Group 21 Lab 3 Report

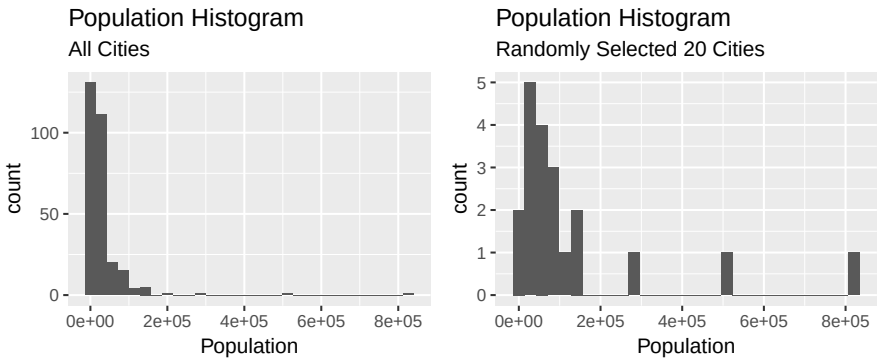
Stefano Toffol (steto820), Mim Kemal Tekin (mimte666)

08 March, 2019

Chapter 1

Lab 3

Question 1: Cluster Sampling



In this task we do random sampling without replacement in a cities of Sweden dataset which has populations as an attribute. We use this population information to calculate proportional probabilities. We calculate proportions by this formula: $P_i = \frac{x_i}{\sum_i^n x_i}$. Probabilities are between 0-1 and we will use a uniform random number generator to pick cities proportional randomly. After picking a city we should ignore that city for the next element of sample in order to achieve sampling without replacement. This event will effect proportions and total population in every iteration.
After creating this sampling function we run this function 20 times in order to get a sample. We have following 20 cities in our sample ordered by decreasing population: *Stockholm, Göteborg, Malmö, Linköping, Helsingborg, Lund, Eskilstuna, Södertälje, Kristianstad, Kalmar, Mölndal, Östersund, Varberg, Ängelholm, Upplands Väsby, Katrineholm, Ronneby, Höganäs, Oxelösund, Mellerud.*
The plot above shows us distribution of Population over all cities and our sample. As we can see many cities in Sweden has population less than 200000. We have only 4 outlier cities has population more than 200000, which means those cities have bigger proportion of population in Sweden. This shows us when we pick cities randomly by proportional probabilities, it is more likely to pick these cities more than the cities which has less population. Below, we can see a comparison table of the most crowded 6 cities in the all data and our sample and the proportions in the real data. It is visible to see random number generation function captured 4 cities in the most crowded 6 cities.

All	Sample	Proportion
Stockholm	Stockholm	0.0887962
Göteborg	Göteborg	0.0543140
Malmö	Malmö	0.0314655
Uppsala	Linköping	0.0208498
Linköping	Helsingborg	0.0154903
Västerås	Lund	0.0145531

Question 2

In order to generate from a double exponential distribution by using the inverse *CDF method*, we first need to obtain the cumulative distribution function F_x. We are given the density function of the variable, that is:

$$f(x;\mu,\alpha) = \frac{\alpha}{2}\exp(-\alpha|x-\mu|) \quad \text{with: } \alpha > 0$$

The cumulative distribution function can be derived from the density function integrating it over x:

$$F_X(x;\mu,\alpha) = \int_{-\infty}^{x} f_X(t)dt = \frac{\alpha}{2}\left[-\frac{1}{\alpha}\frac{|t-\mu|}{(t-\mu)}\exp(-\alpha|t-\mu|)\right]_{-\infty}^{x} = -\frac{1}{2}\left[sign(t-\mu)\exp(-\alpha|t-\mu|)\right]_{-\infty}^{x} =$$

$$= \begin{cases} -\frac{1}{2}\left\{\left[e^{-\alpha(\mu-t)}\right]_{-\infty}^{\mu} + \left[e^{-\alpha(t-\mu)}\right]_{\mu}^{x}\right\} = 1 - \frac{1}{2}e^{-\alpha(t-\mu)} & \text{if } x \geq \mu \\ -\frac{1}{2}\left[e^{-\alpha(\mu-t)}\right]_{-\infty}^{x} = \frac{1}{2}e^{-\alpha(\mu-t)} & \text{if } x < \mu \end{cases}$$

$$= \frac{1}{2} + \frac{1}{2}sgn(x-\mu)\Big(1-\exp(-\alpha|x-\mu|)\Big)$$

We now need the inverse of the comulative distribution function, $F^{-1}(p)$, that is equal to:

$$2|p-\frac{1}{2}| = 1-\exp(-\alpha|x-\mu|)$$
$$\ln(1-2|p-\frac{1}{2}|) = -\alpha|x-\mu|$$
$$\downarrow$$
$$F^{-1}(p) = \mu - \frac{1}{\alpha}sgn(p-\frac{1}{2})\ln(1-2|p-\frac{1}{2}|)$$

We have finally managed to invert the CDF of the Laplace distribution. In order to randomly generate number following the density to this random variable in sufficient to generate random numbers from a *Uniform* distribution U and compute $F_X^{-1}(U;\mu,\alpha)$.
The task is achieved with the following code:

```r
# Densitiy function of a Laplace distribution
dlaplace <- function(x, mu=0, alpha=1) {

  return( (alpha/2)*exp(-alpha*abs(x-mu)) )

}

# Inverse of the CDF of a Laplace distribution
inv_CDF_laplace <- function(p, mu=0, alpha=1) {

  return( mu-(1/alpha)*sign(p-0.5)*log(1-2*abs(p-0.5)) )

}

# Random generation from Laplace distribution
rlaplace <- function(n, mu=0, alpha=1) {

  rand_u <- runif(n)
  return( inv_CDF_laplace(rand_u, mu, alpha) )

}

set.seed(12345)

sample_laplace01 = rlaplace(10000, 0, 1)
sample_laplace04 = rlaplace(10000, 0, 4)
sample_laplace32 = rlaplace(10000, 3, 2)
sample_laplace_neg = rlaplace(10000, -2, 0.5)
```

The generated number are plotted in Figure 1.1. Various combinations of the parameters μ and α have been tried to test the validity of our function. In particular, the pairs (0,1), (0,4), (3,2) and (-2,0.5) for the parameters (μ, α) have been chose to carry out these visual tests.
As one can observe, the generated data follows perfectly the theoretical distribution of the random variable, no matter the choice of the parameters μ and α.

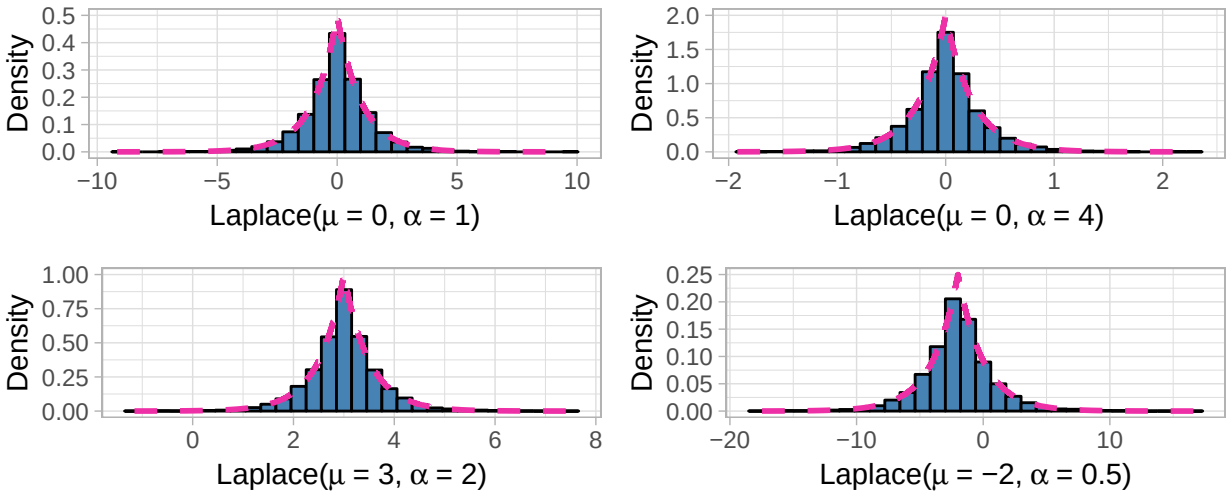


Figure 1.1: Density histograms of the randomly generated numbers from the Laplace distribution, with different choiches of the parameters μ and α.

The random generator we have just created can also be used to generate data from other distributions, such as the Normal one exploiting the *acceptance/rejection* methods. In fact for the Laplace distribution, with its heavy tails and a shape similar to the gaussian function, exists a constant k such that $k \cdot f_Y(x) \geq f_X(x) \quad \forall x$ where $f_Y(x)$ refers to the Laplace distribution and $f_X(x)$ to the normal distribution.
Setting the parameters of both variables equal to (1, 0) leads to the following density distributions:

$$f_X(x) = \frac{1}{\sqrt{2\pi}} \exp -\frac{x^2}{2}$$
$$f_Y(x) = \frac{1}{2} \exp -|x|$$

The next task is to find a suitable k, in other words a constant that grants to the Laplace distribution the property of being a majorizing density. We could simply take a very large k in order to be sure our Laplace density is majorizing the target one, however this would lead to a huge number of rejection, making the algorithm inefficient (the total number of draws will in fact follow a geometric distribution $Geom(1/k)$). Therefore the best k is the one that leads to the equality of the two density function when the difference between the two original ones reach its maximum. This can be solve analytically, in fact:

$$k \cdot \frac{1}{2} \exp -|x| \geq \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right) \quad \text{for each } -\infty \leq x \leq +\infty$$
$$\implies \quad k \geq \sup_x \sqrt{\frac{2}{\pi}} \exp\left(|x| - x^2/2\right)$$
$$k \geq \sqrt{\frac{2}{\pi}} \exp\left(\sup_x (|x| - x^2/2)\right)$$

where, since $(|x| - x^2/2)$ is an even function, we can maximize on just $(x - x^2/2)$ with $x > 0$:

$$\implies \quad x = 1 \quad \text{which implies} \quad k_{opt} = \sqrt{\frac{2e}{\pi}}$$

We now have all the ingredients necessary for the algorithm. We proceed in the implementation and generation of 2000 random numbers from $N(0, 1)$ with the following code:

```
# Takes as arguments the following:
#       · The number n of observations to generate;
#       · The target density f (standar normal distribution);
#       · The majorizing density g (standard laplace distribution);
#       · The function rg to generate random numbers according to the density g;
#       · The majorizing constant k.

acc_rej_norm <- function(n, f, g, rg, k) {

  results <- double(n)  # Prepare space for the generated values
  ntry <- 0  # Counter to keep track of the draws

  for(i in 1:n) {

    accepted = F  # Keep generating from rg() until acceptance
    while(!accepted) {
      ntry <- ntry+1  # One more draw
```

```
      temp_g <- rg(1)  # Generate a number from Laplace
      temp_unif <- runif(1)  # Generate a number from uniform (probability)
      if(temp_unif <= f(temp_g)/(k*g(temp_g))) {  # Is it acceptable?
        accepted <- T  # Exit from while...
        results[i] <- temp_g  # ...and save the result
      }
      # If not accepted the algorithm keeps going (we only increase the counter)
    }

  }

  return( list(generated = results, n_draws = ntry) )

}

k <- sqrt( (2*exp(1)) / pi )
random_norm_AR <- acc_rej_norm(2000, dnorm, dlaplace, rlaplace, k)
random_norm_st <- rnorm(2000)
```

Using the just implemented algorithm we are now able to generate numbers from the standard normal distribution. The total number of draws necessary before accepting the required 2000 values was 2645, meaning that the acceptance rate is equal to 0.756144. The rate is pretty high, but it still implies that almost a forth of the generated number where rejected in the process. The expected rejection rate is the mean of the random variable $Geom(1/k)$, that is equal to $1/k_{opt}$ itself, so 0.760173. This value closely resamble what we observed: the two rates just differ of 0.00403.
As visible in Figure 1.2, the numbers generated using our algorithm follow the shape of a standard normal distribution (left, dashed line) and their histogram is almost identical to the one from the data generated using the standard R functions (right).

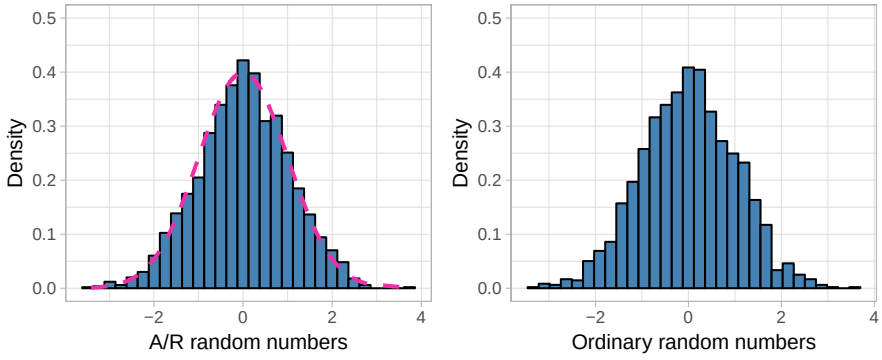


Figure 1.2: Histograms of the random generated numbers of the normal distribution using A/R algorithm (left) and the standard R function (right).

As extra proof of the correctness of our k_{opt}, we can observe in Figure 1.3 how the standard Laplace distributions dominate the Gaussian one after the multiplication with k_{opt}. Approximately around the values ± 1, the two lines overlaps exactly, showing once more that our computation of k_{opt} was indeed correct.

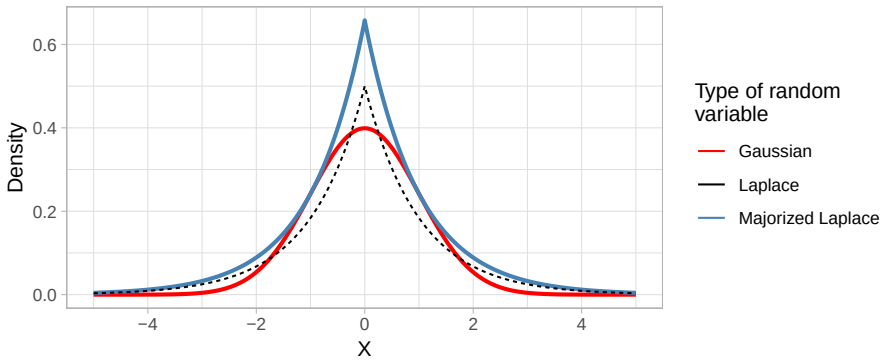


Figure 1.3: Shape of the standard Laplace (dotted black), standard Gaussian (solid red) and majorized standard Laplace (solid blue) distributions.

```
# Code for the inline results
# Count of Rejection:
random_norm_AR$n_draws  # Total number of draws
round(2000/random_norm_AR$n_draws, 6)  # Observed rejection rate
round(1/k, 6)  # Expected rejection rate
round(abs(2000/random_norm_AR$n_draws-1/k), 6)  # Difference between expected
                                                # and observed rejection rate
```

Appendix

```
knitr::opts_chunk$set(echo = F, warning = F, error = F, message = F,
                      fig.height = 2.7,
                      fig.align='center', out.width="70%")
library(ggplot2)
library(gridExtra)
# Import data
# Eastern Europe encoding
data = read.csv2("../datasets/population.csv", fileEncoding = "ISO8859-15")
# Calculate probobilities from proportions of populations and sort them by this
data = data[order(data$Population, decreasing = T), ]
data$probs = data$Population/sum(data$Population)

# function that returns a city by randomly
# selected according to probability sceme
random_city = function(mydata){
  # get probabilities (proportions between 0-1)
  mydata$probs = mydata$Population/sum(mydata$Population)
  # get uniform random number
  rand_num = runif(n = 1)
  message(rand_num)
  sums = 0
  # find the city which is selected by probabilities
  for(i in 1:nrow(mydata)){
    if(sums > rand_num)
      return(mydata[i-1,])
    sums = sums + mydata$probs[i]
  }

}

# pick the random indexes
set.seed(1519)
random_cities = data.frame(random_city(data))
for(i in 2:20){
  ignored_indexes = which(data$Municipality %in% random_cities$Municipality)
  random_cities = rbind(random_cities, random_city(data[-ignored_indexes,]))
}

plot1 = ggplot(data) +
  geom_histogram(aes(x = Population)) +
  labs(title = "Population Histogram",
       subtitle = "All Cities")


plot2 = ggplot(random_cities) +
  geom_histogram(aes(x = Population)) +
  labs(title = "Population Histogram",
       subtitle = "Randomly Selected 20 Cities")

grid.arrange(grobs = list(plot1, plot2), nrow=1)
```

```r
sorted_rand_cities = random_cities[order(random_cities$Population, decreasing = T), ]

library(kableExtra)
kableExtra::kable(data.frame(All = head(data$Municipality),
                         Sample = head(sorted_rand_cities$Municipality),
                         Proportion = head(data$probs)), booktabs = T) %>%
  kable_styling(bootstrap_options = "striped", full_width = F)


# ------------------------------------------------------------------------------
# A2
# ------------------------------------------------------------------------------


# Densitiy function of a Laplace distribution
dlaplace <- function(x, mu=0, alpha=1) {

  return( (alpha/2)*exp(-alpha*abs(x-mu)) )

}

# Inverse of the CDF of a Laplace distribution
inv_CDF_laplace <- function(p, mu=0, alpha=1) {

  return( mu-(1/alpha)*sign(p-0.5)*log(1-2*abs(p-0.5)) )

}

# Random generation from Laplace distribution
rlaplace <- function(n, mu=0, alpha=1) {

  rand_u <- runif(n)
  return( inv_CDF_laplace(rand_u, mu, alpha) )

}

set.seed(12345)

sample_laplace01 = rlaplace(10000, 0, 1)
sample_laplace04 = rlaplace(10000, 0, 4)
sample_laplace32 = rlaplace(10000, 3, 2)
sample_laplace_neg = rlaplace(10000, -2, 0.5)


library(ggplot2)

p1 <- ggplot() +
  geom_histogram(aes(x = sample_laplace01, y = ..density..),
                 col = "black", fill = "steelblue") +
  geom_line(aes(x = sample_laplace01, y = dlaplace(sample_laplace01)),
            col = "maroon2", size = 1, lty = 2) +
  labs(x = expression("Laplace(" * mu ~ "= 0," ~ alpha ~ "= 1)"),
       y = "Density") +
```

```
  theme_light()
p2 <- ggplot() +
  geom_histogram(aes(x = sample_laplace04, y = ..density..),
                 col = "black", fill = "steelblue") +
  geom_line(aes(x = sample_laplace04, y = dlaplace(sample_laplace04, 0, 4)),
            col = "maroon2", size = 1, lty = 2) +
  labs(x = expression("Laplace(" * mu ~ "= 0," ~ alpha ~ "= 4)"),
       y = "Density") +
  theme_light()
p3 <- ggplot() +
  geom_histogram(aes(x = sample_laplace32, y = ..density..),
                 col = "black", fill = "steelblue") +
  geom_line(aes(x = sample_laplace32, y = dlaplace(sample_laplace32, 3, 2)),
            col = "maroon2", size = 1, lty = 2) +
  labs(x = expression("Laplace(" * mu ~ "= 3," ~ alpha ~ "= 2)"),
       y = "Density") +
  theme_light()
p4 <- ggplot() +
  geom_histogram(aes(x = sample_laplace_neg, y = ..density..),
                 col = "black", fill = "steelblue") +
  geom_line(aes(x = sample_laplace_neg, y = dlaplace(sample_laplace_neg, -2, .5)),
            col = "maroon2", size = 1, lty = 2) +
  labs(x = expression("Laplace(" * mu ~ "= -2," ~ alpha ~ "= 0.5)"),
       y = "Density") +
  theme_light()

gridExtra::grid.arrange(p1, p2, p3, p4, ncol = 2)


# Takes as arguments the following:
#       · The number n of observations to generate;
#       · The target density f (standar normal distribution);
#       · The majorizing density g (standard laplace distribution);
#       · The function rg to generate random numbers according to the density g;
#       · The majorizing constant k.

acc_rej_norm <- function(n, f, g, rg, k) {

  results <- double(n)  # Prepare space for the generated values
  ntry <- 0  # Counter to keep track of the draws

  for(i in 1:n) {

    accepted = F  # Keep generating from rg() until acceptance
    while(!accepted) {
      ntry <- ntry+1  # One more draw
      temp_g <- rg(1)  # Generate a number from Laplace
      temp_unif <- runif(1)  # Generate a number from uniform (probability)
      if(temp_unif <= f(temp_g)/(k*g(temp_g))) {  # Is it acceptable?
        accepted <- T  # Exit from while...
        results[i] <- temp_g  # ...and save the result
      }
      # If not accepted the algorithm keeps going (we only increase the counter)
```

```
    }

  }

  return( list(generated = results, n_draws = ntry) )

}

k <- sqrt( (2*exp(1)) / pi )
random_norm_AR <- acc_rej_norm(2000, dnorm, dlaplace, rlaplace, k)
random_norm_st <- rnorm(2000)


p1 <- ggplot() +
  geom_histogram(aes(x = random_norm_AR$generated, y = ..density..),
                 col = "black", fill = "steelblue") +
  geom_line(aes(x = random_norm_AR$generated, y = dnorm(random_norm_AR$generated)),
            col = "maroon2", size = 1, lty = 2) +
  scale_y_continuous(limits = c(0,0.5)) +
  labs(x = "A/R random numbers", y = "Density") +
  theme_light()
p2 <- ggplot() +
  geom_histogram(aes(x = random_norm_st, y = ..density..),
                 col = "black", fill = "steelblue") +
  # geom_line(aes(x = sample_laplace01, y = dlaplace(sample_laplace01)),
  #           col = "maroon2", size = 1) +
  scale_y_continuous(limits = c(0,0.5)) +
  labs(x = "Ordinary random numbers", y = "Density") +
  theme_light()

gridExtra::grid.arrange(p1, p2, ncol=2)



x <- seq(-5, 5, 0.01)
df_plot <- data.frame(x = x, densities = c(dlaplace(x), dnorm(x), dlaplace(x)*k),
                      col = c(rep("Laplace", 1001),
                              rep("Gaussian", 1001),
                              rep("Majorized Laplace", 1001)),
                      lty = c(rep("2", 1001), rep("1", 1001), rep("1", 1001)),
                      size = c(rep("2", 1001), rep("1", 1001), rep("1", 1001)))
ggplot(aes(x = x, col = col, lty = lty), data = df_plot) +
  geom_line(aes(y = densities, size = size)) +
  scale_color_manual(values = c("red", "black", "steelblue"),
                     name = "Type of random\nvariable") +
  scale_linetype_discrete(guide = F) +
  scale_size_manual(guide = F, values = c(1, 0.5)) +
  scale_x_continuous(breaks = seq(-4, 4, 2)) +
  labs(y = "Density", x = "X") +
  theme(legend.position = "bottom") +
  theme_light()
```

```r
# Code for the inline results
# Count of Rejection:
random_norm_AR$n_draws  # Total number of draws
round(2000/random_norm_AR$n_draws, 6)  # Observed rejection rate
round(1/k, 6)  # Expected rejection rate
round(abs(2000/random_norm_AR$n_draws-1/k), 6)  # Difference between expected
                                                # and observed rejection rate
```