Universidade de Brasília

DEPARTAMENTO DE ESTATÍSTICA

22 abril 2023

Lista 1

Prof. Dr. Donald Matthew Pianto

Aluno: Bruno Gondim Toledo

Matrícula: 15/0167636

Estatística Computacional

1º/2023

Questão 1

a)

Resultado analítico:

```
# Separando apenas as palavras com exatamente 5 letras

pcl <- dicionario %>%
  filter(str_length(palavras) == 5)%>%
  mutate(palavras = tolower(str_replace_all(iconv(palavras, "latin1", "ASCII//TRANSLIT"), "[^[:alnum

# Como o exercicio cita que iremos trabalhar apenas com combinações de letras sem inclusão de caract

pcl <- pcl %>%
  distinct()

# 5481-5427

# Com isso, estaremos trabalhando com 54 palavras a menos (palavras idêntias à outras, salvo por um

# Certificando a quantidade de letras para calcular a permutação
letras <- dicionario %>%
  mutate(letras = tolower(str_sub(palavras, 1, 1)))%>%
  select(letras)%>%
  distinct()

# Então, sendo 26 letras diferentes, a quantidade de combinações possíveis com 5 letras é da ordem d

# 26*26*26*26*26

# Ou seja,

# 26^5

# Como são 5427 palavras válidas de 5 letras, então a probabilidade de se obter uma combinação válid

prob <- 5427/(26^5)

prob
```

```
## [1] 0.0004567653
```

```
# 0.0004567653, ou seja, 0,04567653%.

letras$prob <- seq(1, 26, by = 1) / 26
```

Resultado com processos iterativos:

```
probabilidade <- function(){
  contador <- 0
  existe <- FALSE
  while (existe == FALSE) {
    palavra <- NULL
    for (i in 1:5){
      u <- runif(1)
      indice <- which.min(abs(letras$prob - u))
      letra <- letras$letras[indice]
      palavra <- paste0(palavra,letra)
```

```
      }
    existe <- any(grepl(palavra, pcl$palavras))
    contador <- contador+1
    }
  rm(u,indice,i,letra,existe,palavra)
  contador
  }

# probabilidade()
# CF <- mean(map_dbl(1:5, ~probabilidade()))

# Como minha função ficou pouco otimizada, vamos paralelizar para melhorar um pouco a velocidade, pe

# Testando as alternativas de paralelização

# tic()
# plan(multicore)
# CF <- mean(future_map_dbl(1:10, ~probabilidade()))
# toc()

# tic()
# plan(multisession, workers = cores)
# CF <- mean(future_map_dbl(1:10, ~probabilidade()))
# toc()

# O multisession foi superior; aumentando o nível de iterações para a forma final:

plan(multisession, workers = cores)
CF <- mean(future_map_dbl(1:100, ~probabilidade()))

1/CF
```

```
## [1] 0.0006457779
```

```
# Pedi ao ChatGPT que otimizasse minha função. Este foi o output:

probabilidade2 <- function(){
  contador <- 0
  existe <- FALSE
  while (!existe) {
    palavra <- paste0(sample(letras$letras, 5, replace = TRUE), collapse = "")
    existe <- any(pcl$palavras == palavra)
    contador <- contador + 1
  }
  contador
}

# probabilidade2()

# De fato, roda bem mais rápido.

# Testando a convergência desta:
# (como ela é mais eficiente, dá para fazer mais iterações)
plan(multisession, workers = cores)
CF2 <- mean(future_map_dbl(1:300, ~probabilidade2()))
1/CF2
```

```
## [1] 0.0004466978
```

```r
# Por algum motivo, a função do ChatGPT converge para o real valor melhor que a minha.
```

b)

Resultado analítico:

```r
# A quantidade de pseudo-palíndromos possíveis para uma palavra de 5 letras é de

# 26*26*26*1*1
# ou:
# 26^3
# = 17576

# Visto que as 3 primeiras letras podem ser quaisquer, mas a penúltima deve ser igual a segunda (1 c

# Ou seja, das 26^5 combinações possíveis, apenas 26^3 serão palíndromos;

(26^3)/(26^5)
```

```
## [1] 0.00147929
```

```r
# = 0.00147929
```

Resultado iterativo:

```r
# Para isso, tentarei aproveitar minha última função (a que eu fiz, não a do ChatGPT), adaptando alg

palindromo <- function(){
  contador <- 0
  check <- FALSE
  while (check == FALSE) {
    palavra <- NULL
    for (i in 1:5){
      u <- runif(1)
      indice <- which.min(abs(letras$prob - u))
      letra <- letras$letras[indice]
      palavra <- paste0(palavra,letra)
      }
    espelho <- paste0(strsplit(palavra, "")[[1]][5:1], collapse = "")
    check <- palavra == espelho
    contador <- contador+1
    }
  rm(u,indice,i,letra,check,palavra)
  contador
  }

# palindromo()

#tic()
plan(multisession, workers = cores)
CFb <- mean(future_map_dbl(1:300, ~palindromo()))
#toc()
1/CFb
```

```
## [1] 0.001506538
```

c)

```
# Definindo vogais e consoantes:
vogais <- c("a","e","i","o","u")
consoantes <- letras$letras[-c(1,5,9,15,21)]

# Construindo a função
gerador <- function(){
  contador <- 0
  check <- FALSE
  while(check==FALSE){
    letra <- sample(letras$letras,1)
    ifelse(letra %in% consoantes, letra2 <- sample(vogais,1),letra2 <- sample(consoantes,1))
    ifelse(letra2 %in% consoantes, letra3 <- sample(vogais,1),letra3 <- sample(consoantes,1))
    ifelse(letra3 %in% consoantes, letra4 <- sample(vogais,1),letra4 <- sample(consoantes,1))
    ifelse(letra4 %in% consoantes, letra5 <- sample(vogais,1),letra5 <- sample(consoantes,1))
    palavra <- paste0(letra,letra2,letra3,letra4,letra5,sep="")
    rm(letra,letra2,letra3,letra4,letra5)
    check <- any(palavra %in% pcl$palavras)
    contador <- contador+1
    }
  check <- FALSE
  contador
}

# gerador()

# Iterando a função algumas vezes:
plan(multisession, workers = cores)
CF3 <- mean(future_map_dbl(1:500, ~gerador()))

# Resultado:
1/CF3
```

```
## [1] 0.007112477
```

d)

```
freq$Frequência <- gsub("%", "", freq$Frequência)
freq$Frequência <- as.numeric(freq$Frequência)
freq$Frequência <- freq$Frequência/100

# Amostrando:

acabou_a_criatividade_para_nome_de_funcao <- function(){
  contador <- 0
  check2 <- FALSE
  while(check2 == FALSE){
    letras <- sample(x=freq$Letra,size=5,prob = freq$Frequência)
    palavra <- paste0(letras,collapse="")
    check1 <- grepl("a", palavra)
    if(check1 == TRUE){
      check2 <- palavra %in% pcl$palavras
      }
    contador <- contador+1
    }
  contador
  }
```

```r
# acabou_a_criatividade_para_nome_de_funcao()

# Iterando a função algumas vezes:
plan(multisession, workers = cores)
CF4 <- mean(future_map_dbl(1:500, ~acabou_a_criatividade_para_nome_de_funcao()))

# Resultado:
1/CF4
```

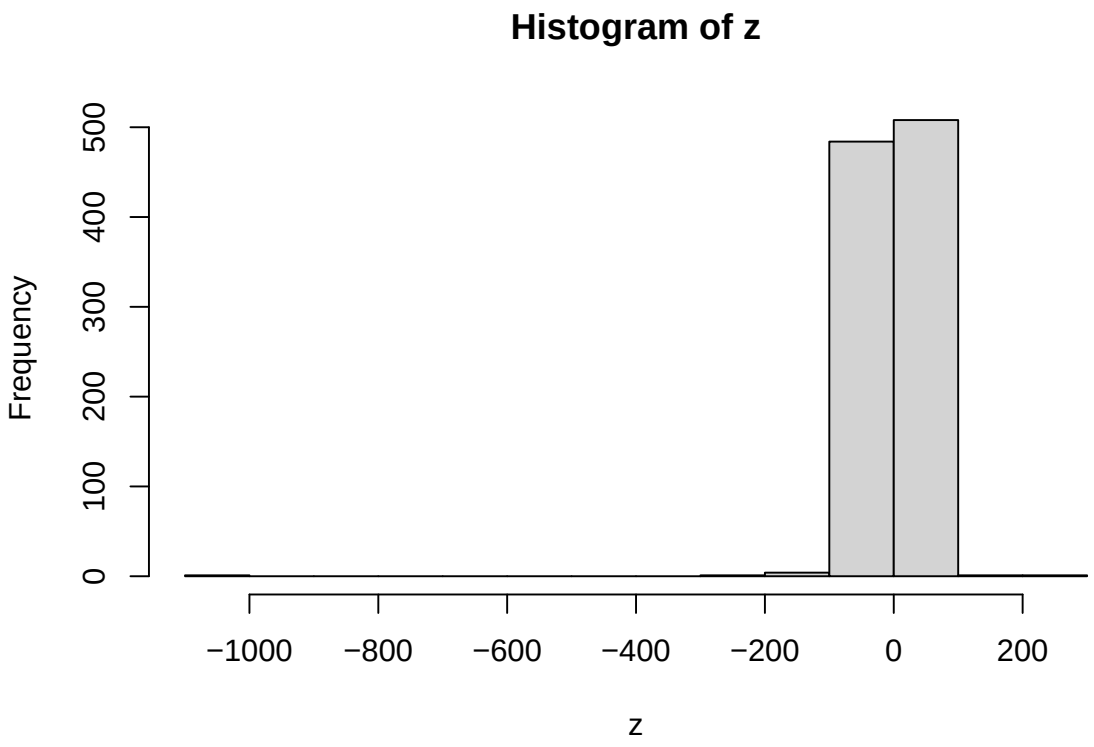
```
## [1] 0.006409846
```

Quanto à demonstração analítica, *Cuius rei demonstrationem mirabilem sane detexi hanc marginis exiguitas non caperet*

2)

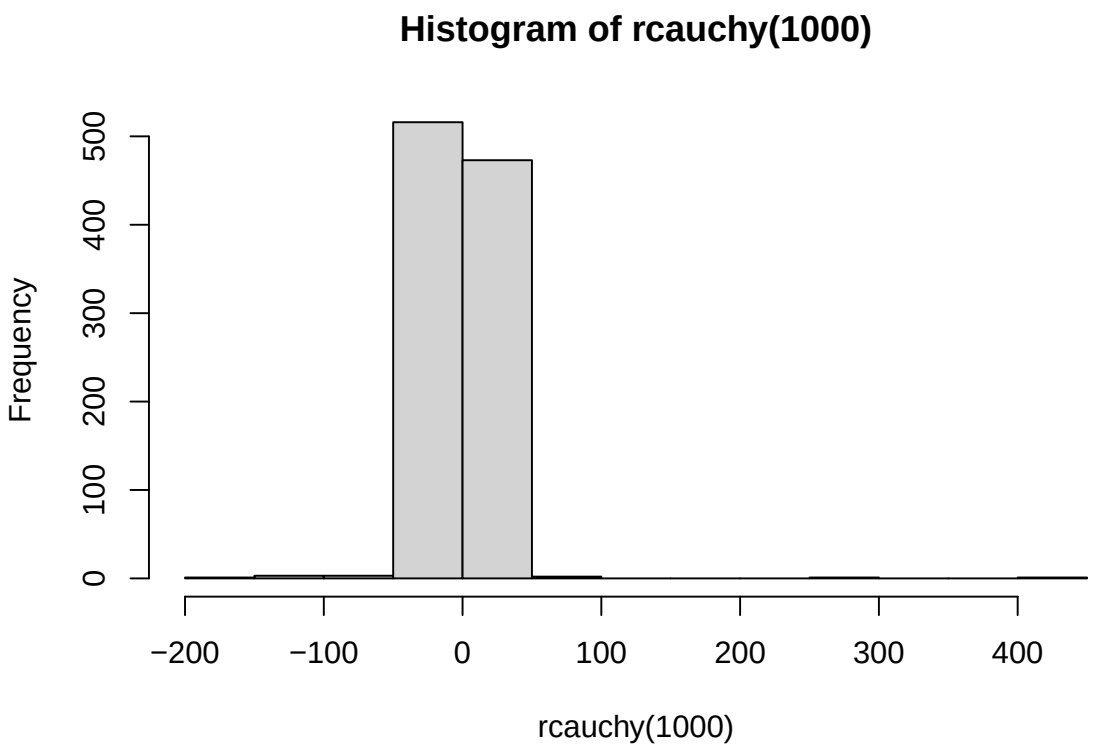
a)

```r
# usando a função quantílica (assim como no caso da exponencial do slide); e fixando gama=1;
n<-1000
u<-runif(n)
z <- tan(pi * (u - 0.5))
hist(z)
```




```r
#comparando com a distribuição "real"

hist(rcauchy(1000))
```




```
# Aparenta estar correto..
```

b)

```
# Usando a runif
u <- runif(1000)
x <- case_when(
  u < 0.2 ~ 2,
  u >= 0.2 & u < 0.3 ~ 3,
  u >= 0.3 & u < 0.5 ~ 5,
  u >= 0.5 & u < 0.7 ~ 7,
  u >= 0.7 ~ 9)

# Gerando a tabela de frequências relativas observadas; após inserindo a frequência esperada
df <- data.frame(factor(x)) %>%
  rename(valor = 1)%>%
  group_by(valor) %>%
  summarise(n=n(),freq_observada = n/1000)
df$freq_esperada <- c(.2,.1,.2,.2,.3)

# Usando a função sample:
df2 <- sample(x=c(2,3,5,7,9),size=1000,prob=c(.2,.1,.2,.2,.3),replace=T)

# Criando a tabela dos numeros gerados pela segunda função

df2 <- data.frame(factor(df2)) %>%
  rename(valor = 1)%>%
  group_by(valor) %>%
  summarise(n=n(),freq_observada = n/1000)
df2$freq_esperada <- c(.2,.1,.2,.2,.3)
```

```
kable(df)
```

valor	n	freq_observada	freq_esperada
2	208	0.208	0.2
3	97	0.097	0.1
5	198	0.198	0.2
7	213	0.213	0.2
9	284	0.284	0.3

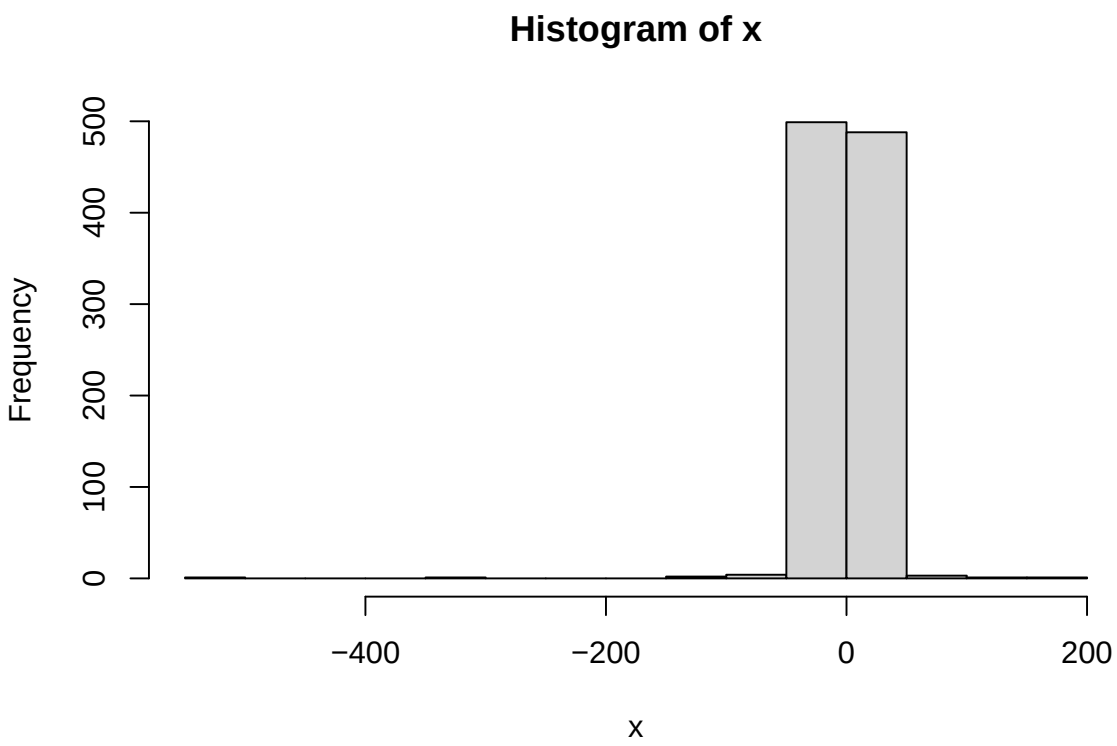
```
kable(df2)
```

valor	n	freq_observada	freq_esperada
2	219	0.219	0.2
3	89	0.089	0.1
5	190	0.190	0.2
7	203	0.203	0.2
9	299	0.299	0.3

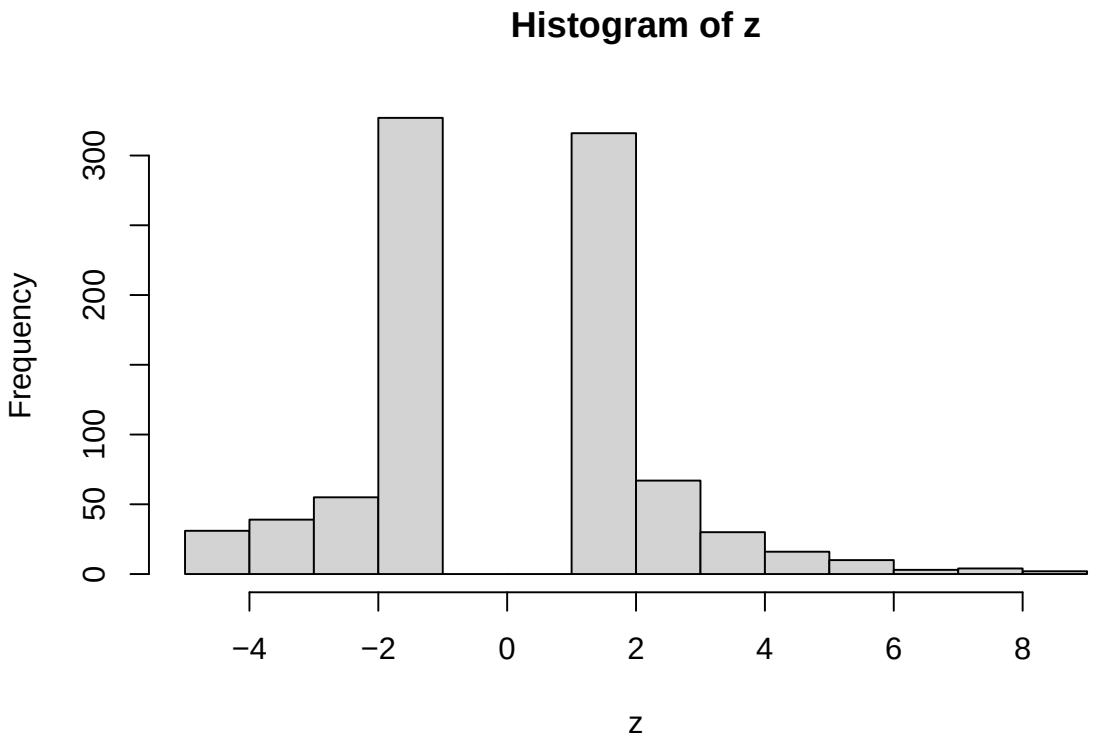
c) Preciso arrumar essa ainda, tá bem ruim

```
x <- rcauchy(1000)
y <- (x^2 + 1) / (2 * x)
z <- qnorm(pnorm(y))
# Como a dist. Cauchy tem média infinita, irei manualmente remover os valores gerados > 5
hist(z)
```

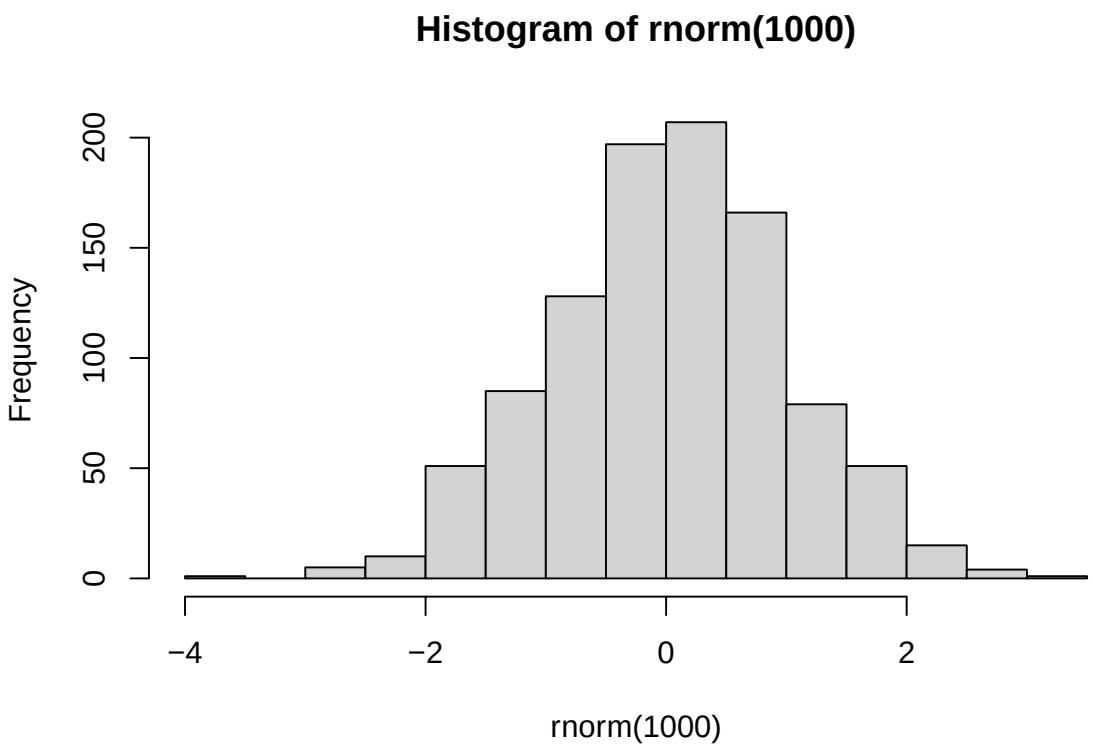

```
z <- z[(z > 5 | z > -5)]
z <- z[!is.infinite(z)]
hist(x)
```




```
hist(z)
```



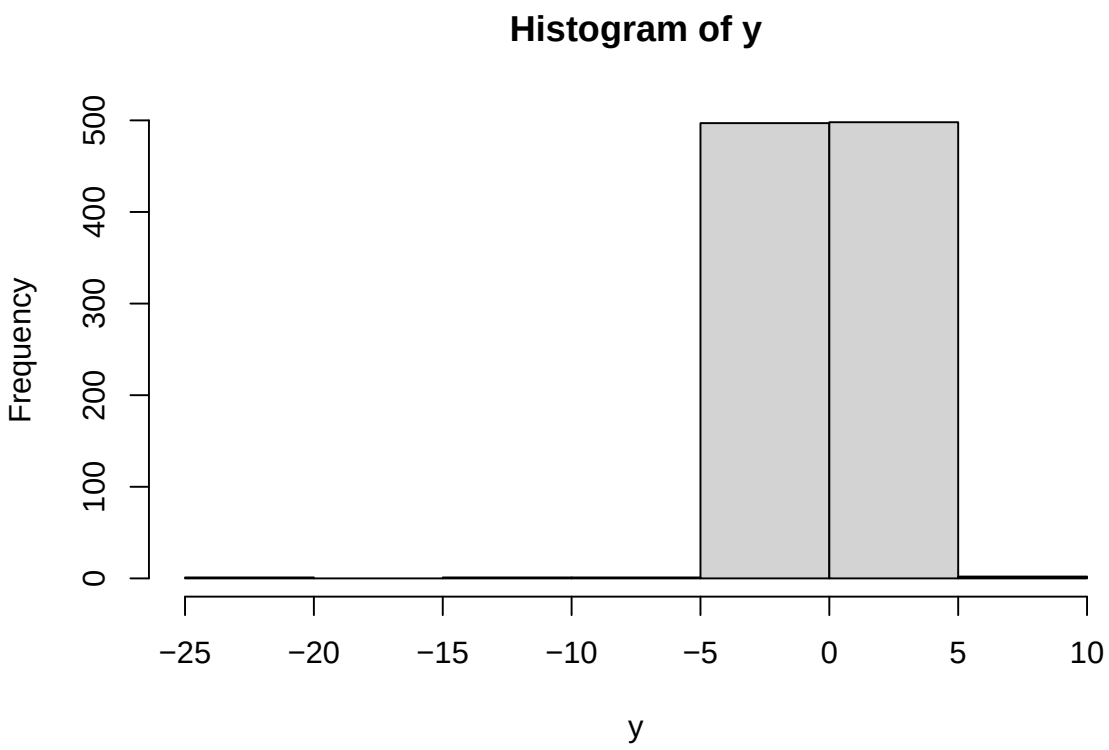

```
hist(rnorm(1000))
```




```
y <- (x - median(x)) / sd(x)
z <- qnorm((atan(y / pi) + 0.5))
```

```
## Warning in qnorm((atan(y/pi) + 0.5)): NaNs produzidos
```

```
hist(y)
```


Histogram of y
Frequency
0
100
200
300
400
500
−25
−20
−15
−10
−5
0
5
10
y