

# Computer Lab 1

## Computational Statistics

Linköpings Universitet, IDA, Statistik

2019/01/23

---

Kurskod och namn:	732A90 Computational Statistics
Datum:	2019/01/22—2019/01/31 (lab session 23 January 2019)
Delmomentsansvarig:	Krzysztof Bartoszek, Eric Herwin, Sara Johansson
Instruktioner:	This computer laboratory is part of the examination for the Computational Statistics course Create a group report, (that is directly presentable, if you are a presenting group), on the solutions to the lab as a <b>.PDF</b> file. Be concise and do not include unnecessary printouts and figures produced by the software and not required in the assignments. <b>All R code should be included as an appendix into your report.</b> A typical lab report should 2-4 pages of text plus some amount of figures plus appendix with codes. In the report reference <b>ALL</b> consulted sources and disclose <b>ALL</b> collaborations. The report should be handed in via LISAM (or alternatively in case of problems e-mailed to krzysztof.bartoszek@liu.se or Eric Herwin erihe068@student.liu.se or Sara Johansson sarjo775@student.liu.se), by <b>23:59 31 January 2019</b> at latest. Notice there is a final deadline of <b>23:59 1 April 2019</b> after which no submissions nor corrections will be considered and you will have to redo the missing labs next year. The seminar for this lab will take place <b>7 March 2019</b> . The report has to be written in English.

---

## Question 1: Be careful when comparing

Consider the following two R code snippets

```
x1<-1/3;x2<-1/4  
if (x1-x2==1/12){  
print("Subtraction is correct")  
}else{  
print("Subtraction is wrong")  
}
```

and

```
x1<-1;x2<-1/2  
if (x1-x2==1/2){  
print("Subtraction is correct")  
}else{  
print("Subtraction is wrong")  
}
```

1. Check the results of the snippets. Comment what is going on.
2. If there are any problems, suggest improvements.

## Question 2: Derivative

From the definition of a derivative a popular way of computing it at a point  $x$  is to use a small  $\epsilon$  and the formula

$$f'(x) = \frac{f(x + \epsilon) - f(x)}{\epsilon}.$$

1. Write your own R function to calculate the derivative of  $f(x) = x$  in this way with  $\epsilon = 10^{-15}$ .
2. Evaluate your derivative function at  $x = 1$  and  $x = 100000$ .
3. What values did you obtain? What are the true values? Explain the reasons behind the discovered differences.

## Question 3: Variance

A known formula for estimating the variance based on a vector of  $n$  observations is

$$\text{Var}(\vec{x}) = \frac{1}{n-1} \left( \sum_{i=1}^n x_i^2 - \frac{1}{n} \left( \sum_{i=1}^n x_i \right)^2 \right)$$

1. Write your own R function, `myvar`, to estimate the variance in this way.
2. Generate a vector  $x = (x_1, \dots, x_{10000})$  with 10000 random numbers with mean  $10^8$  and variance 1.
3. For each subset  $X_i = \{x_1, \dots, x_i\}$ ,  $i = 1, \dots, 10000$  compute the difference  $Y_i = \text{myvar}(X_i) - \text{var}(X_i)$ , where `var`( $X_i$ ) is the standard variance estimation function in R. Plot the dependence  $Y_i$  on  $i$ . Draw conclusions from this plot. How well does your function work? Can you explain the behaviour?
4. How can you better implement a variance estimator? Find and implement a formula that will give the same results as `var()`?

## Question 4: Linear Algebra

The Excel file “tecator.xls” contains the results of a study aimed to investigate whether a near-infrared absorbance spectrum and the levels of moisture and fat can be used to predict the protein content of samples of meat. For each meat sample the data consists of a 100 channel spectrum of absorbance records and the levels of moisture (water), fat and protein. The absorbance is  $-\log_{10}$  of the transmittance measured by the spectrometer. The moisture, fat and protein are determined by analytic chemistry. The worksheet you need to use is “data” (or file “tecator.csv”). It contains data from 215 samples of finely chopped meat. The aim is to fit a linear regression model that could predict protein content as function of all other variables.

1. Import the data set to R
2. Optimal regression coefficients can be found by solving a system of the type  $\mathbf{A}\vec{\beta} = \vec{b}$  where  $\mathbf{A} = \mathbf{X}^T \mathbf{X}$  and  $\vec{b} = \mathbf{X}^T \vec{y}$ . Compute  $\mathbf{A}$  and  $\vec{b}$  for the given data set. The matrix  $\mathbf{X}$  are the observations of the absorbance records, levels of moisture and fat, while  $\vec{y}$  are the protein levels.
3. Try to solve  $\mathbf{A}\vec{\beta} = \vec{b}$  with default solver `solve()`. What kind of result did you get? How can this result be explained?
4. Check the condition number of the matrix  $\mathbf{A}$  (function `kappa()`) and consider how it is related to your conclusion in step 3.
5. Scale the data set and repeat steps 2–4. How has the result changed and why?

# CompStatsLab1

*Omkar Bhutra*

*7 December 2018*

## Question 1:

**Be careful when comparing**

```
## [1] "subtraction is wrong"
```

Due to underflow the subtraction is displaying the same number although when the digits are increased using options we can see that the number is actually different. Underflow is the loss of significant digits.

```
## [1] "subtraction is correct"
```

```
## [1] "subtraction is correct"
```

Evaluating the results of the 2 snippets we see that in the first occasion we get the wrong print of the if-else statement. The problem lies to the fact that float numbers that have infinite numbers of decimals can't be represented exactly in the binary system in computers due to memory storage limitation. Using `print(x1-x2,digits=16)` and `print(1/12,digits=16)` we will see that the resulting floats are (0.0833333333333331,0.0833333333333333) respectfully and they are not the same causing the condition of unerflow which leads to the failure of the if statement and evaluation of else. We can address this problem using the “`all_equal()`” in the if statement instead of “`==`” to compare the numbers and we will see that the if statement will be executed and the correctly print message will be outputed. The second statement is evaluated correctly and we get the correct print output because  $1/2$  has finite numbers of decimals so we don't have the occurence of underflow here.

## Question 2:

**Derivative**

```
## [1] 1.1102230246251565
```

```
## [1] 0
```

The value of the derivative for when  $x=1$  is 1.11022 while the value obtained for the derivative when  $x=100000$  is 0. Looking at the equation algebraically it seems that the answer should be 0. But in the first case, when  $x=1$  the addition of a small value epsilon retains its effect compared to the 2nd case and hence produces a different result than the expected value of 0.

The true value for the function using the function  $f(x) = x$  is  $f'(x) = \frac{f(x+\epsilon)-f(x)}{\epsilon} = \frac{(x+\epsilon)-x}{\epsilon} = 1$  is always constant with value 1. Regarding the result of the derivative function we see that for  $x = 100000$  R doesn't take into account the decimals after a specific number of x and rounds the number to the nearest integer which is 100000 due to underflow occurrence so the numeretor of the derivative formula becomes 0 leading finally to 0. When instead  $x = 1$  the numerator evaluated is 1.1102230246251565e-15 and the devision with epsilon  $10^{-15}$  is just discards the last 15 decimals resulting 1.1102230246251565.

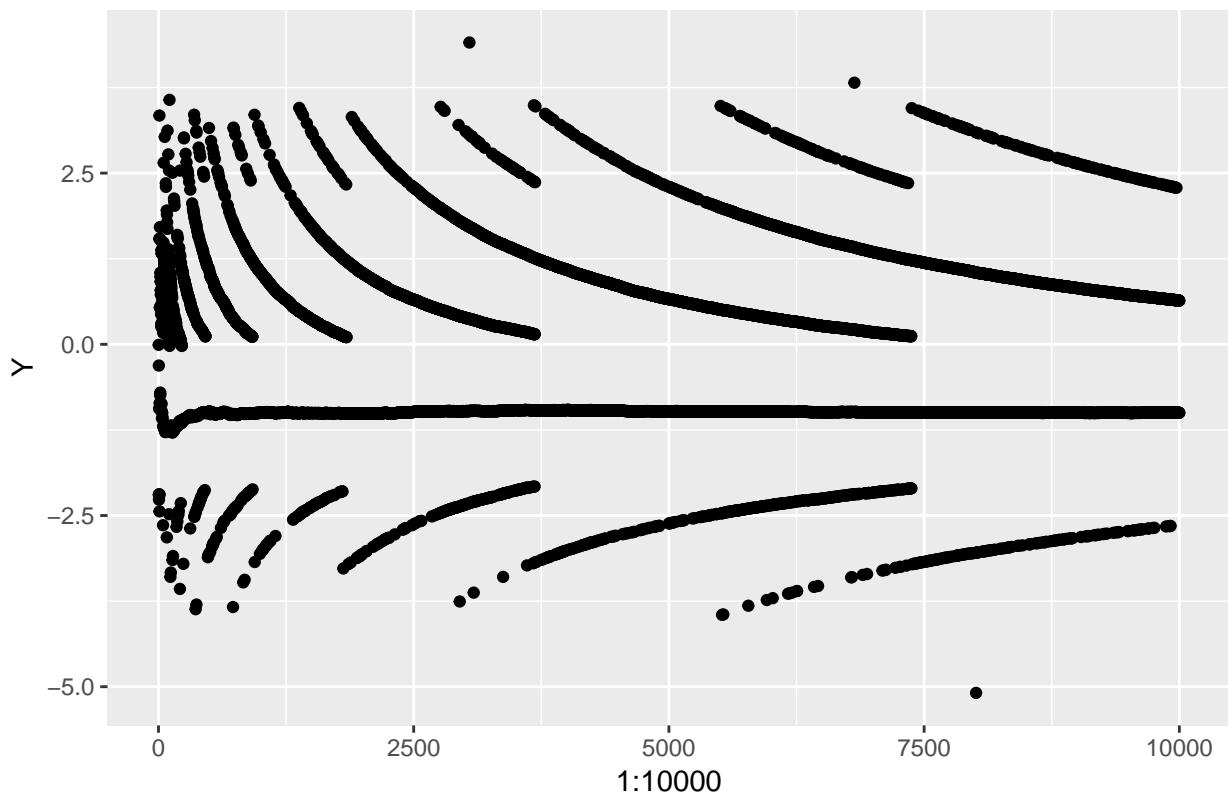
### Question 3:

#### Variance

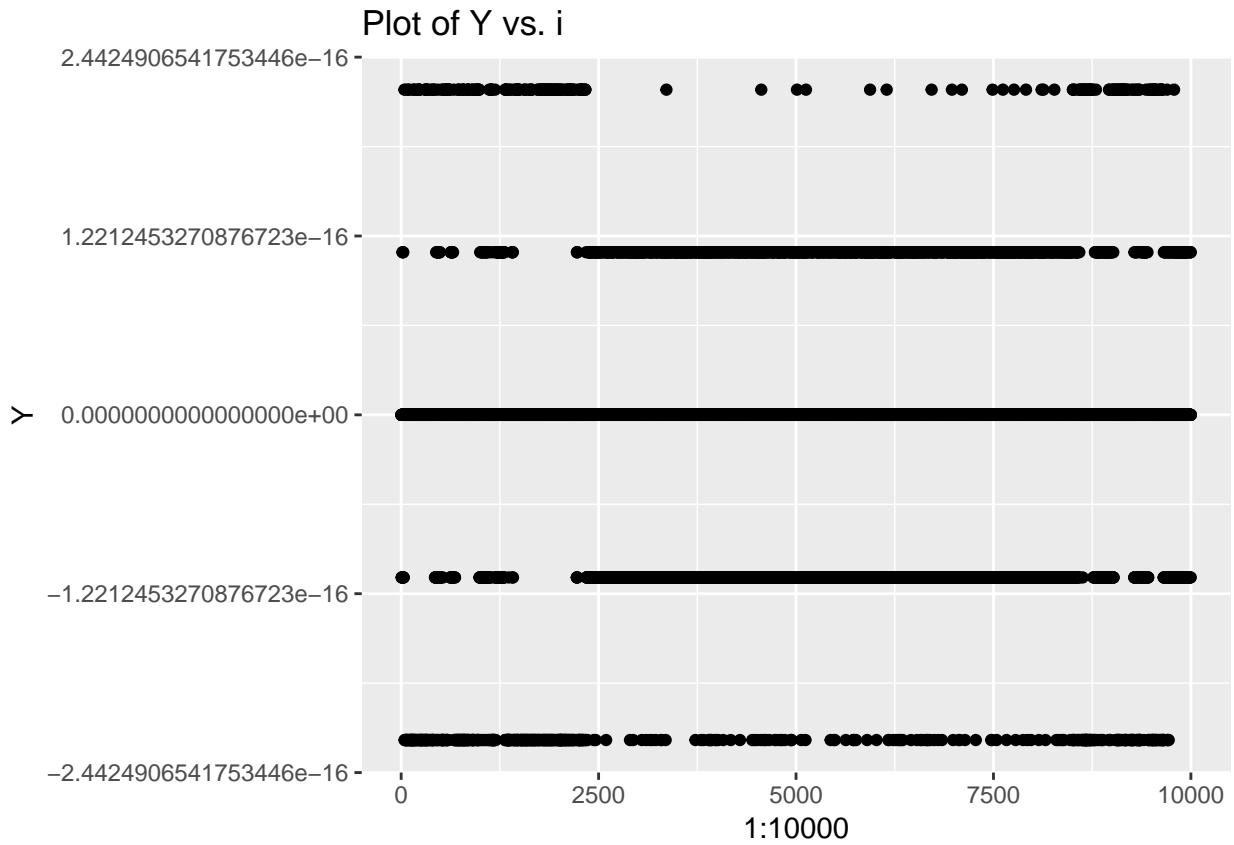
```
## [1] 1.6385638563856386
```

The plot above shows the dependence  $Y_i$  on  $i$  with the formula  $Var(x) = \frac{\sum_{i=1}^n (x_i - \mu)^2}{n-1}$  where  $\mu$  is the mean. Using the new formula where we center the points around the mean we see that we have an improvement in the range of the errors and the deviation of the errors is steady and we can see an upper and a lower band with few errors lie beyond these linear bands represented with red in the plot. Also we can observe that the range of the errors is much smaller with means the formula used almost as good as the var() basic function in R.

Plot of Y vs. i



```
## [1] 0.99969989923081803
```



The function does not perform as well as expected due to the numerical precision of the expression myvar(Xi)-var(Xi) which shows us negative values on most occurrences.

#### Question 4:

##### Linear Algebra

```
## [1] 1157834236871692.2
```

Error in solve.default(A, b) : system is computationally singular: reciprocal condition number = 7.13971e-17

Printing the number of kappa for the value of A matrix we see that is very big and that implies that the matrix is said to be ill-conditioned a very small change in matrix A will cause a large error in b and makes the solution unstable.

```
## [,1]
## Channel1    -110.64200663177356887
## Channel2   -221.18999213628123357
## Channel3    378.00670489934344687
## Channel4   -129.70382900682116656
## Channel5    413.41802724878726849
## Channel6   -79.75199462292130193
## Channel7   -203.00600103836293897
## Channel8     82.79496481288938980
## Channel9   -132.38107625535101874
## Channel10   255.82331130982933587
```

```

## Channel11  -328.60850197387361504
## Channel12  -304.14980672129559025
## Channel13  624.12672247994066765
## Channel14  -298.89902984730480284
## Channel15   40.74336715520891516
## Channel16  -257.53594209754874100
## Channel17  169.23891908084701186
## Channel18  296.66293885907538197
## Channel19  -325.06601012322818178
## Channel20   -3.00774341513279264
## Channel21  554.56043514869395494
## Channel22  -1366.02819464972321839
## Channel23  1860.35645048260312251
## Channel24  -1416.13210988432069826
## Channel25  631.83966192630055048
## Channel26  -112.04395491771877857
## Channel27   17.01671931621323708
## Channel28  -228.93063252299120336
## Channel29  444.27242587767221949
## Channel30  -597.38053691993104621
## Channel31  438.14844153976412144
## Channel32  315.03940879430871291
## Channel33  -349.8143768365787779
## Channel34  -285.91097574921798241
## Channel35  418.58105049764253636
## Channel36  -79.10682360375271571
## Channel37  -305.94133876342527856
## Channel38  284.25434730174231390
## Channel39  -435.56578629077682763
## Channel40  819.74862823138187196
## Channel41  -885.00733142452941138
## Channel42  324.58934112061319865
## Channel43  524.58956696461768843
## Channel44  -583.44189030616257696
## Channel45  -140.17097449229814288
## Channel46  577.23617438118196787
## Channel47  -294.26844675559453890
## Channel48  -68.07512752921009280
## Channel49  -90.49228410711393167
## Channel50  404.14395235715244326
## Channel51  -698.99905269889643478
## Channel52  1258.88337827146801828
## Channel53  -1672.73084007547663532
## Channel54  1486.22991720257186898
## Channel55  -812.36134431546076939
## Channel56  192.49547990397724107
## Channel57  -32.91204662098924416
## Channel58   7.37525455331103430
## Channel59  -88.69071417608546426
## Channel60  344.87690207084136773
## Channel61  -454.35186074457743644
## Channel62  447.62052960726805395
## Channel63  -197.41868472109501909
## Channel64  222.33707920152659199

```

```

## Channel165 -399.25583177909453525
## Channel166 364.86655737276095124
## Channel167 -367.16148297452463112
## Channel168 243.92206215756519327
## Channel169 -76.29483445491032967
## Channel170 -318.19160711413701392
## Channel171 327.66533461201169075
## Channel172 -178.52315275400727046
## Channel173 119.18564986588171450
## Channel174 445.11500447600673169
## Channel175 -20.01273187169612910
## Channel176 -642.75099614710870810
## Channel177 369.48095078598106511
## Channel178 -74.90113656891863059
## Channel179 -23.48543216535520983
## Channel180 -676.86153344610886506
## Channel181 1013.45380290573928050
## Channel182 -889.76231887539347554
## Channel183 403.00656326222281223
## Channel184 424.08483028785201441
## Channel185 -801.09561546783777430
## Channel186 655.01342015748275571
## Channel187 659.18297852899979716
## Channel188 -2150.83256466095554060
## Channel189 1671.80888532636413402
## Channel190 298.69770682030031139
## Channel191 -332.17277624942408920
## Channel192 -487.36897587493234596
## Channel193 278.62773677083617940
## Channel194 201.66273526775202640
## Channel195 -609.50814557014871298
## Channel196 565.28517886262272896
## Channel197 -133.34075951392054549
## Channel198 -368.00872501373430623
## Channel199 238.20159678039942719
## Channel100 24.64181878308056639
## Fat -1.66664028405493303
## Moisture -0.93410994774249811

```

This happens because the tolerance returned is larger than the default threshold set by the function solve (argument tolerance) so an error returned and we cannot get a solution. The torque is related to condition number by the function  $\text{tolerance} = \frac{1}{\text{conditionnumber}}$  so in our case  $\text{tolerance} = \frac{1}{\kappa(A)} = 7.425326e-16$  and it is bigger than the threshold of  $7.425326e-17$  that is set by solve function as we see in the printed error resulting the end of execution of the function. Using the scaled data we were able to solve the linear system and get coefficients for every feature value. Printing the number of kappa again we can see that is still high but much less than the previous used with the unscaled data and we were able to solve the linear system and get coefficient values.

When we scale the data we see that the linear system did not get any better or worse the linear dependences of the column features are still present but we manage to make the value of condition number smaller with scaling. This is happening because If we look at the definition of the condition number  $k(A) = \|A\| * \|A^{-1}\|$  and just by making the range of the columns smaller the magnitude got smaller leading to a smaller value of condition number which is below threshold value of solve function and we manage to get the solution. The tolerance now is  $\text{tolerance} = \frac{1}{\kappa(A)} = \frac{1}{490471518993} = 2.038854e-12$  which is smaller than the default

$7.425326e - 17$  set by solve so now we are able to get a solution.

## Appendix

```
knitr::opts_chunk$set(echo = TRUE)
library(dplyr)
library(plotly)
library(ggplot2)
library(xlsx)
library(readxl)
library(boot)
library(kableExtra)
library(knitr)
library(testthat)
options(digits=22)
x1<-1/3;x2<-1/4
if(x1-x2==1/12){
  print("subtraction is correct")
}else{
  print("subtraction is wrong")
}
x1<-1/3;x2<-1/4
if(all.equal((x1-x2),(1/12))){
  print("subtraction is correct")
}else{
  print("subtraction is wrong")
}
x1<-1;x2<-1/2
if(x1-x2==1/2){
  print("subtraction is correct")
}else{
  print("subtraction is wrong")
}
derivative<-function(x){
  f<-function(x){
    return(x)
  }
  epsilon<-10^-15
  x<-(f(x+epsilon)-f(x))/epsilon
  return(x)
}

derivative(x=1)

derivative(x=100000)

set.seed(12345)
myvar<-function(x){
  n=length(x)
  var<-(1/(n-1))*(sum(x^2)-((1/n)*(sum(x)^2)))
  return(var)
}
```

```

x<-rnorm(n=10000,mean=10^8,sd=sqrt(1))
myvar(x)
Y <- c()
for (i in 1:10000){
  options(digits = 22)
  Y[i] <- myvar(x[1:i])-var(x[1:i])
}

p1<-ggplot() + geom_point(aes(1:10000,Y)) + labs(title="Plot of Y vs. i")
p1
set.seed(12345)
varfun <- function(x){
  vari <- (1/(length(x) - 1)) * sum((x - mean(x))^2)
  return(vari)
}

x<-rnorm(n=10000,mean=10^8,sd=sqrt(1))
varfun(x)

Y <- c()
for (i in 1:10000){
  Y[i] <- varfun(x[1:i])-var(x[1:i])
}

p2<-ggplot() + geom_point(aes(1:10000,Y)) + labs(title="Plot of Y vs. i")
p2
tecator = read_excel("tecator.xls",sheet ="data" )
X<-as.matrix(tecator[,c(2:102,104)])
Y<-as.matrix(tecator[,c(103)])
A<-t(X)%*%X
b<-t(X)%*%Y
kappa(A)
#solve(A,b)
tecatorscale <- as.matrix(scale(tecator))
Xscale <- as.matrix(tecatorscale[,-c(1,103)])
yscale <- as.matrix(tecatorscale[,103])
Ascale <- t(Xscale)%*%Xscale
bscale <- t(Xscale)%*%yscale
solve(Ascale,bscale)

```

# Computer Lab 2

## Computational Statistics

Linköpings Universitet, IDA, Statistik

2019/01/25

---

Kurskod och namn:	732A90 Computational Statistics
Datum:	2019/01/24—2019/02/07 (lab session 25 January 2019)
Delmomentsansvarig:	Krzysztof Bartoszek and Eric Herwin
Instruktioner:	<p>This computer laboratory is part of the examination for the Computational Statistics course</p> <p>Create a group report, (that is directly presentable, if you are a presenting group), on the solutions to the lab as a <b>.PDF</b> file.</p> <p>Be concise and do not include unnecessary printouts and figures produced by the software and not required in the assignments.</p> <p><b>All R code should be included as an appendix into your report.</b></p> <p>A typical lab report should 2-4 pages of text plus some amount of figures plus appendix with codes.</p> <p>In the report reference <b>ALL</b> consulted sources and disclose <b>ALL</b> collaborations.</p> <p>The report should be handed in via LISAM</p> <p>(or alternatively in case of problems e-mailed to krzysztof.bartoszek@liu.se or Eric Herwin erihe068@student.liu.se or Sara Johansson sarjo775@student.liu.se), by <b>23:59 7 February 2019</b> at latest.</p> <p>Notice there is a final deadline of <b>23:59 1 April 2019</b> after which no submissions nor corrections will be considered and you will have to redo the missing labs next year.</p> <p>The seminar for this lab will take place <b>7 March 2019</b>.</p> <p>The report has to be written in English.</p>

---

## Question 1: Optimizing a model parameter

The file `mortality_rate.csv` contains information about mortality rates of the fruit flies during a certain period.

1. Import this file to R and add one more variable `LMR` to the data which is the natural logarithm of `Rate`. Afterwards, divide the data into training and test sets by using the following code:

```
n=dim(data)[1]  
set.seed(123456)  
id=sample(1:n, floor(n*0.5))  
train=data[id,]  
test=data[-id,]
```

2. Write your own function `myMSE()` that for given parameters  $\lambda$  and list `pars` containing vectors `X`, `Y`, `Xtest`, `Ytest` fits a LOESS model with response `Y` and predictor `X` using `loess()` function with penalty  $\lambda$  (parameter `enp.target` in `loess()`) and then predicts the model for `Xtest`. The function should compute the predictive MSE, print it and return as a result. The predictive MSE is the mean square error of the prediction on the testing data. It is defined by the following Equation (for you to implement):

$$\text{predictive MSE} = \frac{1}{\text{length(test)}} \sum_{\text{ith element in test set}} (Y_{\text{test}}[i] - f_{\text{Ypred}}(X[i]))^2,$$

where `fYpred(X[i])` is the predicted value of `Y` if `X` is `X[i]`. Read on R's functions for prediction so that you do not have to implement it yourself.

3. Use a simple approach: use function `myMSE()`, training and test sets with response `LMR` and predictor `Day` and the following  $\lambda$  values to estimate the predictive MSE values:  $\lambda = 0.1, 0.2, \dots, 40$
4. Create a plot of the MSE values versus  $\lambda$  and comment on which  $\lambda$  value is optimal. How many evaluations of `myMSE()` were required (read `?optimize`) to find this value?
5. Use `optimize()` function for the same purpose, specify range for search  $[0.1, 40]$  and the accuracy 0.01. Have the function managed to find the optimal MSE value? How many `myMSE()` function evaluations were required? Compare to step 4.
6. Use `optim()` function and BFGS method with starting point  $\lambda = 35$  to find the optimal  $\lambda$  value. How many `myMSE()` function evaluations were required (read `?optim`)? Compare the results you obtained with the results from step 5 and make conclusions.

## Question 2: Maximizing likelihood

The file `data.RData` contains a sample from normal distribution with some parameters  $\mu, \sigma$ . For this question read `?optim` in detail.

1. Load the data to R environment.
2. Write down the log-likelihood function for 100 observations and derive maximum likelihood estimators for  $\mu, \sigma$  analytically by setting partial derivatives to zero. Use the derived formulae to obtain parameter estimates for the loaded data.
3. Optimize the minus log-likelihood function with initial parameters  $\mu = 0, \sigma = 1$ . Try both Conjugate Gradient method (described in the presentation handout) and BFGS (discussed in the lecture) algorithm with gradient specified and without. Why it is a bad idea to maximize likelihood rather than maximizing log-likelihood?
4. Did the algorithms converge in all cases? What were the optimal values of parameters and how many function and gradient evaluations were required for algorithms to converge? Which settings would you recommend?

# Lab2 Computational Statistics

Andreas Christopoulos Charitos (andch552) Omkar Bhutra (omkbh878)

27 jan 2019

## Question 1

1

```
#importing and diving data
mortality<-read.csv2("mortality_rate.csv")
mortality$LMR<-log(mortality$Rate)

n=dim(mortality)[1]
set.seed(123456)
id=sample(1:n , floor(n*0.5))
train=mortality[id, ]
test=mortality[-id, ]

#defining the function "myMSE"

myMSE<-function(lambda,pars,iterCounter = T){
  model<-loess(pars$Y~pars$X,emp.target = lambda)
  preds<-predict(model,newdata=pars$X_test)
  mse<-sum((pars$Y_test-preds)^2)/length(pars$Y_test)

  # If we want a iteration counter
  if(iterCounter){
    if(!exists("iterForMyMSE")){
      # Control if the variable exists in the global environment,
      # if not, create a variable and set the value to 1. This
      # would be the case for the first iteration
      # We will call the variable 'iterForMyMSE'
      assign("iterForMyMSE",
            value = 1,
            globalenv())
    } else {
      # This part is for the 2nd and the subsequent iterations.
      # Starting of with obtaining the current iteration number
      # and then overwrite the current value by the incremental
      # increase of the current value
      currentNr <- get("iterForMyMSE")
      assign("iterForMyMSE",
            value = currentNr + 1,
            globalenv())
    }
  }
  return(mse)
}

set.seed(123456)
steps=seq(0.1,40,0.1)
```

```

mses<-double(length(steps))

mypars<-list(X=train$Day,Y=train$LMR,X_test=test$Day,Y_test=test$LMR)

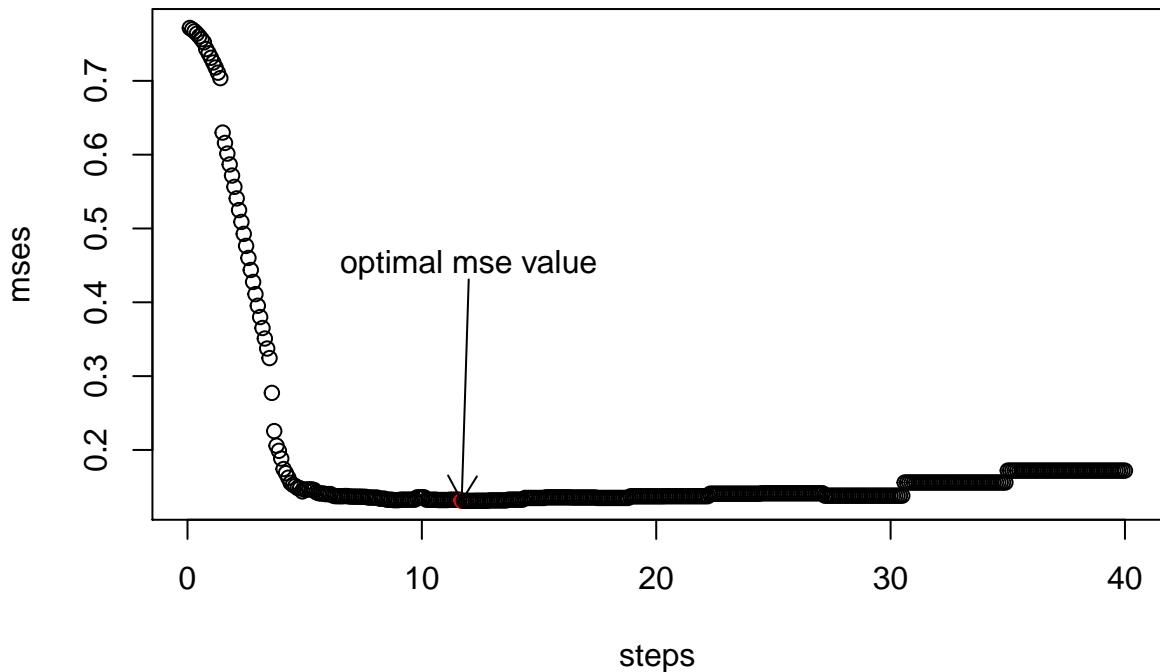
for(i in steps){
  mses[which(i==steps)]<-myMSE(i,mypars)
}

optimal_l=steps[which.min(mses)]
optimal_mse=min(mses)

plot(x=steps,y= mses,main = "Mse vs lambda",
      col=ifelse(steps==optimal_l,"red","black"))
arrows(optimal_l+0.3, optimal_mse+0.3, x1 = optimal_l, y1 = optimal_mse,
       length = 0.15, angle = 30)
text(optimal_l+0.3,optimal_mse+0.32,labels=c("optimal mse value"))

```

**Mse vs lambda**



```

iters1=iterForMyMSE

results1=list("par"=optimal_l,"value"=optimal_mse)

cat("The number of iterations using brute force are :", iters1,"\n",
    "The results from the brute force are :\n")

## The number of iterations using brute force are : 400

```

```

## The results from the brute force are :
results1

## $par
## [1] 11.7
##
## $value
## [1] 0.131047

5

set.seed(123456)
remove("iterForMyMSE")
xmin <- optimize(myMSE,pars=mypars,
                  interval=c(0.1,40),maximum = FALSE,tol=0.01)
iters2=iterForMyMSE
cat("The number of iterations using lambda in [0.1,40] and accuracy 0:01 are :", iters2,"\n",
    "The output of optimize function is : \n")

## The number of iterations using lambda in [0.1,40] and accuracy 0:01 are : 18
## The output of optimize function is :

## $minimum
## [1] 10.69361
##
## $objective
## [1] 0.1321441

```

We can see comparing the results with the previous step that the number of iterations needed for the algorithm to converge decreased dramatically from 400 to only 18.

## 6

```

set.seed(123456)
remove("iterForMyMSE")
xmin1=optim(c(35), myMSE,pars=mypars,
            method = c( "BFGS"))

iters3=iterForMyMSE
cat("The new number of iterations using BFGS algorithm and lambda = 35 are :", iters3,"\n",
    "The output of optimize function is : \n")

## The new number of iterations using BFGS algorithm and lambda = 35 are : 3
## The output of optimize function is :

## $par
## [1] 35
##
## $value
## [1] 0.1719996
##
## $counts
## function gradient
##          1          1
##
```

```

## $convergence
## [1] 0
##
## $message
## NULL

```

Comparing again the number of iterations with the previous step we can see that the iterations decreased from 18 to only 3 using BFGS.

**The table below summarises the results of the 3 methods used**

```

library(knitr)

sumtable<-data.frame("Brute Force Method"=c(optimal_l,optimal_mse,iters1),
                      "Optimize function"=c(xmin$minimum,xmin$objective,iters2),
                      "Optim function BFGS"=c(xmin1$par,xmin1$value,iters3))
rownames(sumtable)<-c("lambda","mse","iters")

kable(sumtable)

```

	Brute.Force.Method	Optimize.function	Optim.function.BFGS
lambda	11.700000	10.6936107	35.0000000
mse	0.131047	0.1321441	0.1719996
iters	400.000000	18.0000000	3.0000000

As we can see from the table above the BFGS algorithm is able to converge with only 3 iterations which are lower compared to the other 2 methods but with a little higher mse value.

## Question 2

2

### Maximum Likelihood estimation

The probability function is given by the formula

$$f(x) = \frac{1}{\sqrt{(2\pi\sigma^2)}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right)$$

Assuming that the observations from the sample are i.i.d. the likelihood formula is given by

$$f(x_1, x_2, \dots, x_n | \mu, \sigma^2) = \prod_{j=1}^n (2\pi\sigma^2)^{-(n/2)} \exp\left(-\frac{(x_j - \mu)^2}{2\sigma^2}\right)$$

which is equivalent to :

$$f(x_1, x_2, \dots, x_n | \mu, \sigma^2) = (2\pi\sigma^2)^{-(n/2)} \exp\left(-\frac{\sum_{j=1}^n (x_j - \mu)^2}{2\sigma^2}\right)$$

The loglikelihood function is then

$$\begin{aligned}
L &= \ln[(2\pi\sigma^2)^{(-n/2)} \exp(-(\frac{1}{2\sigma^2}) \sum_{j=1}^n (x_j - \mu)^2)] = \\
&\ln((2\pi\sigma^2)^{(-n/2)}) + \ln(\exp(-(\frac{1}{2\sigma^2}) \sum_{j=1}^n (x_j - \mu)^2)) = \\
&-\frac{n}{2} \ln(2\pi\sigma^2) - (\frac{1}{2\sigma^2}) \sum_{j=1}^n (x_j - \mu)^2 = \\
&-\frac{n}{2} \ln(2\pi) - \frac{n}{2} \ln(\sigma^2) - (\frac{1}{2\sigma^2}) \sum_{j=1}^n (x_j - \mu)^2
\end{aligned}$$

Getting the derivative with respect to  $\mu$  to we get :

$$\begin{aligned}
\frac{\partial L}{\partial \mu} &=> -\frac{1}{2\sigma^2} (\sum_{j=1}^n (x_j^2 - 2x_j\mu + \mu^2))' => \\
&-\frac{1}{2\sigma^2} (\sum_{j=1}^n x_j^2 - 2 \sum_{j=1}^n x_j\mu + n\mu^2)' => \\
&-\frac{1}{2\sigma^2} (-2 \sum_{j=1}^n x_j + 2n\mu) => -\frac{1}{2\sigma^2} (\sum_{j=1}^n x_j - n\mu)(-2) \quad (e.1)
\end{aligned}$$

Setting (e.1) equal to zero we get

$$\frac{\partial L}{\partial \mu} = 0 \Rightarrow \mu = \frac{\sum_{j=1}^n x_j}{n}$$

Which is the estimator for the parameter  $\hat{\mu}$

Getting the derivative with respect to  $\sigma$  we get :

$$\frac{\partial L}{\partial \sigma} => -\frac{n}{2} \frac{1}{\sigma^2} 2\sigma - \frac{1}{2} \sum_{j=1}^n (x_j - \mu)^2 \frac{\partial L}{\partial \sigma} (\sigma^2)^{-1} => -\frac{n}{\sigma} - \frac{1}{2} \sum_{j=1}^n (x_j - \mu)^2 \frac{-2}{(\sigma^3)} => -\frac{n}{\sigma} + \sum_{j=1}^n (x_j - \mu)^2 \frac{1}{(\sigma^3)} \quad (e.2)$$

Setting (e.2) equal to zero we get

$$\frac{\partial L}{\partial \sigma} = 0 \Rightarrow \sum_{j=1}^n (x_j - \mu)^2 = \frac{n\sigma^3}{\sigma} \Rightarrow \sigma^2 = \frac{\sum_{j=1}^n (x_j - \mu)^2}{n} \Rightarrow \sigma = \sqrt{\frac{\sum_{j=1}^n (x_j - \mu)^2}{n}}$$

We can use the obtained formulas to obtain the mean  $\hat{\mu}$  and variance  $\hat{\sigma}$  analytically.

```
#load data
load("data.RData")

#make estimators function
estimators<-function(data){
  n<-length(data)
  mean<-sum(data)/n
  sigma<-sum((data-mean)^2)/n
  return(c("mean"=mean, "sd"=sqrt(sigma)))
}
```

```

}

res=estimators(data)

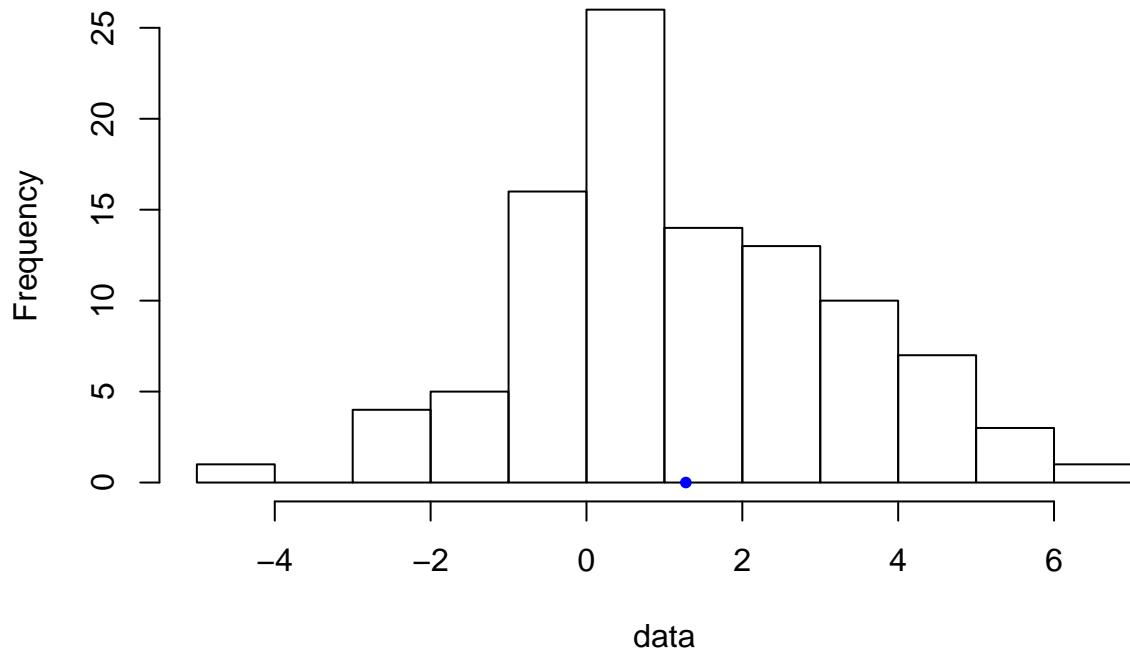
cat("=====\\n",
    "The obtained mean is :",res["mean"],"and the sd is :",res["sd"])

## =====
## The obtained mean is : 1.275528 and the sd is : 2.005976

hist(data)
points(res["mean"],0,col="blue",pch=20,
      main="Histogram of data")

```

**Histogram of data**



The above histogram shows the distribution of our data and the blue point is indicating the mean value.

### 3

First we optimize minus loglikelihood without specifying gradient

```

loglike<-function(args,x){

  n<-length(x)
  logminus<- (-n*log(2*pi*args[2]^2)/2)-(sum((args[1]-x)^2)/(2*args[2]^2))
  logminus<- -1*logminus
  return(logminus)
}

```

```

}

#with BFGS
myres1<-optim(c(0,1),fn=loglike,x=data,method = "BFGS")

#with Conjugate Gradient
myres2<-optim(c(0,1),fn=loglike,x=data,method = "CG")

cat("=====\\n",
  "The output of optim with BFGS is :\\n")

## =====
## The output of optim with BFGS is :
myres1

## $par
## [1] 1.275528 2.005977
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##        41      15
##
## $convergence
## [1] 0
##
## $message
## NULL
cat("\\n")

cat("=====\\n",
  "The output of optim with Conjugate Gradient is :\\n")

## =====
## The output of optim with Conjugate Gradient is :
myres2

## $par
## [1] 1.275528 2.005977
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##        208      35
##
## $convergence
## [1] 0

```

```
##  
## $message  
## NULL
```

## Second we optimize minus loglikelihood specifying gradient

The gradient with respect to  $\mu$  is given by the formula (e.1) we calculated earlier  $\frac{\partial L}{\partial \mu} = \frac{\sum_{j=1}^n x_j - n\mu}{\sigma^2}$  and the derivative with respect to  $\sigma$  is given by the formula (e.2)  $\frac{\partial L}{\partial \sigma} = -\frac{n}{\sigma} + \frac{\sum_{j=1}^n (x_j - \mu)}{\sigma^3}$

```
gradient<-function(args,x){  
  n<-length(x)  
  gr_mu<- (sum(x)-n*args[1])/(args[2]^2)  
  
  gr_sigma<- sum((x-args[1])^2)/args[2]^3-n/args[2]  
  
  return(c(-gr_mu,-gr_sigma))  
}  
  
#with BFGS  
myres3<-optim(c(0,1),fn=loglike,gr=gradient,x=data,method = "BFGS")  
  
#with Conjugate Gradient  
myres4<-optim(c(0,1),fn=loglike,gr=gradient,x=data,method = "CG")  
  
  
cat("=====\\n",  
    "The output of optim with BFGS with gradient is :\\n")  
  
## =====  
## The output of optim with BFGS with gradient is :  
myres3  
  
## $par  
## [1] 1.275528 2.005977  
##  
## $value  
## [1] 211.5069  
##  
## $counts  
## function gradient  
##      39      15  
##  
## $convergence  
## [1] 0  
##  
## $message  
## NULL  
cat("\\n")  
  
cat("=====\\n",  
    "The output of optim with Conjugate Gradient with gradient is :\\n")  
  
## =====  
## The output of optim with Conjugate Gradient with gradient is :
```

```

myres4

## $par
## [1] 1.275528 2.005976
##
## $value
## [1] 211.5069
##
## $counts
## function gradient
##      53      17
##
## $convergence
## [1] 0
##
## $message
## NULL

```

As we can see loglikelihood is monotonically increasing function and is has the same relations of order as the likelihood  $p(x|\theta_1) > p(x|\theta_2) \Leftrightarrow \ln(p(x|\theta_1)) > \ln(p(x|\theta_2))$  so maximizing likelihood is equivalent to maximizing log likelihood. The reason why we use  $\ln(x)$  is for computational convenience as we see in the calculations of loglikelihood we did before. Using  $\ln()$  we were able to discard the exponent and also use the  $\ln(ab) = \ln(a) + \ln(b)$  we manage to replace multiplication with summation which is more convenient and we are able to maximize just by setting the derivatives to 0.

## 4

```

library(knitr)

sumdat<-data.frame("BFGS_without_gr"=c(myres1$par[1],myres1$par[2],
                                         myres1$counts[1],myres1$counts[2],myres1$convergence),
                     "CG_without_gr"=c(myres2$par[1],myres2$par[2],
                                         myres2$counts[1],myres2$counts[2],myres2$convergence),
                     "BFGS_with_gr"=c(myres3$par[1],myres3$par[2],
                                         myres3$counts[1],myres3$counts[2],myres3$convergence),
                     "CG_with_gr"=c(myres4$par[1],myres4$par[2],
                                         myres4$counts[1],myres4$counts[2],myres4$convergence)
                     )

rownames(sumdat)<-c("mean","sd","iterations function","iterations gradient","convergence")
sumdat[5,]<-ifelse(sumdat[5,]==0,"yes","no")

kable(sumdat)

```

	BFGS_without_gr	CG_without_gr	BFGS_with_gr	CG_with_gr
mean	1.27552755151932	1.27552771909709	1.27552755040258	1.27552759112531
sd	2.00597696486639	2.00597650338868	2.00597654945241	2.00597647249389
iterations function	41	208	39	53
iterations gradient	15	35	15	17
convergence	yes	yes	yes	yes

The table provides summary information using the 2 algorithms (“BFGS”,“CG”) with and without specifying gradient function in optim function. All the algorithms converge as we can see and the results are almost

identical regarding the mean and the sd. The settings that we would recommend are using BFGS algorithm with gradient because as we can see the number of iterations for both function and gradient are lower compared with the other settings.

Area of maj. func. =

Rej. rate is the prob. that a pt. is outside target dist.

After being maximized  $\rightarrow$

$$\text{Area } f(x) = c$$

Exp. Rej. rate

= Rej. regions

Total region

$$\rightarrow \text{Area } f(x) = 1$$

$$\Rightarrow \frac{c-1}{c}$$

~~Ques 4~~

Lab 2-

Q2)

$$P(x|\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

when no particular distribution is mentioned, take normal distr.

Likelihood:

$$L(\mu, \sigma|x) = \prod_{n=1}^N P(x_n|\mu, \sigma)$$
$$= \prod_{n=1}^N \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_n-\mu)^2}{2\sigma^2}}$$

Log Likelihood:

$$\log L(\mu, \sigma|x) = \sum_{n=1}^N (\log c_1 - \log(2\pi\sigma^2)) - \frac{\sum_{n=1}^N (x_n - \mu)^2}{2\sigma^2}$$
$$= -\frac{1}{2} \sum_{n=1}^N \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \sum_{n=1}^N (x_n - \mu)^2 \quad \text{--- (1)}$$

Partially  $\Rightarrow$  diff. (1) w.r.t  $\mu$  & set to 0

$$\frac{\partial \log L(\mu, \sigma|x)}{\partial \mu} = -\frac{1}{2\sigma^2} \sum_{n=1}^N (x_n - \mu) \rightarrow \sum_{n=1}^N \frac{(x_n - \mu)}{\sigma^2} = 0$$

Gradient  
of  $\mu$

$$\sum_{n=1}^{\infty} (x_n - \mu) = 0$$

$$\sum_{n=1}^{\infty} x_n = n\mu$$

$$\therefore \boxed{\mu = \frac{\sum_{n=1}^{\infty} x_n}{n}}$$

Partially diff. ① w.r.t.  $\sigma$  & set to 0,

$$\frac{\partial \log L(\mu, \sigma | x)}{\partial \sigma} = -\frac{1}{2} \cdot n \cancel{\left( \frac{1}{2\pi\sigma^2} \right)} \cancel{\left( \frac{x - \mu}{\sigma} \right)} + \frac{1}{2} \frac{n}{\sigma^3} \sum_{n=1}^{\infty} (x_n - \mu)^2$$

$$0 = \frac{-n}{\sigma} + \underbrace{\sum_{n=1}^{\infty} \frac{(x_n - \mu)^2}{\sigma^3}}_{\text{Gradient of } \sigma} \quad \text{Gradient of } \sigma$$

$$0 = -n\sigma^2 + \sum_{n=1}^{\infty} (x_n - \mu)^2$$

$$\sigma^2 = \frac{\sum_{n=1}^{\infty} (x_n - \mu)^2}{n}$$

$$\boxed{\sigma = \sqrt{\frac{\sum_{n=1}^{\infty} (x_n - \mu)^2}{n}}}$$

# Computational Statistics - Lab 3

*Yusur Al-Mter (yusal621), Lakshidaa Saigiridharan (laksa656)*

*2/8/2019*

## Contents

<b>QUESTION 1 : Cluster sampling</b>	<b>1</b>
TASK 1.1: . . . . .	1
TASK 1.2: . . . . .	1
TASK 1.3: . . . . .	2
TASK 1.4 . . . . .	2
TASK 1.5 . . . . .	3
<b>QUESTION 2: Different distributions</b>	<b>5</b>
TASK 2.1 . . . . .	5
TASK 2.2 . . . . .	7
<b>Appendix</b>	<b>10</b>

## 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 1.1:

Import necessary information to R.

```
# TASK 1.1  
  
# Importing the file  
data <- read.csv2("population.csv")
```

### TASK 1.2:

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).

```
# TASK 1.2  
  
# Function which uses a uniform random number generator to select a city from a list  
city_selection <- function(data){  
    data$cumsum <- cumsum(data$Population) / sum(data$Population)  
    selected <- which(runif(1) <= data$cumsum)[1]
```

```

city_selected <- data$Municipality[selected]
as.numeric(rownames(data[selected,]))
}

```

### TASK 1.3:

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.

```

# TASK 1.3

select <- data.frame()

# List of all cities stored in data1
data1 <- data

# Selecting 20 cities
for(i in 1:20){
  selected <- which(city_selection(data1) == as.numeric(rownames(data)))
  select <- rbind(select, data1[selected,])
  # Removing the selected city from the list
  data1 <- data1[-c(selected),]
}

```

### TASK 1.4

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

```

# TASK 1.4

# Dataframe of 20 selected cities
cat("Selected cities : \n")

## Selected cities :
select$Municipality

## [1] Borgholm          Sollefte\345      Osby           Torsby
## [5] Sigtuna           Bjurholm         Hjo            Gnesta
## [9] Trosa             Ludvika          Mullsj\366    Tran\345s
## [13] Haninge          S\366dert\344lje Arvidsjaur   Enk\366ping

```

```
## [17] Oskarshamn      Flen          Gullsp\345ng      Sval\366v
## 290 Levels: Ale Alings\345s Alvesta Aneby Arboga Arjeplog ... \326vertorne\345
```

The selected cities along with their respective size is shown below :

```
# Selected cities along with their respective sizes
city_size = data.frame(City = select$Municipality, Size = select$Population)

kable(city_size, booktabs = T) %>%
kable_styling(latex_option = "striped")
```

City	Size
Borgholm	10806
Sollefte<e5>	20442
Osby	12656
Torsby	12508
Sigtuna	39219
Bjurholm	2500
Hjo	8859
Gnesta	10318
Trosa	11446
Ludvika	25650
Mullsj<f6>	7027
Tran<e5>s	18043
Haninge	76237
S<f6>dert<e4>lje	85270
Arvidsjaur	6622
Enk<f6>ping	39360
Oskarshamn	26232
Flen	16139
Gullsp<e5>ng	5335
Sval<f6>v	13290

On observing the sizes of the cities that have been selected by the function, it can be seen that the function picked 20 cities which are of relatively large sizes. This is because the cities with huge populations have a much higher probability of being selected than the ones with small population.

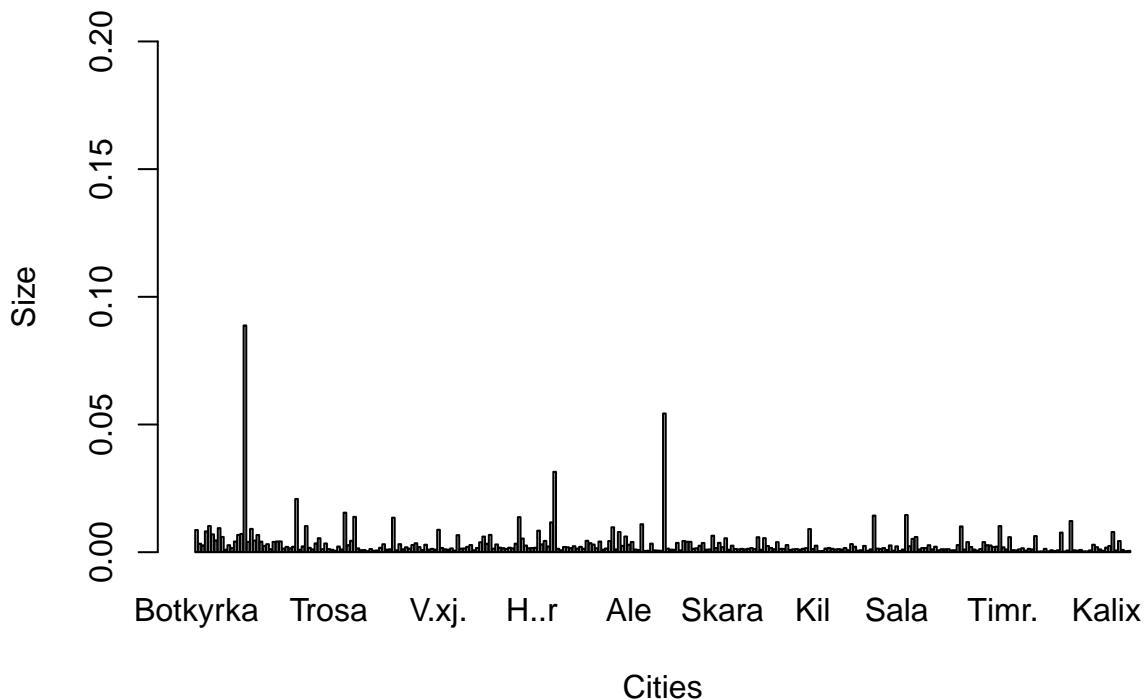
## TASK 1.5

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

```
# TASK 1.5

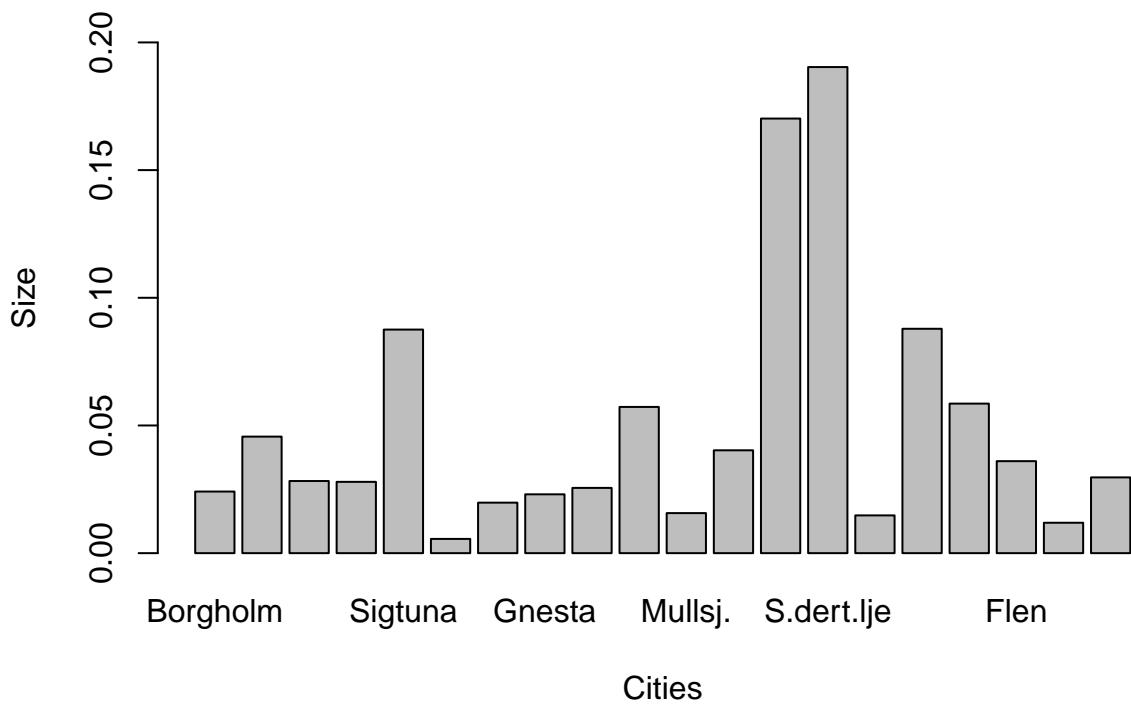
# Plot showing the size of all cities of the country
percentage<-data$Population/sum(data$Population)
barplot(percentage,names.arg = data$Municipality, xlab="Cities", ylab = "Size", ylim = c(0, 0.20))
title("Plot showing the size of all cities of the country")
```

## Plot showing the size of all cities of the country



```
# Plot showing the size of 20 selected cities
percentage_select<-select$Population/sum(select$Population)
barplot(percentage_select, names.arg = select$Municipality, xlab="Cities", ylab = "Size", ylim = c(0, 0.2))
title("Plot showing the size of 20 selected cities")
```

## Plot showing the size of 20 selected cities



It can be seen that the two histograms follow a similar distribution. This shows that the function implemented

to select 20 cities based on the probabilities proportional to the number of inhabitants of the city performs a good job in the city selection.

## QUESTION 2: Different distributions

### TASK 2.1

**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.**

The probability density for the double exponential (Laplace) function (PDF) is given by the formula:

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

where the variable  $x$  is a real number as is the location parameter  $\mu$  while the parameter  $\alpha$  is a real positive number.

The cumulative density function (CDF) of a continuous random variable  $X$  is defined as:

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

Thus, the cumulative distribution for the double exponential distribution is given by:

For  $x > \mu$  :

$$F(x) = \int_{-\infty}^x \frac{\alpha}{2} e^{-\alpha(x-\mu)} dx, \quad x > \mu$$

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

Integrating with respect to  $x$ , we have:

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

Now, for  $x \leq \mu$  :

$$F(x) = \int_{-\infty}^x \frac{\alpha}{2} e^{\alpha(x-\mu)} dx, \quad x \leq \mu$$

Integrating with respect to  $x$ , we have:

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

### The inverse CDF technique:

Given a uniform random number between zero and one in  $U$  (i.e  $U \sim U(0, 1)$ ), a random number from a Double-exponential distribution is given by solving the equation  $F(x)=U$  for  $x$ , as shown below:

For  $U > 1/2$  , we have:

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

Solving for x, we will get:

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

For  $U \leq 1/2$ , we have:

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

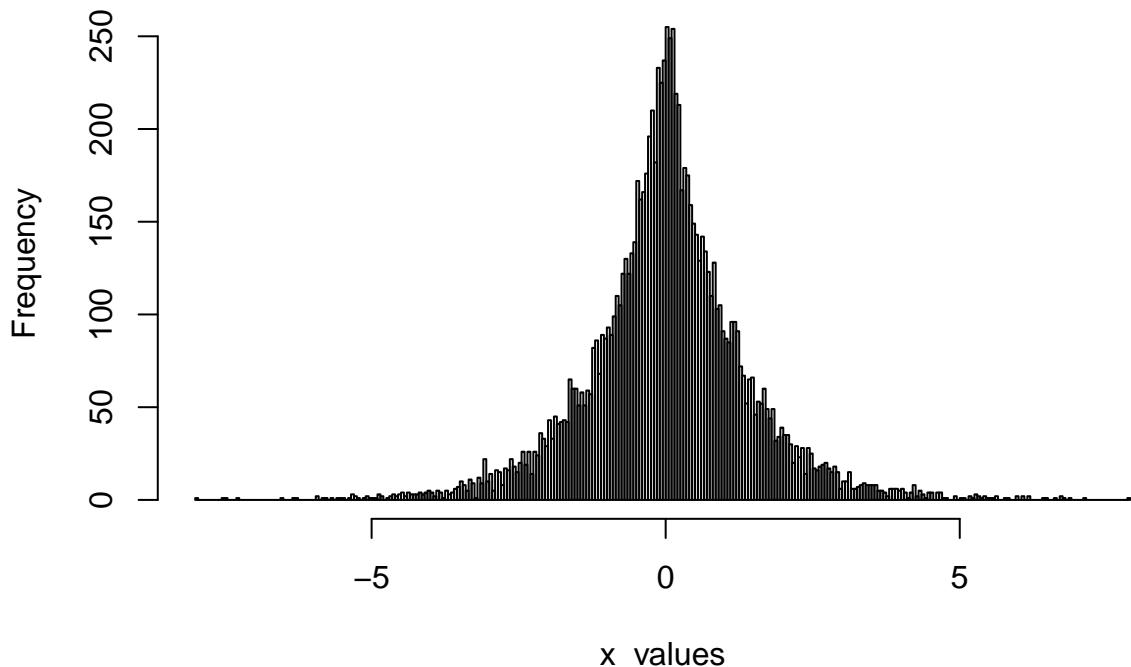
Solving for x, we will get:

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

# TASK 2.1

```
inverse_cdf = function(mu,alpha){  
  U = runif(1,0,1)  
  if (U <= 0.5){  
    x=mu+(log(2*U))/alpha  
  }else{  
    x=mu-(log(2-2*U))/alpha  
  }  
  return(x)  
}  
  
x_values <- c()  
for (i in 1:10000){  
  x_values[i] <- inverse_cdf(0,1)  
  x_values  
}  
  
hist(x_values, breaks = 300, main = "Generating Random Numbers using Inverse CDF Method")
```

## Generating Random Numbers using Inverse CDF Method



We can conclude from the histogram, that the implementation of our Non Uniform Random Number generators using inverse CDF method appears to be correct, at least when compared empirically against standard histogram for the laplace distribution

## TASK 2.2

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.

Acceptance/rejection methods for generating realizations of a random variable  $X$  make use of realizations of another random variable  $Y$  whose probability density  $g_y(x)$  is similar to the probability density of  $X$ ,  $f_x(x)$ . The random variable  $Y$  is chosen so that we can easily generate realizations of it and so that its density  $g_y(x)$  can be scaled to majorize  $f_x(x)$  using some constant  $c$ ; that is,

$$c * g_y(x) \geq f_x(x) \quad \text{for all } x$$

Where, the density  $g_y(x)$  is called the *majorizing* density or the *proposal* density, and  $c * g_y(x)$  is called the majorizing function. The density of interest,  $f_x(x)$ , is called the *target* density.

Thus, our majorizing density  $f_y(x) \sim DE(0, 1)$  is,

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

And the target density  $f_y(x) \sim N(0, 1)$  is,

$$f_y(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

To evaluate the majorizing constant  $c$ , we have,

$$c \geq \frac{f_x(x)}{g_y(x)}$$

Which is,

$$c \geq \sqrt{\frac{2}{\pi}} e^{-\frac{x^2}{2} + |x|}$$

By differentiating the above formula and setting to zero we got the maximum at  $x=1$ , Hence the  $c$  value will be,

$$\sqrt{\frac{2}{\pi}} e^{\frac{1}{2}} = \sqrt{\frac{2e}{\pi}}$$

```
# TASK 2.2
```

```
accept_reject <- function(){
```

```
  X<-c()
```

```

counter= 0 # counter for how many numbers been rejected

while (length(X)==0) { # as long as our vector is empty >> do
  counter = counter+1

Y<-inverse_cdf(0,1)
fy <- (1/sqrt(2*pi))*exp(-(1/2)*Y^2)

Gy <- 0.5*exp(-abs(Y))

c <- sqrt(2*exp(1)/pi)

U = runif(1,0,1)

if (U <= fy/(c*Gy)){
  X=Y
}
}
gen_itr <- cbind("gen" = X, "itr" = counter)
return(gen_itr)
}

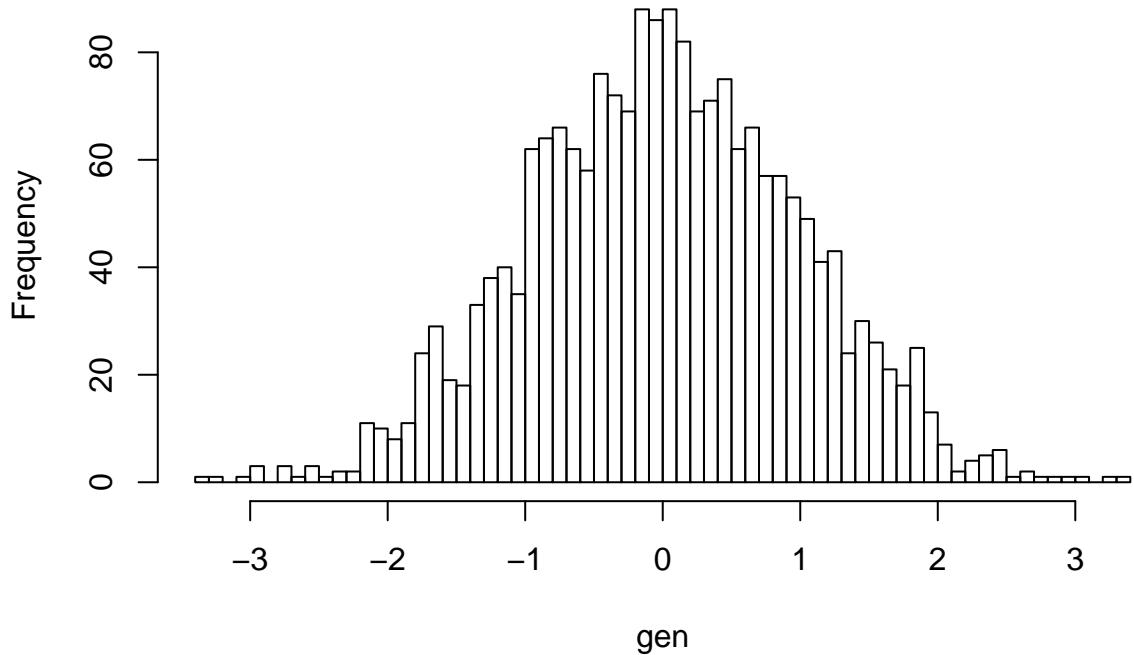
# generating 2000 random numbers using Acceptance\Rejection method
res <- list()
for (i in 1:2000){
  res[[i]]<- accept_reject()
}

# excluding the random numbers from the list for plot
gen <- c()
for (i in 1:2000){
  gen[i]<- res[[i]][1]
}

# plotting the generated random numbers using N(0,1)
hist(gen , breaks = 50, main = "Random numbers generated using Acceptance-Rejection Method")

```

## Random numbers generated using Acceptance–Rejection Method



```
# getting the total number of iterations (accepted+rejected) to compute the average
itr <- c()
for (i in 1:2000){
  itr[i] <- res[[i]][2]
}

# average rejection rate
average_rej <- 1-(2000/sum(itr))
average_rej

## [1] 0.2351816

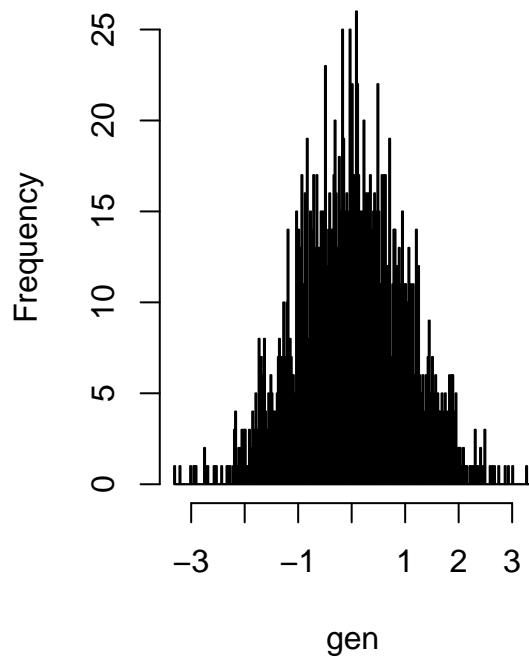
# expected rejection rate
c <- sqrt(2*exp(1)/pi)
expected_rej_rate<- 1-(1/c)
expected_rej_rate

## [1] 0.2398265

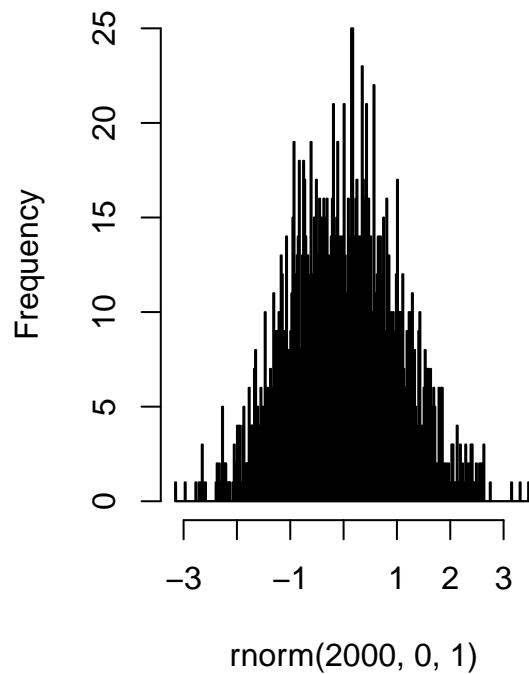
# Comparing Two plots for the generated random numbers

# plotting the generated random numbers using Acceptance-Rejection Method
par(mfrow=c(1,2))
hist(gen , breaks = 300, main = "R.N's using Acceptance-Rejection")
# histogram for 2000 R.N's generated by standard rnorm()
hist(rnorm(2000,0,1), breaks = 300,main = "R.N's using standard rnorm")
```

R.N's using Acceptance-Rejection



R.N's using standard rnorm



The above two histograms for the Random Numbers using Acceptance/Rejection method and the exact normal distribution show similar pattern, that means our method is quite acceptable.

## Appendix

```
knitr:::opts_chunk$set(echo = TRUE)
# Loading required R packages
library(ggplot2)
library(kableExtra)
library(gridExtra)

# QUESTION 1 : Cluster sampling

# TASK 1.1

# Importing the file
data <- read.csv2("population.csv")

# TASK 1.2

# Function which uses a uniform random number generator to select a city from a list
city_selection <- function(data){
  data$cumsum <- cumsum(data$Population) / sum(data$Population)
  selected <- which(runif(1) <= data$cumsum)[1]
```

```

city_selected <- data$Municipality[selected]
as.numeric(rownames(data[selected,]))
}

# TASK 1.3

select <- data.frame()

# List of all cities stored in data1
data1 <- data

# Selecting 20 cities
for(i in 1:20){
  selected <- which(city_selection(data1) == as.numeric(rownames(data)))
  select <- rbind(select, data1[selected,])
  # Removing the selected city from the list
  data1 <- data1[-c(selected),]
}

# TASK 1.4

# Dataframe of 20 selected cities
cat("Selected cities : \n")
select$Municipality

# Selected cities along with their respective sizes
city_size = data.frame(City = select$Municipality, Size = select$Population)

kable(city_size, booktabs = T) %>%
kable_styling(latex_option = "striped")

# TASK 1.5

# Plot showing the size of all cities of the country
percentage<-data$Population/sum(data$Population)
barplot(percentage,names.arg = data$Municipality, xlab="Cities", ylab = "Size", ylim = c(0, 0.20))
title("Plot showing the size of all cities of the country")

# Plot showing the size of 20 selected cities
percentage_select<-select$Population/sum(select$Population)
barplot(percentage_select, names.arg = select$Municipality, xlab="Cities", ylab = "Size", ylim = c(0, 0
title("Plot showing the size of 20 selected cities")

# QUESTION 2 : Different distributions

# TASK 2.1

```

```

inverse_cdf = function(mu,alpha){
  U = runif(1,0,1)
  if (U <= 0.5){
    x=mu+(log(2*U))/alpha
  }else{
    x=mu-(log(2-2*U))/alpha
  }
  return(x)
}

x_values <- c()
for (i in 1:10000){
  x_values[i] <- inverse_cdf(0,1)
  x_values
}

hist(x_values, breaks = 300, main = "Generating Random Numbers using Inverse CDF Method")

# TASK 2.2

accept_reject <- function(){

  X<-c()
  counter= 0 # counter for how many numbers been rejected

  while (length(X)==0) { # as long as our vector is empty >> do
    counter = counter+1

    Y<-inverse_cdf(0,1)
    fy <- (1/sqrt(2*pi))*exp(-(1/2)*Y^2)

    Gy <- 0.5*exp(-abs(Y))

    c <- sqrt(2*exp(1)/pi)

    U = runif(1,0,1)

    if (U <= fy/(c*Gy)){
      X=Y
    }
  }
  gen_itr <- cbind("gen" = X, "itr" = counter)
  return(gen_itr)
}

# generating 2000 random numbers using Acceptance\Rejection method
res <- list()
for (i in 1:2000){
  res[[i]]<- accept_reject()
}
# excluding the random numbers from the list for plot
gen <- c()

```

```

for (i in 1:2000){
  gen[i]<- res[[i]][1]
}

# plotting the generated random numbers using N(0,1)
hist(gen , breaks = 50, main = "Random numbers generated using Acceptance-Rejection Method")

# getting the total number of iterations (accepted+rejected) to compute the average
itr <- c()
for (i in 1:2000){
  itr[i]<- res[[i]][2]
}

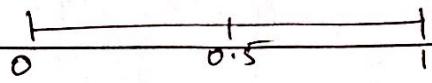
# average rejection rate
average_rej <- 1-(2000/sum(itr))
average_rej
# expected rejection rate
c <- sqrt(2*exp(1)/pi)
expected_rej_rate<- 1-(1/c)
expected_rej_rate
# Comparing Two plots for the generated random numbers

# plotting the generated random numbers using Acceptance-Rejection Method
par(mfrow=c(1,2))
hist(gen , breaks = 300, main = "R.N's using Acceptance-Rejection")
# histogram for 2000 R.N's generated by standard rnorm()
hist(rnorm(2000,0,1), breaks = 300,main = "R.N's using standard rnorm")

```

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

$$u = \text{unif}(0, 1)$$



Inverse CDF -

$$f(x) \leftarrow \text{pdf}$$

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

need to  
Rewrite

$$\text{pdf} \rightarrow \text{cdf} = u \rightarrow x = (\text{cdf})^{-1}$$

$$\textcircled{1} \frac{\alpha}{2} e^{-\alpha(x-\mu)} \quad | \quad x \geq \mu, x \geq 0$$

$$-\infty < x < \infty$$

DE

$$\textcircled{2} \frac{\alpha}{2} e^{\alpha(x-\mu)}, \quad x \leq \mu, x \leq 0$$

$$\textcircled{1} u = 1 - \int_x^{\infty} \frac{\alpha}{2} e^{-\alpha(x-\mu)} dx \rightarrow \text{as } x > \mu \star [\text{cdf always sums up to 1}]$$

as  $x > \mu$   $\star$  [cdf always sums up to 1]

$\hookrightarrow$  area under the curve

$$= 1 - \frac{\alpha}{2} \cdot e^{-\alpha(x-\mu)} \Big|_x^{\infty}$$

$$= 1 + \frac{1}{2} \left( \textcircled{0} - e^{-\alpha(x-\mu)} \right) \quad | \quad \frac{1}{e^{\infty}} \Rightarrow \frac{1}{\infty} = 0$$

$$u = 1 - \frac{1}{2} (e^{-\alpha(x-\mu)}), \quad \text{when } u > 1/2$$

$$e^{-\alpha(x-\mu)} = 2 - u$$

$$-\alpha(x-\mu) = \ln(2-u)$$

$$x = \mu - \frac{1}{\alpha} (\ln(2-u)) \quad | \quad \text{when } u > \frac{1}{2}$$

$$\textcircled{2} \quad u = \int_{-\infty}^x \alpha e^{\alpha(x-\mu)} dx$$

$$= \frac{\alpha}{2} \cdot e^{\alpha(x-\mu)} \Big|_{-\infty}^x$$

$$= \frac{1}{2} (e^{\alpha(x-\mu)} - 0)$$

$$u = \frac{1}{2} (e^{\alpha(x-\mu)}) \quad \text{when } u \leq 1/2$$

$$e^{\alpha(x-\mu)} = 2u$$

$$\alpha(x-\mu) = \ln(2u)$$

$$x = \mu + \frac{1}{\alpha} (\ln(2u)) \quad \text{where } u \leq 1/2$$

Acceptance / Rejection method -

$y \sim g(y) \sim DE$  ← majorizing density

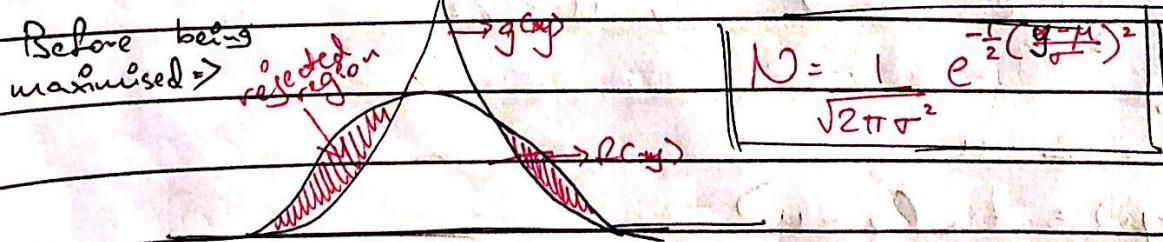
$x \sim f(x) \sim N(0, 1)$  ← target density

$U \sim \text{Unif}(0, 1)$

target func.

If  $U \leq \frac{f(x)}{c(g(y))}$   $\Rightarrow N(0, 1)$

then  $X = y$  → use  $f(x)$  takes  $y$  as desired realization



note:

there exists a constant  $c$  such that,

$$\forall y \ c g(y) \geq f(x)$$

$$\therefore c \geq f(y) \\ g(y)$$

$f(y) = N(0, 1) \rightarrow \text{subst. } \mu=0 \text{ & } \sigma=1$

$$f(y) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}y^2}$$

$$\Rightarrow c = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}y^2}$$

$$c = \sqrt{\frac{2}{\pi}} e^{-\frac{1}{2}y^2 + ly^2}$$

$$c' = \frac{\partial c}{\partial y} = \sqrt{\frac{2}{\pi}} e^{-\frac{1}{2}y^2 + ly^2} \cdot \left( -y + \frac{y}{|y|} \right)$$

To get value of  $y$ ,

Set  $c'$  to 0,

$$0 = \sqrt{\frac{2}{\pi}} e^{-\frac{1}{2}y^2 + ly^2} \left( -y + \frac{y}{|y|} \right)$$

$$\ln \sqrt{\frac{2}{\pi}} = -\frac{1}{2}y^2 + ly^2 \text{ When } y=1, c'=0.$$

Subst.  $y=1$  in  $c$ ,

$$c = \sqrt{\frac{2}{\pi}} e^{-\frac{1}{2}} \approx 1.3$$

Random samples obtained should be of a normal distribution.

When you  $\propto$   $g(y)$  with  $c$ , it will get maximised.

Area under target func. = 1

Because area of prob. = 1

Area under prob. dist. = prob.

Area of maj. func. =

Rej. rate is the prob. that a pt. is outside target distr.

After being maximized  $\rightarrow$

$$\text{Area}(g(x)) = c$$

$$\begin{aligned} \text{Exp. Rej. rate} \\ = \frac{\text{Rej. region}}{\text{Total region}} \end{aligned}$$

$$\text{Area}(f(x)) = 1$$

$$\Rightarrow \frac{c-1}{c}$$