Lab3

Andreas Stasinakis

January 31, 2019

Contents

Question 1 Cluster sampling

1.1 Import csv, encoding data

```
# import the data
#using fileEncoding in order to recognize the swedish letters
cities = read.csv2("../population.csv", fileEncoding = "ISO8859-15")
cities = cities[order(cities$Population,decreasing = TRUE),]
```

1.2 Function generates numbers using probabilities

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).

```
set.seed(1519)
# function choose an element of a data frame.
Generator = function(data){
  #order the data
  #data = data[order(data$Population),]

  #vector to store the probabilities
  prob =c()

  # a for loop to calculate the probability for each city.
  for (i in 1:nrow(data)){

    prob[i] = data[i,2]/sum(data$Population)
  }

  # add the probabilities in the data
  data$probabilities = prob
  data$cumulative = cumsum(data$probabilities)
```

```
  # Use a random sample from the list using runif
  # Choose the one closest to the random number generated from runif
  # use the function findinterval which finds the left extreme from the
  #interval in which the random number is.

  random = runif(1)
  city_index = findInterval(x = random , c(0,data$cumulative))

  return(city_index)
}
```

In this task we have to implement a function which returns the index of a city in a list of 300 cities in Sweden. Firsty the function calculates the probability for each city to be choosen. This probabilities are proportional to the number of the cities. So for the i_{th} city the probability to be choosen is $\frac{\text{population}}{\text{sum of the populations}}$. The cumulative propabilities are also computed. A random number in the interval $[0, 1]$ is given and the choice of the city depends on that number. More specific, using the built in function findInterval we find the interval in which this random number lies and we choose the index for the lower extreme. This is the index which the function return and we can use it in oredr to obtain information for a specific city.

1.3 Using the function to choose 20 cities without replacement.

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.

```
# function for calculating a vector of length n for a given dataset
fun_cities = function(n,data){

  #vector to store n  cities
  all_cities = c()

  #For loop for store one city without replacement
  for (i in 1:n) {

    # use the function to find one city for the list
    index = Generator(temp_cities)

    #store the city choosen
    all_cities[i] = as.character(temp_cities$Municipality[index])

    # eliminate the city
    temp_cities = temp_cities[-index,]
    }

    return(all_cities)
}
```

In this task we have to use the function from task 2 in order to create a list including 20 Swedish cities using a random sampling without replacement. In order for this target to be achieved we use the function 20 times *but* every time we have to exclude the index which is choosen from the dataset.

1.4 Run the function, interpret the list

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

```r
temp_cities = read.csv2("../population.csv", fileEncoding = "ISO8859-15")
temp_cities = temp_cities[order(temp_cities$Population,decreasing = TRUE),]

#Use the function for the data
cities_choosen = fun_cities(n = 20,data = temp_cities)

# prin the result

print(list("The 20 choosen cities are" = cities_choosen))
```

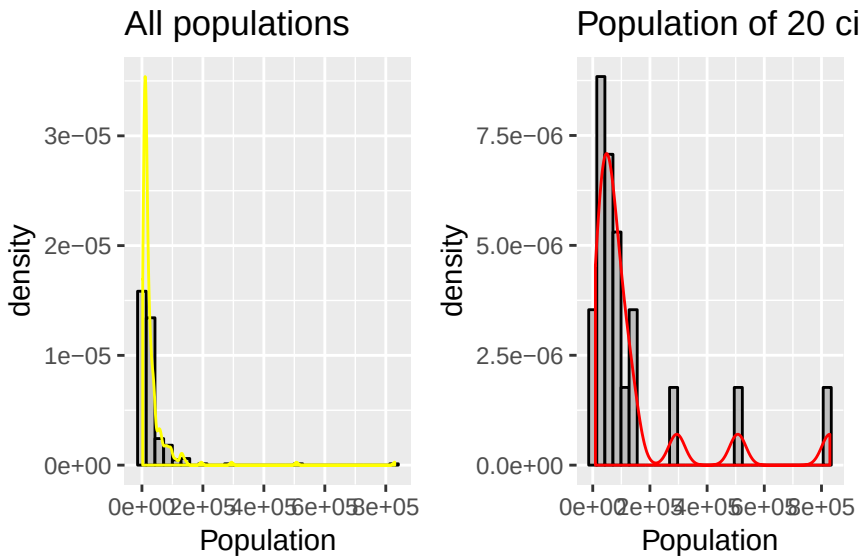
```
## $`The 20 choosen cities are`
##  [1] "Linköping"      "Ängelholm"      "Helsingborg"    "Oxelösund"
##  [5] "Södertälje"     "Kristianstad"   "Kalmar"         "Varberg"
##  [9] "Lund"           "Mölndal"        "Stockholm"      "Höganäs"
## [13] "Ronneby"        "Mellerud"       "Upplands Väsby" "Malmö"
## [17] "Eskilstuna"     "Göteborg"       "Katrineholm"    "Östersund"
```

The names of the 20 cities choosen can be seen above. As we mentioned before the choice of each cities is proportional to the population which means that the higher the population of a city is, the higher probability for this city to be exctract. Looking at the cities we can confirm this assumption because the most crowed cities are selected.

1.5 Histograms, interpret plots and conclusions.

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

```r
library(gridExtra)
library(ggplot2)
# data frames for the histogram

df_all = data.frame(all_cities = cities$Population)

df_20 = data.frame(cities_choosen =
                     cities$Population[cities$Municipality %in% cities_choosen])

plot1 = ggplot(df_all) +
  geom_histogram(mapping = aes(df_all$all_cities,y = ..density..), color = "black", fill = "grey")  +
  geom_density(mapping = aes(df_all$all_cities), color = "yellow")+
  labs(title = "All populations", x = "Population")

plot2 = ggplot(df_20) +
  geom_histogram(mapping = aes(df_20$cities_choosen,y = ..density..), color = "black", fill = "grey") +
  geom_density(mapping = aes(df_20$cities_choosen), color = "red")+
  labs(title = "Population of 20 cities choosen", x = "Population")


grid.arrange(grobs = list(plot1,plot2), ncol = 2)
```



Finally, we plot two histograms(one with all the cities and one with only the 20 choosen). The first plot, for all the cities, seems quite unbalanced. The crowed cities have a really huge impact in the distribution while those cities are the minority of the total number of cities. From the second plot, we can confirm once again our assumption. The cities choosen are the most crowded ones. Also some cities with lower population are selected, as a result the distribution of the data looks more balanced that the one before.

Question 2: Different Distributions(Inverse CDF method)

The double exponential (Laplace) distribution is given by formula:

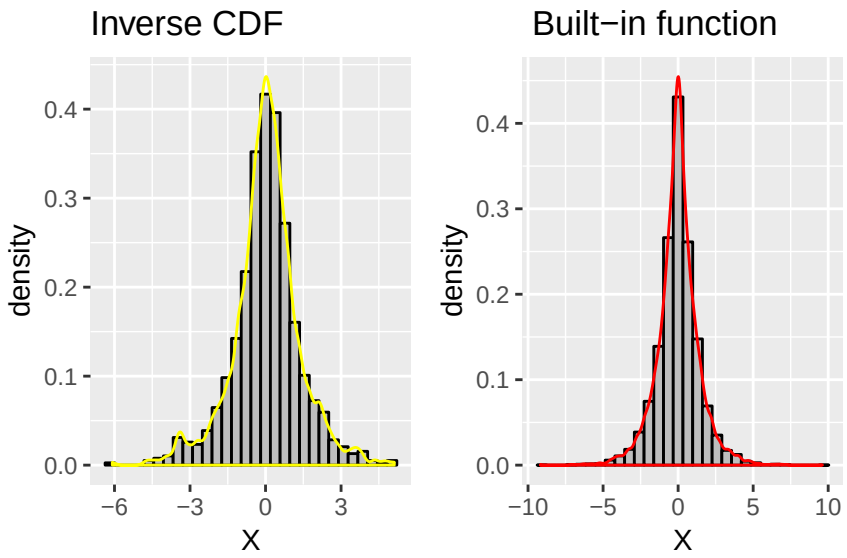
$$DE(\mu, \alpha) = \frac{\alpha}{2} e^{(-\alpha|x-\mu|)}$$

2.1 CDF method: Calculate CDF, inverse CDF and generate numbers, Histogramm, gridExtra.

1. 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.

```
library(ggplot2)
library( VGAM)
library(gridExtra)
set.seed(12345)
#Given the pdf, we have to find the CDF firstly.
# After that we calculate the inverse of the CDF
# We also pay attention in all constrains.
#Specific calculations above in the analysis

# The final function for generate random numbers
# Input is the random numbers for a given distribution
#output is the X using the inverse CDF method

#U <- runif(1000,0,1)
```

```r
CDF_method = function(n){

  #vector to store the random numbers
  X = c()

  # generate number numbers from the known distribution
  U <- runif(n,0,1)

  # a for loop through all the generated numbers
  # Using the inverse CDF, calculate new numbers
  for (i in 1:length(U)){

    # in this procedure we have some constrains
    # The if condition is for each random number

    if (U[i]>1/2){
      X[i] = -log(2- 2*U[i])
    }
    else {
      X[i] = log(2*U[i])
    }
  }
  return(X)
}

X = CDF_method(1000)

# real laplace values
df_data = rlaplace(10000,location = 0,scale = 1)

df = data.frame(X,df_data)

#plot the Generated numbers
plot1 = ggplot(df) +
  geom_histogram(mapping = aes(df$X,y = ..density..), color = "black", fill = "grey")  +
  geom_density(mapping = aes(df$X), color = "yellow")+
  labs(title = "Inverse CDF", x = "X")

plot2 = ggplot(df) +
  geom_histogram(mapping = aes(df$df_data,y = ..density..), color = "black", fill = "grey") +
   geom_density(mapping = aes(df$df_data), color = "red")+
  labs(title = " Built-in function", x = "X")


grid.arrange(grobs = list(plot1,plot2), ncol = 2)
```



In this task, we have to implement the inverse CDF method in order to generate numbers which are double exponential distributed using a random sample of the uniform distribution. The specific steps are appear below.

First of all, given the pdf(probability density function), we have to find the CDF(cumulative distribution function). The general formula for the CDF is :

$$F(x) = \int_{-\infty}^{x} f(x)dx$$

. In this case though, we have absolute value $|x - \mu|$ so we will split it in two cases.

For $x > \mu$:

$$F(X) = \int_{-\infty}^{x} \frac{\alpha}{2} e^{-\alpha(x-\mu)}dx = \int_{-\infty}^{\mu} \frac{\alpha}{2} e^{-\alpha(\mu-x)}dx + \int_{\mu}^{x} \frac{\alpha}{2} e^{-\alpha(x-\mu)}dx = \frac{1}{2} - \frac{1}{2}[e^{-\alpha(x-\mu)} - 1]\Big|_{\mu}^{x} = \frac{1}{2} - \frac{1}{2}e^{-\alpha(x-\mu)} + \frac{1}{2} = 1 - \frac{1}{2}e^{-\alpha}$$

For $x \leq \mu$:

$$F(X) = \int_{-\infty}^{x} \frac{\alpha}{2} e^{\alpha(x-\mu)}dx = \frac{1}{2}e^{\alpha(x-\mu)}\Big|_{\infty}^{x} = \frac{1}{2}e^{\alpha(x-\mu)}$$

After that we have to find the inverse function of the CDF. In order to do that, we split again the cases for $x > \mu$ and $x \leq \mu$.

For $x > \mu$:

$$F(x) = 1 - \frac{1}{2}e^{-\alpha(x-\mu)} = y$$

We solve the equation with respect to x. The final formula is :

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

, where $x > \mu$.

For $x \leq \mu$:

$$F(x) = \frac{1}{2}e^{\alpha(x-\mu)} = y$$

We solve the equation with respect to x again. So the final formula is :

$$x = \mu + \frac{ln(2y)}{\alpha}$$

, where $x \leq \mu$.

In this question, we want to generate numbers $X \sim DE(0, 1)$, where $\mu = 0, \alpha = 1$.

So the inverse CDF for this values is:

$x = -ln(2 - 2y)$, where $x > 0$.

$x = ln(2y)$, where $x \leq 0$.

Finally, we should be careful with the constrains. More specific, for $x > 0$:

$x = -ln(2 - 2y) > 0$. Solving the Inequality we obtain the constrain:

$y > \frac{1}{2}$.

We do the some procedure for $x \leq 0$ and we have that $y \leq \frac{1}{2}$.

We can also comment to the result looking the plot above. The results look reasonable. In general, the double exponential distribution can be seen as the "merging" of two exponential distributions spliced together next to each other. We also plot random values for DE distribution using a built in function of R and the same parameters($\mu = 0, \alpha = 1$). The two plots are quite similar. It is clear that most of the points are close to zero which is correct because we want a distribution with mean = 0. One can observe that the plot for the function we create is smooth, except for the tails where we have some fluctuations. Moreover, the results are not totally symmetric as they should be. In conclusion, the results we obtain using the CDF function look really close to the real ones with some expected differences.

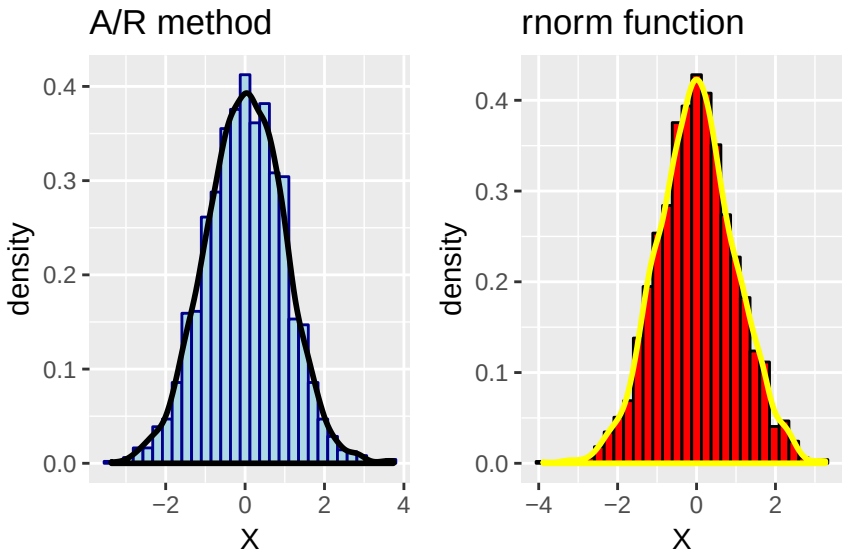
2.2 Acceptance/rejection method

Use the Acceptance/rejection method with DE(0, 1) as a majorizing density to generate 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 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 N(0, 1) using standard rnorm() procedure, plot the histogram and compare the obtained two histograms.

```
set.seed(12345)
# function for the pdf of Normal distribution given $\mu=0, \sigma=1$
normal_pdf = function(x){
  res = (exp(-(x^2)/2))/(sqrt(2*pi))
  return(res)

}

# function for the pdf of the DE distribution
DE_pdf = function(x){

  res = (exp(-abs(x)))/2
  return(res)
}

# the constant M. h(x) = f(x)/g(x). Take the derivative = 0.
# this is the optimal x0. After that h(xo) is the M
# in this case the M is : ( calculations above)
```

```r
M = sqrt((2*exp(1))/pi)

# function using the Acceptance/ Rejection method
# input: length of the sample, Constant M
# pdf for the known distribution(g) and pdf for the one i want to generate(f)
a =0
AR = function(sample,M,g,f){

  #vector to store the numbers
  normal_sample = c()

  #a for loop for the length of the sample the user wants
  for (i in 1:sample) {

    # repeat loop until the generated number verify the condition
    repeat{
      a <<- a +1
      # generate a number from the known distribution
      x = CDF_method(1)

      #generate a number from the uniform distribution
      u = runif(1,0,1)

      # the final condition. The generated number should be less
      # this quantity. The M is a choice of the user.
     if(u<f(x)/(M*g(x))){break}
    }

    #So if the condition is True the algorithm generate a number.
    normal_sample[i] = x
  }
  return(normal_sample)
}


data =  AR(sample = 2000,M,f = normal_pdf,g = DE_pdf)

df = data.frame(x = data)

#plot the Generated numbers
c = ggplot(df) +
  geom_histogram(mapping = aes(df$x,y = ..density..),color="darkblue", fill="lightblue") +
  geom_density(mapping = aes(df$x), color = "black", size =1)+
  scale_y_continuous(breaks = seq(0,1 , by = 0.1)) +
  labs(title = "A/R method ", x = "X")

# generate sample from the rnorm

normal_data = rnorm(n = 2000,0,1)

b = ggplot(as.data.frame(normal_data)) +
  geom_histogram(mapping = aes(normal_data, y = ..density..), color="black", fill="red") +
  geom_density(mapping = aes(normal_data), color = "yellow", size =1)+
```

```
  scale_y_continuous(breaks = seq(0,0.7 , by = 0.1))+
  labs(title = "rnorm function", x = "X")

grid.arrange(grobs = list(c,b), ncol = 2)
```




```
#Calculate the difference betwwen expected and average acceptance rate
rate = abs(1/M - 2000/a)
```

In this task we have to implement the acceptance/ rejection method using $DE(0,1)$ as a majority density in order to generate variables which are normally distributed(N(0,1)). This method is really useful and it is used especially when we can not calculate the Cumulative distribution function. Firstly, we generate a number x of the proposal distribution. In this case, the proposal distribution is the double exponential and we use the function from the task 1.Moreover, we generate one sample u from uniform distribution. The point x is a point in x -axis for which we will calculate the probability using both the pdf for the DE distribution($g(x)$ function) and the pdf for the Normal(target distribution , $f(x)$ function). The final but crucial point is the if statement. If $u < \frac{f(x)}{M*g(x)}$ then x is stored as a generated number otherwise we start the procedure from the beginning until we find a point to satisfy this condition.

It is clear that the most important step in this procedure is how we will choose the constant M. We could try different values and choose the optimal one. One can say that the larger the M is, the more accurate the result will be. This is true but we have to also think about the complexity of the algorithm. If we choose a large M, then the acceptance rate will be low which means that many of the random numbers will be rejected and the alogirtm will be computational heavy. Therefore, in order to find the optimal value for M we will find the optimal x for the function $h(x) = f(x)/g(x)$. Fianlly, we set the derivative equal to zero.

$$h(x) = \frac{f(x)}{g(x)} = \frac{\frac{e^{\frac{-x^2}{2}}}{\sqrt{2\pi}}}{\frac{e^{-|x|}}{2}} = \frac{2}{\sqrt{2\pi}} e^{-\frac{x^2}{2}+|x|}$$

We solve that for both $x > 0$ and $x \leq 0$ and we take the derivative equal to zero. So for $x > 0$:

$$h'(x) = \frac{2}{\sqrt{2\pi}} e^{-\frac{x^2}{2}+x}(-x+1) = 0$$

From the above equation only $-x+1$ could be equal to zero($\frac{2}{\sqrt{2\pi}}$ and $e^{-\frac{x^2}{2}+x}$ are always non zero. So $x = 1$.

We do the exact same calculations for $x \leq 0$ and we find that the optimal one is $x = -1$.

The two optimal solutions are $x = 1, x = -1$. In the function $h(x)$ both 1,-1 will give the same output so finally :

$$M = h(1) = \sqrt{\frac{2e}{\pi}} = 1.315489$$

The formula for the expected rejection rate is $1/M$, which is the mean of the random variable $Geom(1/M)$. So in this case the expected rejection rate is 0.7601735. We also put a counter inside the generate function in order to compute the average rejection rate. We want a sample of 2000 and the iterations are 2629. Therefore the average rejection rate is 0.7607455. Their difference is not that important, only 0.000572, which means that our function for the random sample has really satisfying results.

Finally, we plot one histogram for the generated values using our function and one using the built in rnorm function. The plot have similar structure. Our target was to generate a sample for Normal distribution with parameters $\mu = 0, \sigma^2 = 1$. It is obvious that the mean of the plot is 0. We know also that the 68% of the observations in a normal distribution are in the interval $(\mu+, -\sigma)$ so between (-1,1). We can proof that calculating the points which are between this interval in our generated sample. Therefore, 69.3 % is inside this interval. Moreover, one different between the two samples is that the higher value for our sample is 3.7365837 in contrast to the rnorm function which is 3.2640491. In conclusion, the shape of the plot follows a normal distribution and we can say that the function which we use for the random sample works efficient enough.