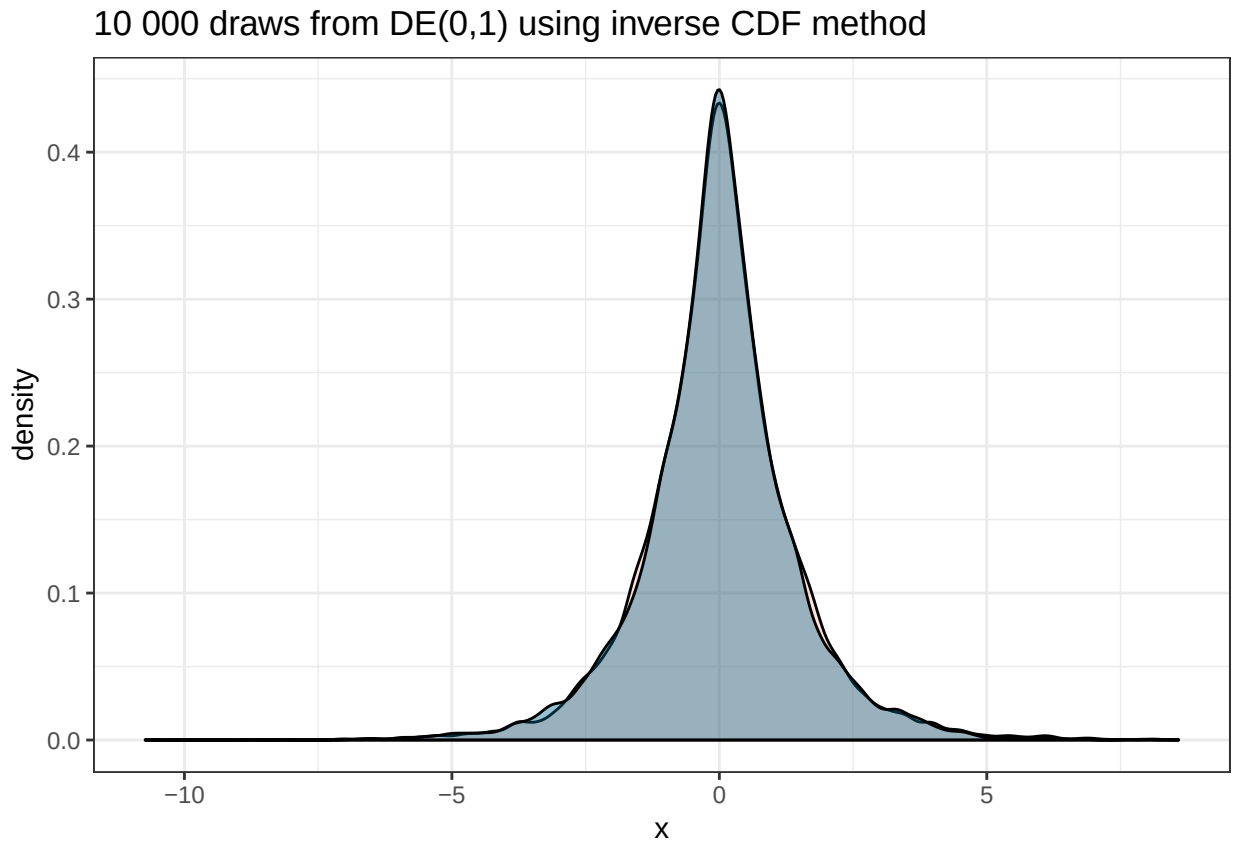
$$\mathbf{x} < \mu: \quad y = F_1(x) = \frac{1}{2}e^{\alpha(x-\mu)} \Rightarrow ln(2y) = \alpha(x-\mu) \Rightarrow \frac{ln(2y)}{\alpha} + \mu = x = G_1(y)$$

$$\mathbf{x} > \mu: \quad y = F_2(x) = \frac{1}{2}\left(2 - e^{\alpha(\mu-x)}\right) \Rightarrow 2(y-1) = -e^{\alpha(\mu-x)} \Rightarrow ln(2-2y) = \alpha(\mu-x) \Rightarrow$$

$$\Rightarrow \frac{ln(2-2y)}{\alpha} = \mu - x \Rightarrow x = \mu - \frac{ln(2-2y)}{\alpha} = G_2(y)$$

With all the formulas defined, we can create a function that simulates uniformly distributed values and transform them to $X \sim DE(0, 1)$.

```
my_DE = function(n = 10, mu = 0, alpha = 1){

  # Random draws from uniform(0, 1):
  u = runif(n = n, min = 0, max = 1)

  # quantiles (x):
  q = sapply(u, FUN = function(u){
    # Two cases; x < mu & x > mu. Generate
    # quantiles from both, where u > .5 means
    # that x > mu (symmetric distribution)
    if(u < .5){
      (log(2*u)/alpha) + mu
    } else {
      mu - (log(2 - 2*u)/alpha)
    }
  })

  # Convert quantiles to density:
  d = (alpha/2) * exp(-alpha*abs(q - mu))

  true_dist = rmutil::rlaplace(n = n)
  data.frame("x" = q, "density" = d, true_dist)
  # dnorm(qnorm(p = c(0,.1,.5,.9,1)))
}

# Data.frame with 10000 draws (x & density):
set.seed(123456)
df = my_DE(n = 10000)

# Plot:
ggplot(df) + geom_density(aes(true_dist), fill = "coral1", alpha = .2) +
  geom_density(aes(x), fill = "deepskyblue4", alpha = .4) + theme_bw() +
  labs(title = "10 000 draws from DE(0,1) using inverse CDF method",
       x = "x")
```



The blue distribution sampled with the inverse CDF method follows the 'true' red `rmutil::dlaplace` distribution very well, an indication that the results are reasonable.

Acceptance/rejection method with majorizing density

In order to use the acceptance/rejection method with a majorizing density we must ensure that the proposal density g_y is always greater than the target density f_x, with the aid of a constant c; $cg_y(x) \geq f_x(x)$, for all x. We must also ensure that the support for the target density is the same or a subset of the proposal density.

I order to obtain c, we use $c \geq \frac{f_x(x)}{g_y(x)} = h(x)$ to obtain the function h. By differentiating $\frac{dh}{dx} = h'(x)$ we find the intersection(s) between the proposal and target distributions. By solving $h'(x) = 0 \Rightarrow x = \mathbf{a}$ and taking $h(\mathbf{a})$ we obtain the potential point(s) for c. The maximizing function is then: $c \cdot g_y$. It is desirable to choose a c that is as small as possible, because the majorizing function is better to use as a realization of the target distribution.

$$\left.\begin{aligned} f_x(x) &= \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}, \quad x \in (-\infty, \infty) \\ g_y(x) &= \frac{1}{2} e^{-|x|}, \qquad\quad x \in (-\infty, \infty) \end{aligned}\right\} \Rightarrow \frac{f_x(x)}{g_y(x)} = h(x) = \sqrt{\frac{2}{\pi}} e^{|x| - \frac{x^2}{2}}$$

Differentiating $h(x)$ yields stationary points:

$$\frac{f_x(x)}{g_y(x)} = \frac{\frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}}}{\frac{1}{2}e^{-|x|}} = \frac{e^{-\frac{x^2}{2}}}{\sqrt{2\pi}} \cdot 2e^{-|x|} = \frac{2e^{-\frac{x^2}{2}-|x|}}{\sqrt{2\pi}}$$

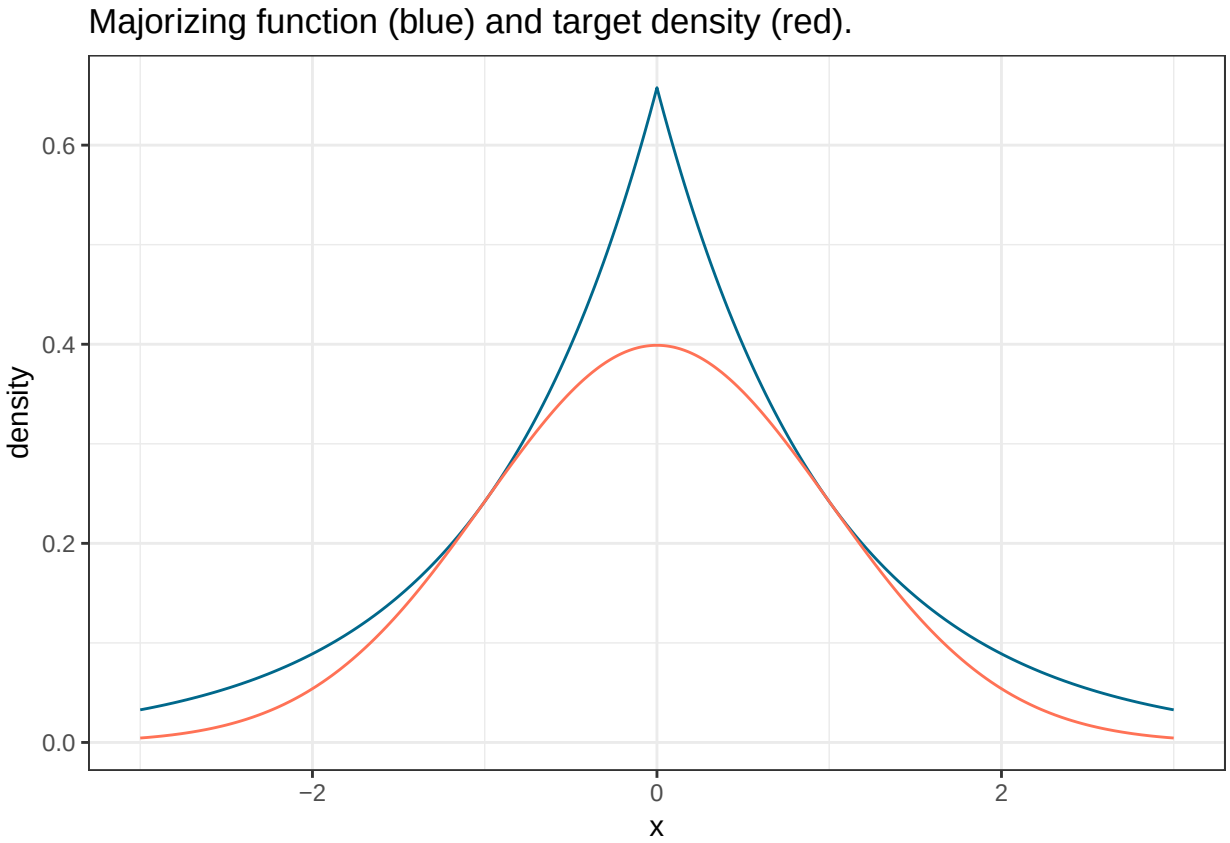
$$\frac{dh}{dx} = 0 \Leftrightarrow \sqrt{\frac{2}{\pi}} e^{|x| - \frac{x^2}{2}} \cdot \frac{d}{dx}\left(|x| - \frac{x^2}{2}\right) = 0 \Leftrightarrow$$
$$\Leftrightarrow \sqrt{\frac{2}{\pi}} e^{|x| - \frac{x^2}{2}} \left(\frac{x}{|x|} - x\right) = 0 \Leftrightarrow x = \pm 1$$

Since both f_x and g_y are symmetrical, so is h, and therefore $h(-1) = h(1)$. We solve for c by taking the stationary points as input to $h(x)$:

$$h(\pm 1) = \sqrt{\frac{2}{\pi}} e^{1 - \frac{1}{2}} = \sqrt{\frac{2}{\pi}} e^{\frac{1}{2}} = \sqrt{\frac{2e}{\pi}} = 1.315489 = c$$

With the obtained c, we can check whether or not the majorizing function $c \cdot g_y \geq f_x$ is larger than the target distribution.

```
# Derived c:
c = sqrt((2*exp(1)) / pi)

# Functions g_y(x) anf f_x(x)
f = function(x) (1 / sqrt(2*pi)) * exp(-(1/2) * x^2)
g = function(x) c * (1 / 2) * exp(-abs(x))

# Chosen sequence...
x = seq(-3, 3, 0.01)

library(dplyr)
# Plot:
data.frame(x = x, g = g(x), f = f(x)) %>%
  ggplot(., aes(x)) + theme_bw() +
  labs(title = "Majorizing function (blue) and target density (red).", y = "density") +
  geom_line(aes(y = g, color = "c*g = c*DE(0, 1)"), color = "deepskyblue4") +
  geom_line(aes(y = f, color = "g = N(0, 1)"), color = "coral1")
```



The results seem to hold! With the derived c, we can use the acceptance/rejection algorithm:

1. Generate y from $g_y = DE(0, 1)$.
2. Generate (indepentently!) u from $Unif(0, 1)$.
3. **If** $u \leq \frac{f_x(y)}{c \cdot g_y(y)}$ **then** set $X = Y$, **else** return to step 1.

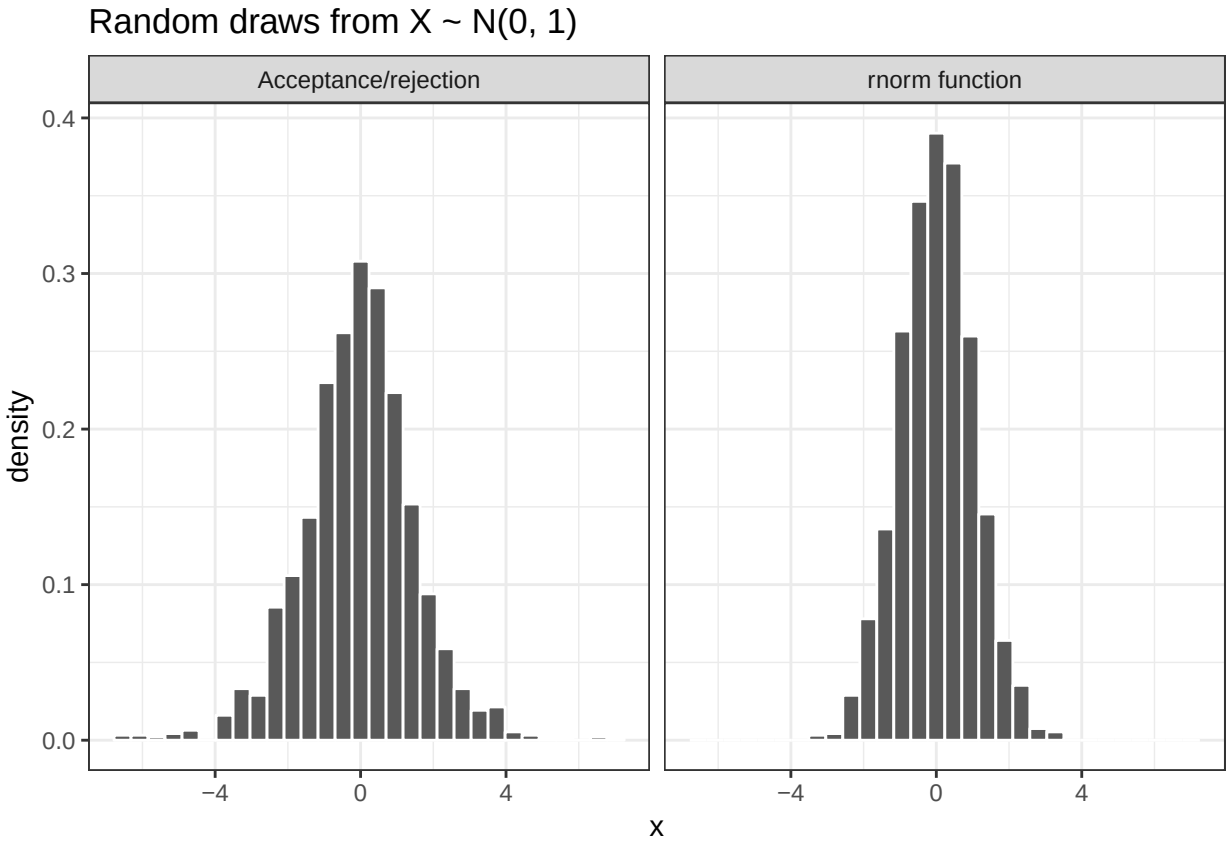
In order to obtain 2000 random numbers, where there is uncertainty (randomness) of how many will be rejected beforehand, a while loop is used. The function `my_DE` defined in the previous sub assignment is used for sampling from the proposal distribution, each time with n = 1 as input.

```
# Derived c:
c = sqrt((2*exp(1)) / pi)

# Acceptance/rejection algorithm (ARA) function:
ara = function(n = 2000){

  # init_value for selected and rejected points:
  n_sel = 0
  n_rej = 0

  # Init empty vector with accepted samples:
  X = c()

  # Loop will select samples until n is reached:
  while(n_sel < n){

    # Generate n DE(0,1) samples using inverse CDF:
```

```
    DE = my_DE(n = 1)    # function from previous assignment
    y  = DE$x            # the x-axis value that...
    gy = DE$density      # ...generates density in DE(0,1)

    # Generate independent(!) sample from Unif(0,1):
    u = runif(n = 1)

    # Convert u to N(0,1) density:
    # fx = dnorm(u)
    fx = f(u)

    # Conditional sampling:
    if( (u <= fx / (c*gy)) ){
      # add 1 to counter:
      n_sel    = n_sel + 1

      # assign chosen sample to X:
      X[n_sel] = y
    } else {
      # add 1 to rejection counter:
      n_rej = n_rej + 1
    }
  }

  # Assign n_rej and n_sel to parent environment:
  n_rej <<- n_rej
  n_sel <<- n_sel
  return(X)
}
```

We call the function with a seed to obtain reproducible results.

```
# Function call:
set.seed(123456)
ara = ara(n = 2000)

# Plot:
data.frame("x"      = c(ara, rnorm(n = 2000)),
           "method" = c(rep("Acceptance/rejection", 2000),
                        rep("rnorm function", 2000)) %>% factor()) %>%
  ggplot(., aes(x)) + ggtitle("Random draws from X ~ N(0, 1)") +
  geom_histogram(aes(y = ..density..), bins = 30, color = "white") +
  theme_bw() + facet_grid(~method, scales = "fixed")
```



The histograms look similar, although the distribution from the acceptance/rejection method looks to have slightly higher standard deviation than the target distribution.

```
R = n_rej/n_sel
R
```

```
## [1] 0.224
```

The expected number of draws requried to obtain an accepted proposal in this algorithm is c. The expected acceptance rate is thus $\frac{1}{c} = 0.7602$. The average rejection rate is *1 - acceptance rate* $= 0.2398$, which is fairly close to the rejection rate `R` from the algorithm presented above.

Appendix

All code is shown in the report.