Lab3,Group15

Naveen Gabriel ,Sridhar Adhirkala

2019-02-14

Contents

1 Appendix

```r
knitr::opts_chunk$set(
    echo = TRUE,
    eval=TRUE,
    message = FALSE,
    warning = FALSE,
    comment = NA
)

library(readxl)
library(ggplot2)

#reading data and setting col names
population <- read_excel("population.xls",
                         skip = 9, col_names = FALSE, na = '.')

colNm = c("Code", "County Municipality", "Population", "Population growth",
          "Live Births", "Deaths", "Population surplus", "In_mig_tot", "In_mig_from_sc",
          "In_mig_from_ros", "In_mig_from_ab", "Out_mig_tot", "Out_mig_from_sc",
          "Out_mig_from_ros", "Out_mig_from_ab", "Net_mig_tot", "Net_mig_from_sc",
          "Net_mig_from_ros", "Net_mig_from_ab", "Adjustments")
colnames(population) = colNm
```

```r
#splitting counties and cities
population$keep = population$Code <= 25
counties = population[population$keep,c("Code", "County Municipality", "Population")]
cities = population[!population$keep,c("Code", "County Municipality", "Population")]

#function to randomly select a city
selectRandCity = function(data){
  total_pop = sum(data$Population)
  data$cumPop = cumsum(data$Population)
  randNum = ceiling(runif(1, 0, total_pop))
  selected_ind = which.max((data$cumPop > randNum )*1)
  return(selected_ind)
}

#setting seed to get reproducable results
set.seed(123456)
#selecting 20 random cities
selectedCities = cities[1,]
for(i in 1:20){
  ind = selectRandCity(cities)
  selectedCities[i,] = cities[ind,]
  cities = cities[-ind,]
}

#print selected cities
print(selectedCities)

ggplot(cities, aes(cities$`County Municipality` , cities$Population)) +
  geom_bar(stat = "identity") + coord_flip()

ggplot(selectedCities, aes(selectedCities$`County Municipality`, selectedCities$Population)) +
  geom_bar(fill=selectedCities$Population, stat = "identity") + coord_flip()
knitr::opts_chunk$set(
    echo = TRUE,
    eval=TRUE,
    message = FALSE,
    warning = FALSE,
    comment = NA
)

library(ggplot2)

set.seed(123456)
p <- data.frame(runif(10000,0,1))  #Generate probabilities between 0 and 1
colnames(p) <- "Uniform_num"

#Generate samples from inverse of CDF
c <- ifelse(p$Uniform_num<0.5, log(2*p$Uniform_num), -log(2-(2*p$Uniform_num)))

p$X <- c

#Estimating distribution of true laplace
p$Laplace <- rmutil::rlaplace(10000,0,1)
```

```r
ggplot(p) + geom_histogram(aes(X),fill="red",alpha =0.7) +
    geom_histogram(aes(Laplace),fill="black") + xlab("Samples") +
    ylab("Frequency") + ggtitle("Inverse CDF vs true Laplace distribution")
accept_reject <- function(sam) {
    f_x <- c()
    cnt <- 1
    while(cnt<=sam){
        U <- runif(1,0,1)
        r_x <- ifelse(U<0.5, log(2*U), -log(2-(2*U)))  #generate a random variable from laplace distrib
        uni <- runif(1)
        frac <- exp(-(r_x^2)/2 + abs(r_x) - 0.5)  #value of f(X)/c(g(X))

        if(uni<=frac){
            f_x[cnt]<- r_x
            cnt <- cnt+1
        }
        total <<- total + 1
    }

return(f_x)
}

n = 2000
total <-0
values <- accept_reject(n)
normal_dist <- rnorm(2000,0,1)

compare_data <- cbind(values,normal_dist)
compare_data <- as.data.frame(compare_data)

colnames(compare_data) <- c("Accept_Rejection","By_Rnorm")

hist(compare_data$Accept_Rejection,main = "Normal distribution generated by Accept/Rejection method", xl
ggplot(compare_data) + geom_histogram(aes(Accept_Rejection,fill="Accept_Rejection"),alpha=0.4) +
      geom_histogram(aes(By_Rnorm,fill ="By_Rnorm"), alpha =0.5)
rj_rt <- 1-(2000/total)
cat("The average rejection rate is :", rj_rt)
```

2 Assignment 1

3 1. Import Data into R

```r
library(readxl)
library(ggplot2)

#reading data and setting col names
population <- read_excel("population.xls",
                         skip = 9, col_names = FALSE, na = '.')

colNm = c("Code", "County Municipality", "Population", "Population growth",
          "Live Births", "Deaths", "Population surplus", "In_mig_tot", "In_mig_from_sc",
          "In_mig_from_ros", "In_mig_from_ab", "Out_mig_tot", "Out_mig_from_sc",
          "Out_mig_from_ros", "Out_mig_from_ab", "Net_mig_tot", "Net_mig_from_sc",
          "Net_mig_from_ros", "Net_mig_from_ab", "Adjustments")
colnames(population) = colNm

#splitting counties and cities
population$keep = population$Code <= 25
counties = population[population$keep,c("Code", "County Municipality", "Population")]
cities = population[!population$keep,c("Code", "County Municipality", "Population")]
```

4 2. Creating a function to select random city

```r
#function to randomly select a city
selectRandCity = function(data){
  total_pop = sum(data$Population)
  data$cumPop = cumsum(data$Population)
  randNum = ceiling(runif(1, 0, total_pop))
  selected_ind = which.max((data$cumPop > randNum )*1)
  return(selected_ind)
}
```

In this part we are creating a function that selects and returns a randomly selected row index. The random selection is based on the population of the cities. Cities with lager population have higher probability of getting selected.

Since it is easier to maintain precision with integers I am not converting the population into probabilities, insted I am using cumulative sum to select a random city. I am generating a random integer between 1 and the total population. The cumulative sum divides the scale of population into bins, and I am just checking which bin does the randomly generated integer falls into and am selecting that city.

5 3. Using the function created in the previous step

```r
#setting seed to get reproducable results
set.seed(123456)
#selecting 20 random cities
selectedCities = cities[1,]
```

```r
for(i in 1:20){
  ind = selectRandCity(cities)
  selectedCities[i,] = cities[ind,]
  cities = cities[-ind,]
}
```

6 4. Selected Cities

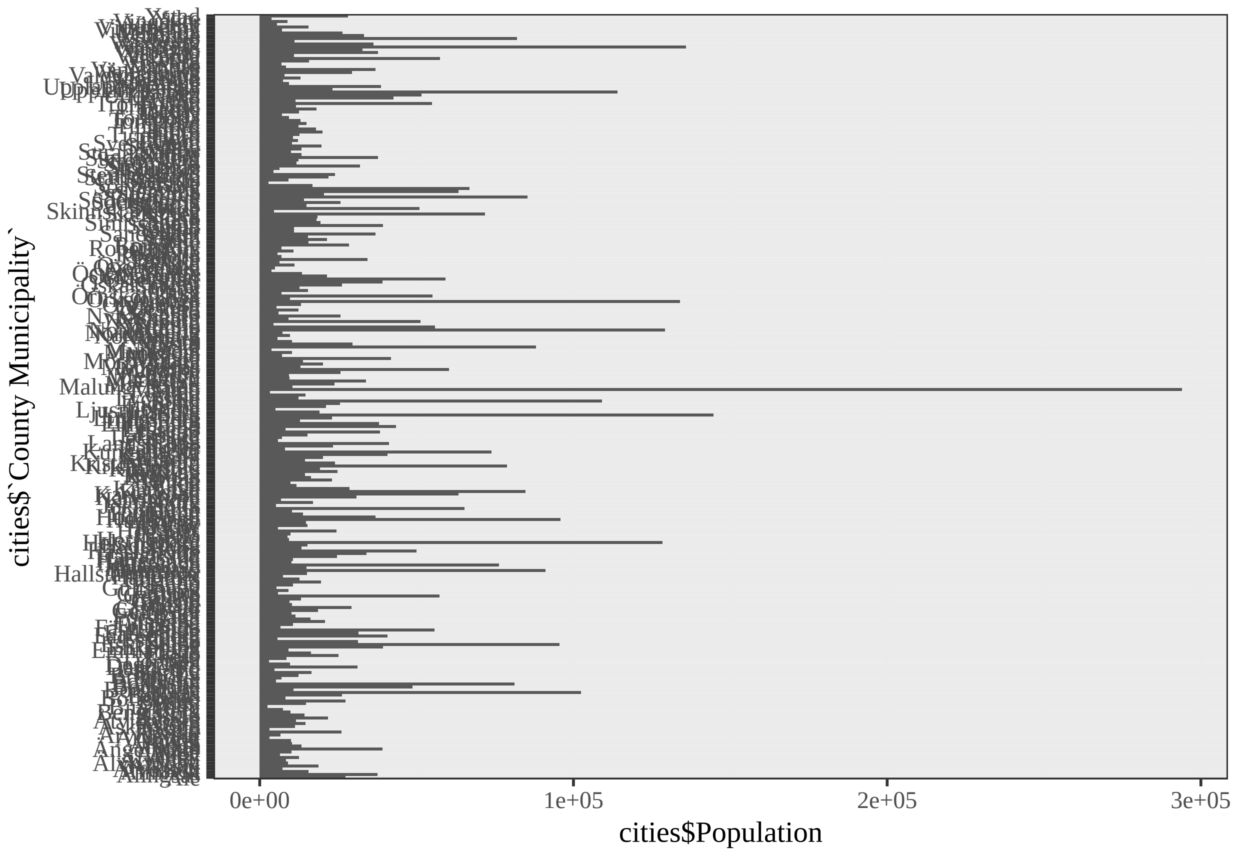
```r
#print selected cities
print(selectedCities)
```

```
# A tibble: 20 x 3
    Code `County Municipality` Population
 * <dbl> <chr>                      <dbl>
 1  1883 Karlskoga                  29742
 2  1491 Ulricehamn                 22753
 3   880 Kalmar                     62388
 4   680 Jönköping                 126331
 5   781 Ljungby                    27410
 6   160 Täby                       63014
 7  1287 Trelleborg                 41891
 8   180 Stockholm                 829417
 9  2580 Luleå                      73950
10   380 Uppsala                   194751
11  1982 Fagersta                   12249
12  1480 Göteborg                  507330
13  2281 Sundsvall                  95533
14  2180 Gävle                      94352
15  2581 Piteå                      40860
16  2121 Ovanåker                   11530
17  2061 Smedjebacken               10758
18   483 Katrineholm                32303
19   881 Nybro                      19576
20  1861 Hallsberg                  15235
```

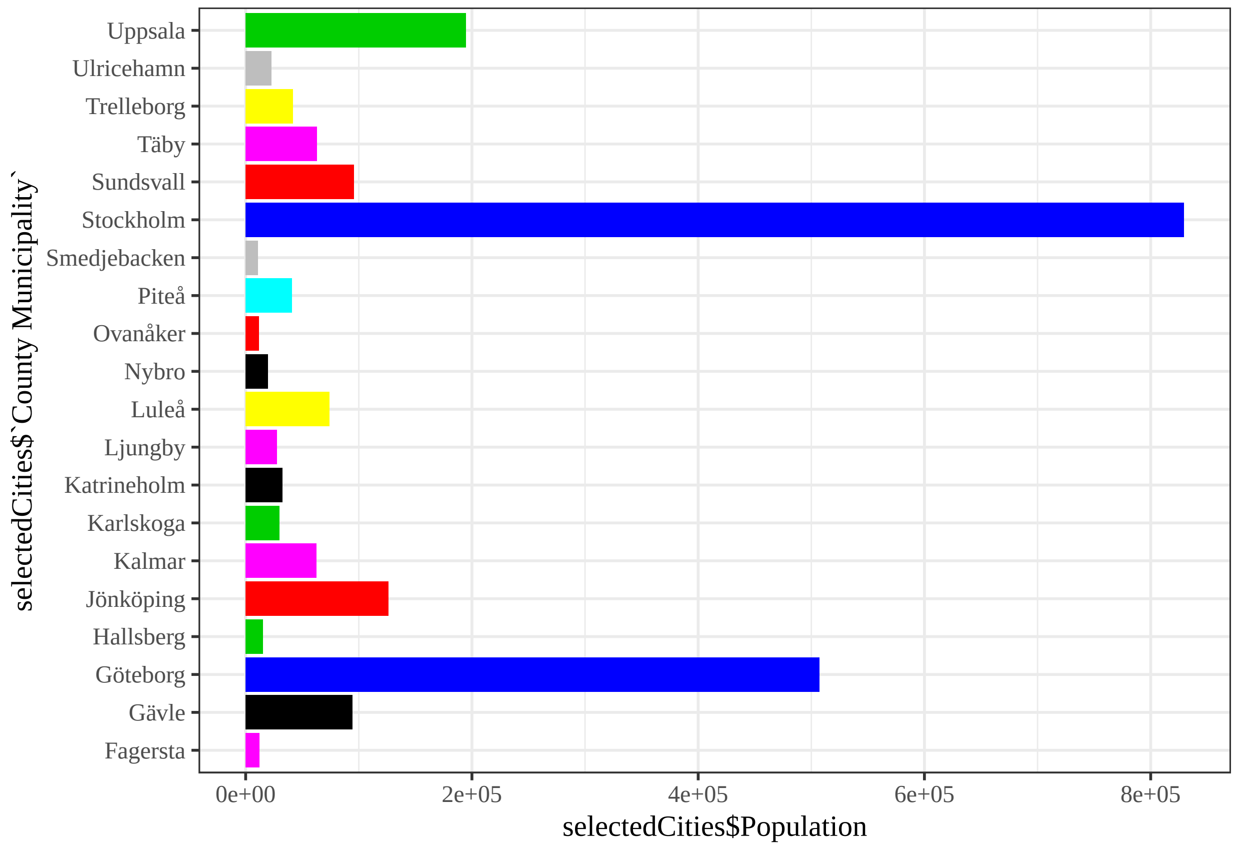
The cities with the largest population are selected in this random process.

7 5. Plot showing population of cities selected

```r
ggplot(cities, aes(cities$`County Municipality` , cities$Population)) +
  geom_bar(stat = "identity") + coord_flip()
```



```r
ggplot(selectedCities, aes(selectedCities$`County Municipality`, selectedCities$Population)) +
  geom_bar(fill=selectedCities$Population, stat = "identity") + coord_flip()
```



Some of the cities with the largest population, like Stockholm, Gothemborg, Upsella, were selected as they were given higher priority than others. Stockholm has the largest population, so it has the highest probability of getting selected. This turns out to be true as it gets selected almost every time we run a simulation with different seed.

The majority of the cities do not have too large a population, so most of the 20 random cities selected is made up from these. They have low probability of gettin selected but there are too many ofsuch cities, so these cities are the ones that fill up the majority of the 20 random picks.

8 Different distribution

8.1 Generate double exponential distribution DE(0,1) from Unif(0,1) by using

the inverse CDF method * There are three steps to generate samples from a distribution using inverse CDF method: 1. Generate random probabilities. This can be done using uniform random generator between 0 and 1. 2. From the pdf of a distribution, find the inverse CDF of the distribution. 3. Substitute the probabilities in the inverse CDF of the distribution to get the samples.

Repeat above procedure many times to generate samples so that it represent the distribution

We have Laplace distribution as :

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

CDF of Laplace distribution, **When x >=** μ, the sign of the mod operator does not change and we get CDF as :

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

When x < μ, the sign of(x-μ) flips and we get CDF as:

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

The inverse of laplace distribution,

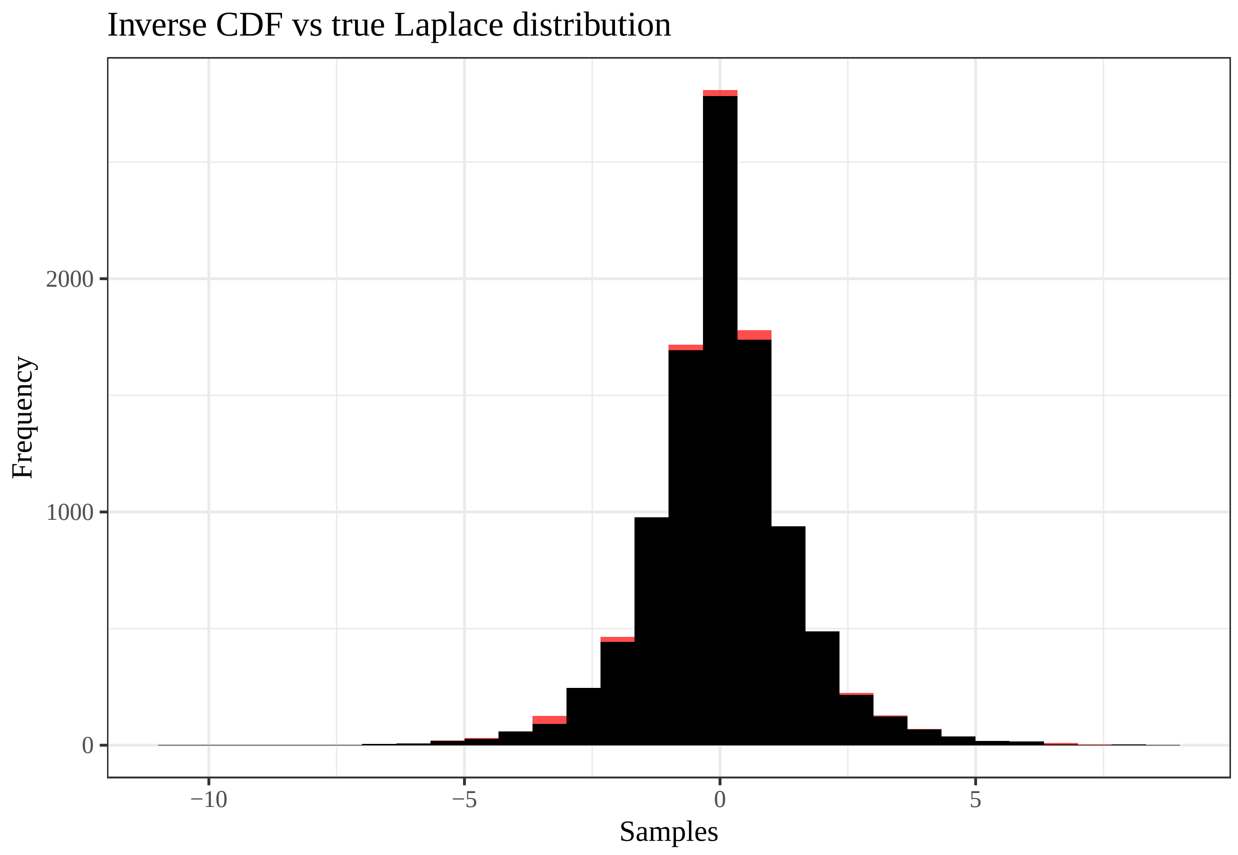
When P<0.5 and this happens when x < μ :

$$F^{-1}(P) = \frac{1}{\alpha} * log(2P) + \mu$$

When **P >=0.5* and this happens when x >= μ :

$$F^{-1}(P) = \mu - \frac{1}{\alpha} * log(2 - log2P)$$

```
library(ggplot2)

set.seed(123456)
p <- data.frame(runif(10000,0,1))  #Generate probabilities between 0 and 1
colnames(p) <- "Uniform_num"

#Generate samples from inverse of CDF
c <- ifelse(p$Uniform_num<0.5, log(2*p$Uniform_num), -log(2-(2*p$Uniform_num)))

p$X <- c

#Estimating distribution of true laplace
p$Laplace <- rmutil::rlaplace(10000,0,1)

ggplot(p) + geom_histogram(aes(X),fill="red",alpha =0.7) +
    geom_histogram(aes(Laplace),fill="black") + xlab("Samples") +
    ylab("Frequency") + ggtitle("Inverse CDF vs true Laplace distribution")
```



The graph shows how similar is the our lapalace histogram generated(red) with inverse CDF method with true laplace(black) for 10000 random numbers. Moreover, the generated distribution looks similar to Laplace distribution which resembles 2 exponential distribution back to back.

8.2 Acceptance/Rejection method

This is used when it is not possible to find inverse CDF of a distribution so that we can get RV using inverse but we have a pdf for the distribution.

The idea is to find a probabiliy distribution, g(x), from which we can generate a RV and able to tell whether this RV can be accepted for our target distribution f(x).

$$g(x|\mu = 0, \alpha = 1) = \frac{1}{2} * e^{-|x|} - LapalceDistribution$$
$$f(x|\mu = 0, \alpha = 1) = \frac{1}{\sqrt{2\pi}} * e^{\frac{-x^2}{2}} - NormalDistribution$$

We assume that the ratio $\frac{f(x)}{g(x)}$ is bounded by a constant C. C(g(x)) acts as a envelope for the target function f(x). This fraction inherently implies that how many fraction of RV for f(x) is included in C*g(x). We need to maximize this fraction so that we can cover most of the points in g(x) for f(x). For this we need to take differential and equate it to zero.

$$C \geq \frac{f(x)}{g(x)}$$
$$C \geq \frac{\sqrt{2} * e^{-\frac{x^2}{2}+|x|}}{\sqrt{\pi}}$$

Then we differentiate the above ratio and equate to 0 to get the value of x which will give the value of C.

$$\frac{\sqrt{2} * e^{-\frac{x^2}{2}+|x|}}{\sqrt{\pi}} * (\frac{|x|}{x} - x)$$

Setting the above differential to zero we get the maximum value of above equation at x=1, value of C is obtained.

$$C = \sqrt{\frac{2e}{\pi}}$$

The condition to accept RV generated from g(x) as RV for f(x) is :

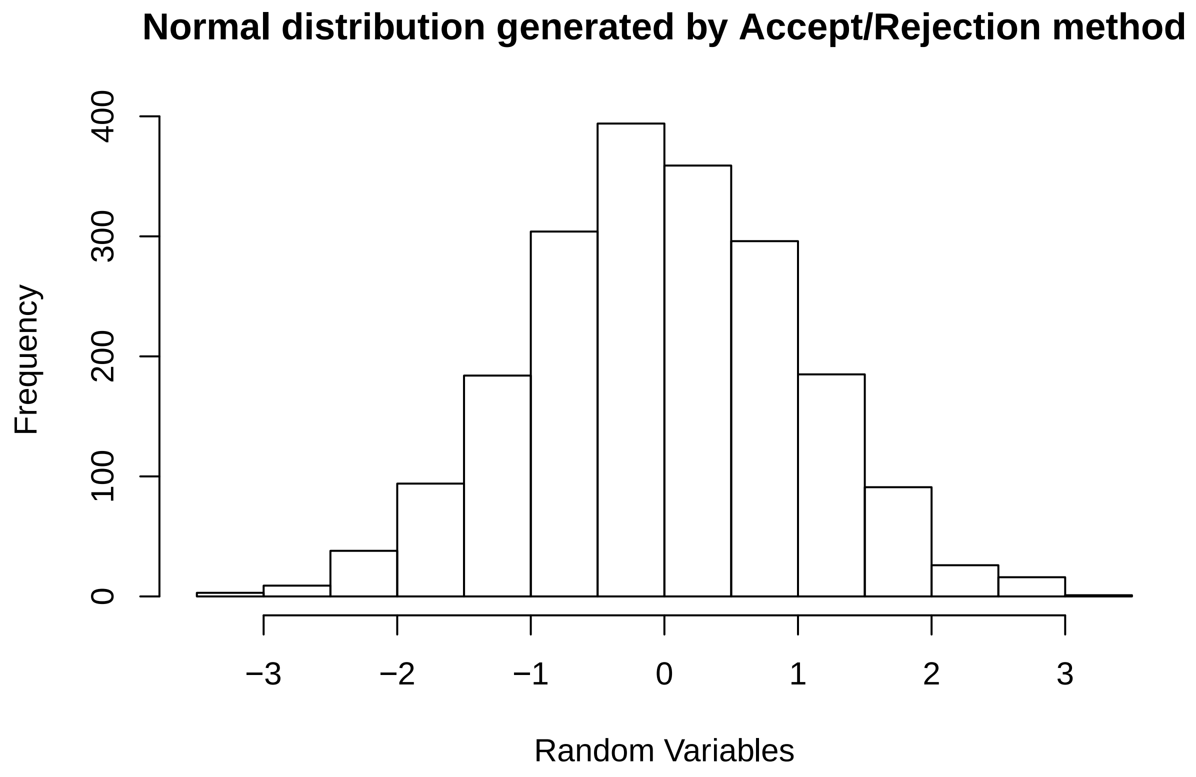
$$U \leq \frac{f(x)}{C * g(x)}$$
$$U \leq 0.5 * e^{\frac{x^2}{2}+|x|}$$

* **Following is the Acceptance Rejection algorithm :**

1. Sample $X \sim g(x)$.
2. Sample $U \sim Unif(0, 1)$.
3. Reject X if U > $\frac{f(x)}{C.g(x)}$. Go to step 1.
4. Else accept X for f(x).
5. Keep repeating the above step for desired number of samples.

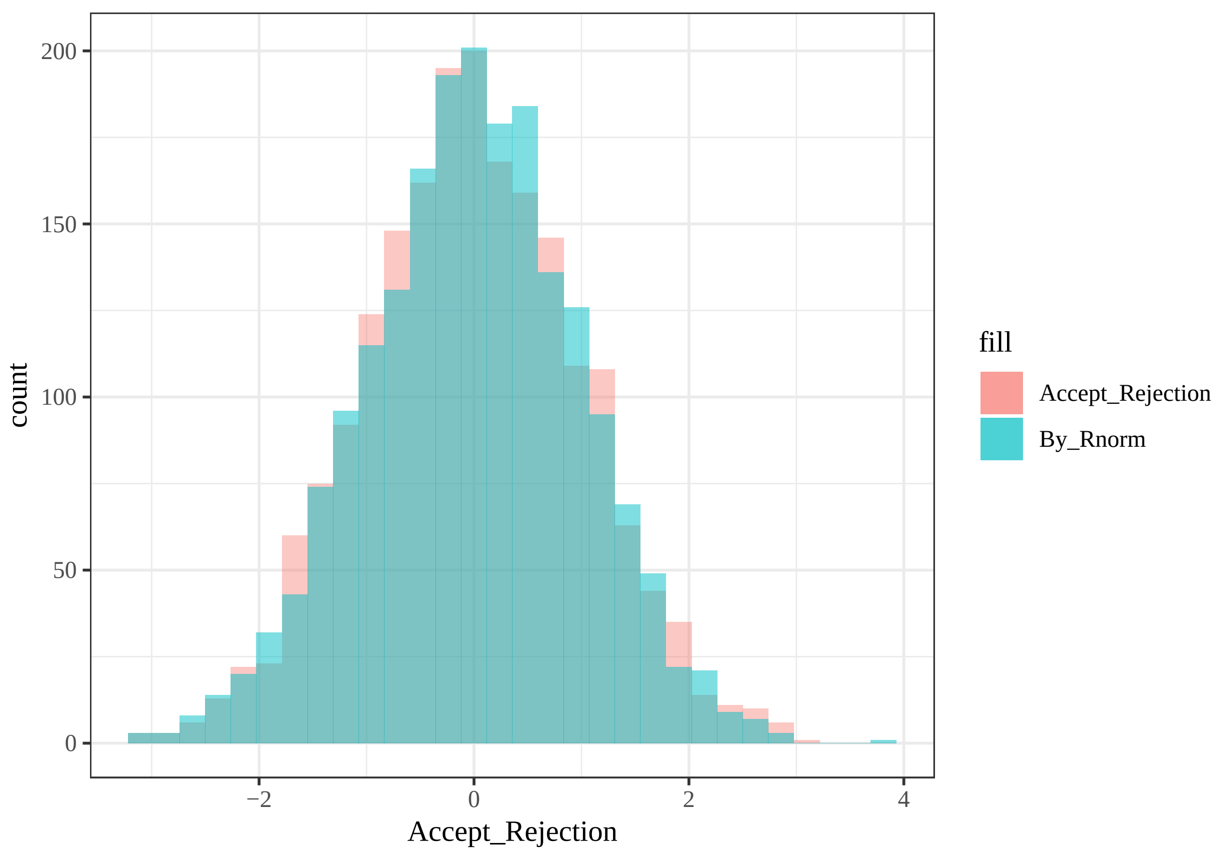
```
accept_reject <- function(sam) {
    f_x <- c()
    cnt <- 1
    while(cnt<=sam){
        U <- runif(1,0,1)
        r_x <- ifelse(U<0.5, log(2*U), -log(2-(2*U)))  #generate a random variable from laplace distrib
        uni <- runif(1)
        frac <- exp(-(r_x^2)/2 + abs(r_x) - 0.5)  #value of f(X)/c(g(X))

        if(uni<=frac){
            f_x[cnt]<- r_x
            cnt <- cnt+1
        }
        total <<- total + 1
    }

return(f_x)
}

n = 2000
total <-0
values <- accept_reject(n)
```

```r
normal_dist <- rnorm(2000,0,1)

compare_data <- cbind(values,normal_dist)
compare_data <- as.data.frame(compare_data)

colnames(compare_data) <- c("Accept_Rejection","By_Rnorm")

hist(compare_data$Accept_Rejection,main = "Normal distribution generated by Accept/Rejection method", x
```



We can see that the normal distribution generated by acceptance rejection method is nearly same as distribution generated by rnorm.

```r
ggplot(compare_data) + geom_histogram(aes(Accept_Rejection,fill="Accept_Rejection"),alpha=0.4) +
       geom_histogram(aes(By_Rnorm,fill ="By_Rnorm"), alpha =0.5)
```



The expected rejection rate is equal to:

$$1 - \frac{1}{c} = 0.2398264$$

The average rejection rate is :

$$1 - \frac{2000}{total}$$

Where total is the total number of iterations required to generate 2000 samples. Our average rejection rate is nearly equal to expected rejection rate.

```
rj_rt <- 1-(2000/total)
cat("The average rejection rate is :", rj_rt)
```

```
The average rejection rate is : 0.2595335
```

9 Appendix

```
knitr::opts_chunk$set(
    echo = TRUE,
    eval=TRUE,
    message = FALSE,
    warning = FALSE,
    comment = NA
)

library(readxl)
```

```r
library(ggplot2)

#reading data and setting col names
population <- read_excel("population.xls",
                         skip = 9, col_names = FALSE, na = '.')

colNm = c("Code", "County Municipality", "Population", "Population growth",
          "Live Births", "Deaths", "Population surplus", "In_mig_tot", "In_mig_from_sc",
          "In_mig_from_ros", "In_mig_from_ab", "Out_mig_tot", "Out_mig_from_sc",
          "Out_mig_from_ros", "Out_mig_from_ab", "Net_mig_tot", "Net_mig_from_sc",
          "Net_mig_from_ros", "Net_mig_from_ab", "Adjustments")
colnames(population) = colNm

#splitting counties and cities
population$keep = population$Code <= 25
counties = population[population$keep,c("Code", "County Municipality", "Population")]
cities = population[!population$keep,c("Code", "County Municipality", "Population")]

#function to randomly select a city
selectRandCity = function(data){
  total_pop = sum(data$Population)
  data$cumPop = cumsum(data$Population)
  randNum = ceiling(runif(1, 0, total_pop))
  selected_ind = which.max((data$cumPop > randNum )*1)
  return(selected_ind)
}

#setting seed to get reproducable results
set.seed(123456)
#selecting 20 random cities
selectedCities = cities[1,]
for(i in 1:20){
  ind = selectRandCity(cities)
  selectedCities[i,] = cities[ind,]
  cities = cities[-ind,]
}

#print selected cities
print(selectedCities)

ggplot(cities, aes(cities$`County Municipality` , cities$Population)) +
  geom_bar(stat = "identity") + coord_flip()

ggplot(selectedCities, aes(selectedCities$`County Municipality`, selectedCities$Population)) +
  geom_bar(fill=selectedCities$Population, stat = "identity") + coord_flip()
knitr::opts_chunk$set(
    echo = TRUE,
    eval=TRUE,
    message = FALSE,
    warning = FALSE,
    comment = NA
)
```

```r
library(ggplot2)

set.seed(123456)
p <- data.frame(runif(10000,0,1))  #Generate probabilities between 0 and 1
colnames(p) <- "Uniform_num"

#Generate samples from inverse of CDF
c <- ifelse(p$Uniform_num<0.5, log(2*p$Uniform_num), -log(2-(2*p$Uniform_num)))

p$X <- c

#Estimating distribution of true laplace
p$Laplace <- rmutil::rlaplace(10000,0,1)

ggplot(p) + geom_histogram(aes(X),fill="red",alpha =0.7) +
    geom_histogram(aes(Laplace),fill="black") + xlab("Samples") +
    ylab("Frequency") + ggtitle("Inverse CDF vs true Laplace distribution")
accept_reject <- function(sam) {
    f_x <- c()
    cnt <- 1
    while(cnt<=sam){
        U <- runif(1,0,1)
        r_x <- ifelse(U<0.5, log(2*U), -log(2-(2*U)))  #generate a random variable from laplace distrib
        uni <- runif(1)
        frac <- exp(-(r_x^2)/2 + abs(r_x) - 0.5)  #value of f(X)/c(g(X))

        if(uni<=frac){
            f_x[cnt]<- r_x
            cnt <- cnt+1
        }
        total <<- total + 1
    }

return(f_x)
}

n = 2000
total <-0
values <- accept_reject(n)
normal_dist <- rnorm(2000,0,1)

compare_data <- cbind(values,normal_dist)
compare_data <- as.data.frame(compare_data)

colnames(compare_data) <- c("Accept_Rejection","By_Rnorm")

hist(compare_data$Accept_Rejection,main = "Normal distribution generated by Accept/Rejection method", x
ggplot(compare_data) + geom_histogram(aes(Accept_Rejection,fill="Accept_Rejection"),alpha=0.4) +
       geom_histogram(aes(By_Rnorm,fill ="By_Rnorm"), alpha =0.5)
rj_rt <- 1-(2000/total)
cat("The average rejection rate is :", rj_rt)
```