Computational Statistics

Lab 3

Emil K Svensson and Rasmus Holm

2017-02-15

Question 1

1.1

```
pop <- read.csv2("../data/population.csv", encoding = "latin1")
```

1.2

```
cityUnif <- function(data){
    data$prob <- data$Population / sum(data$Population)
    data$cumprob <- cumsum(data$prob)
    return(which.min(data$cumprob < runif(1)))
}
```

1.3

```
rmpop <- pop

set.seed(123456)
while(nrow(rmpop) > 20){
    rmpop <- rmpop[-cityUnif(rmpop),]
}
```

1.4

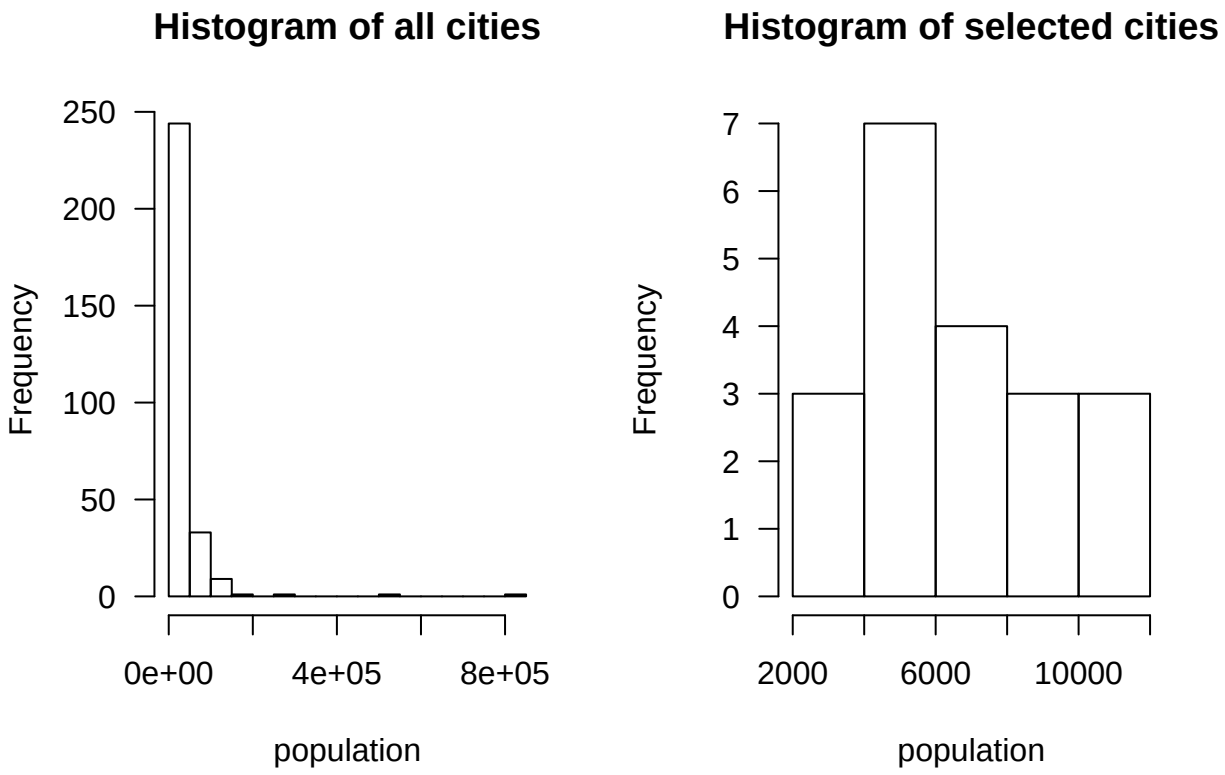
```
print(paste(as.character(rmpop$Municipality),":",rmpop$Population))
```

```
##  [1] "Vingåker : 8911"        "Ydre : 3672"
##  [3] "Aneby : 6446"           "Borgholm : 10806"
##  [5] "Dals-Ed : 4729"         "Grästorp : 5857"
##  [7] "Gullspång : 5335"       "Hjo : 8859"
##  [9] "Strömstad : 11690"      "Hällefors : 7333"
## [11] "Ljusnarsberg : 5055"    "Skinnskatteberg : 4567"
## [13] "Rättvik : 10797"        "Bräcke : 6865"
## [15] "Dorotea : 2900"         "Vilhelmina : 7156"
## [17] "Arjeplog : 3143"        "Jokkmokk : 5210"
## [19] "Älvsbyn : 8387"         "Övertorneå : 4920"
```

As seen in the output above, we in general select cities with small populations.

1.5

```
par(mfrow = c(1,2))

hist(pop$Population, main = "Histogram of all cities", xlab = "population", las = 1, breaks = 20)
hist(rmpop$Population, main = "Histogram of selected cities", xlab = "population", las = 1)
```




```
par(mfrow = c(1,1))
```

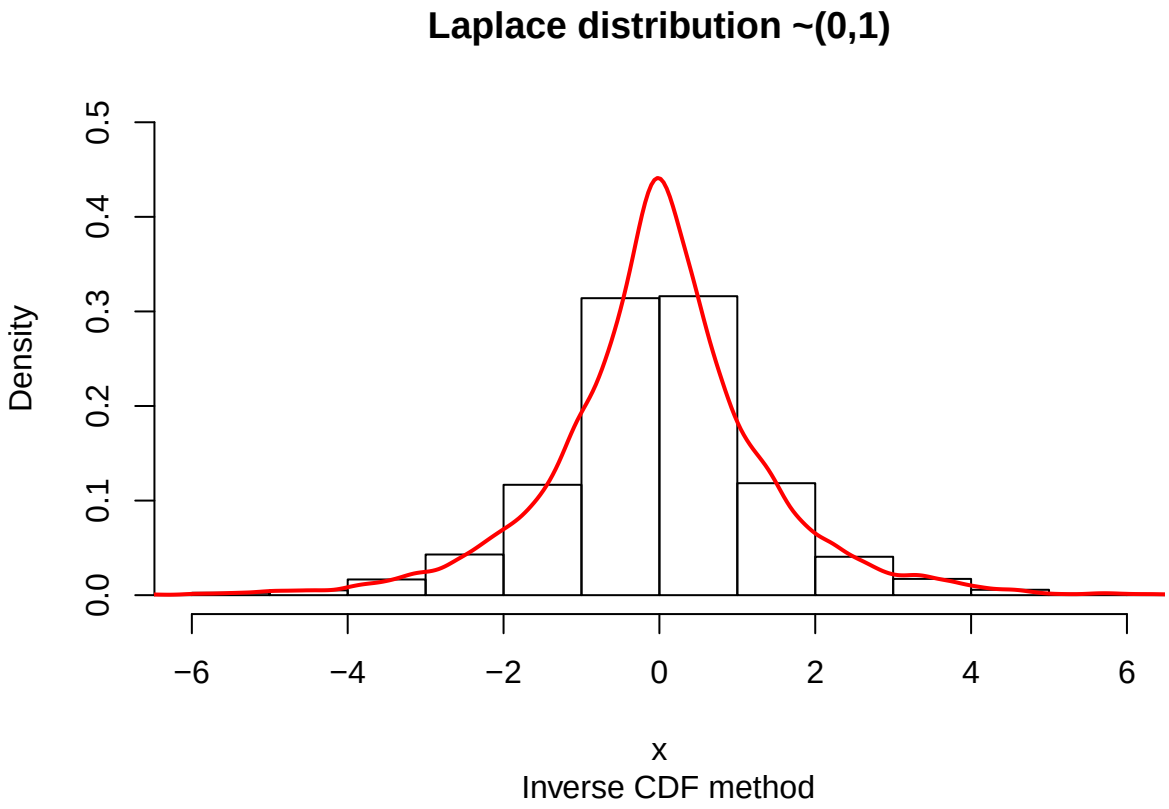
It selects cities with low population, it seems resonable given the large number of cities with small populations.

Question 2

2.1

```
invLap <- function(){
    rng <- runif(1)

    if(rng > 0.5) {
        return(-log(2 - 2 * rng) )
    } else {
        return(log(2 * rng))
    }
}
```

```r
plotdata <- sapply(1:10000,FUN = function(x) invLap())

hist(plotdata, prob = TRUE, ylim = c(0,0.5), xlim = c(-6,6),
     xlab = "x", main = "Laplace distribution ~(0,1)",
     sub = "  Inverse CDF method", breaks = 20)
lines(density(plotdata), col = "red", lwd = 2)
```



Yes, the histogram and the density-curve we included seems resonable compared to the normal shape of a Laplace distribution.

2.2

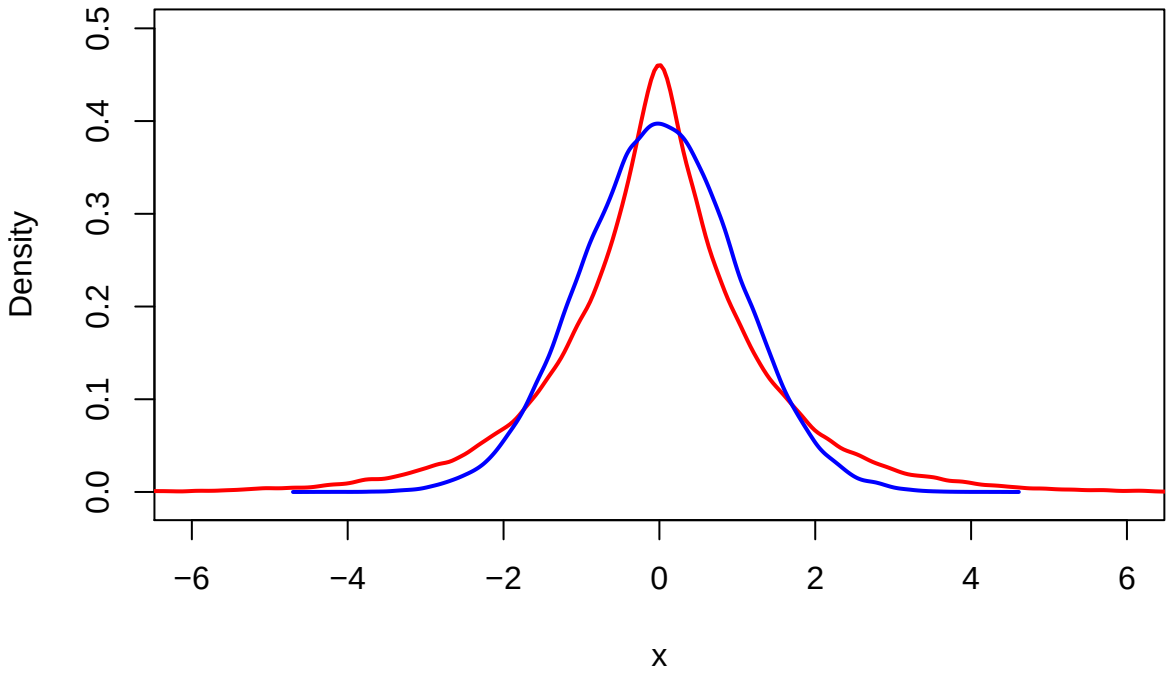
```r
DE01 <- function(x) {
    (1 / 2) * exp(-abs(x))
}

ar <- function(c) {
  rej <- 0

  generated <- FALSE
  x <- c()

  while(!generated){
      y <- invLap() # Y ~ f_y
      u <- runif(1) # U(0,1)
```

```
      fx <- dnorm(y, mean = 0, sd = 1) # fx(y)
      fy <- DE01(y) #f_y

      if(u < fx / (c * fy)){
          return(c(y, rej))  #alt. set x <- y and generated = TRUE, to end the loop
      }

      rej <- rej + 1
  }
}
```

Here is our function.

Choosing the appropriate c

```
set.seed(123456)
n <- 100000
lap <- sapply(1:n,FUN = function(x) invLap())
nor <- rnorm(n, 0, 1)

plot(x = 0, y = 0, col ="white", xlim = c(-6,6),
     ylim = c(-0.01, 0.5), xlab = "x",
     ylab = "Density" )
lines(density(lap), col = "red", lwd = 2)
lines(density(nor), col = "blue", lwd = 2)
```



Our approach for approximating c was to try to make a laplace-curve that would as much as possible cover the whole density-curve for the normal distribution.

For x > 0 since it is symmetric

$cf_y(x) - f_x(x) = 0$

$c/2e^{-x} - 1/(\sqrt{2\pi})e^{-x^2/2} = 0$

$log(c/2) - x - log(1/(\sqrt{2\pi})) + x^2/2 = 0$

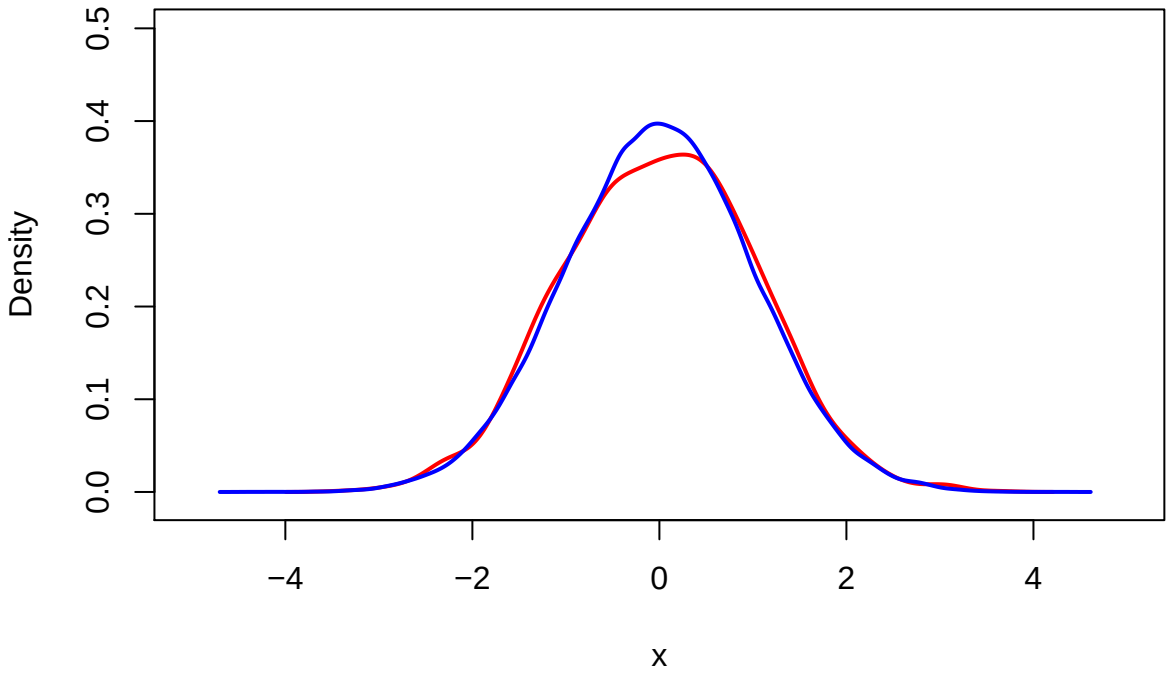
after some equation-solving we come to the soloution

$c = 1.315$

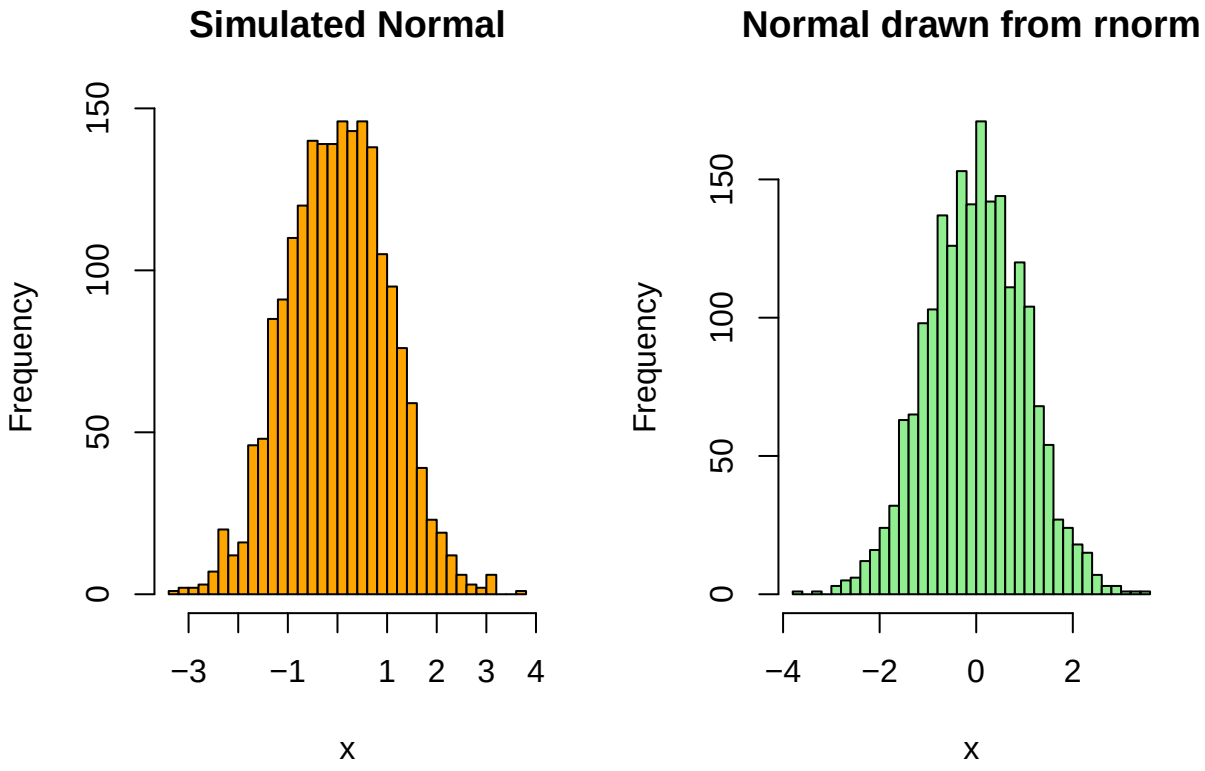
Simulation and comparison

```
ARsim <- data.frame(rn = 1, rej = 1)
c <- 1.32

for(i in 1:2000){
    ARsim[i,] <- ar(c=c)
}
```

```
plot(x = 0, y = 0, col ="white", xlim = c(-5,5),
     ylim = c(-0.01,0.5), xlab = "x", ylab = "Density" )
lines(density(ARsim$rn), col="red", lwd=2)
lines(density(nor), col="blue", lwd=2)
```




```
par(mfrow = c(1,2))
hist(ARsim$rn, main = "Simulated Normal", xlab = "x",
     col = "orange",  breaks = 30)

set.seed(123456)
hist(rnorm(2000,0,1), main = "Normal drawn from rnorm", xlab = "x",
     col = "lightgreen", breaks = 30 )
```



The simulated one is a bit more rough around the edges and more narrow in the center in comparison to the histogram drawn from the rnorm.

The expected rejection rate is $\frac{n(M-1)}{nM}$ since the expected number of iterations before a accepted value is M so intuitively the number of rejections is $M-1$. n here is only the number of values that we use, but these cancle eachother out so they are not neccesary but we left them here for clarity. The theoretical rejection rate would be 0.242 and what we empirically found was 0.262 which is reasonbly close.