



DEPARTAMENTO DE ESTATÍSTICA

02 maio 2023

Lista 1

Prof. Dr. Donald Matthew Pianto

Aluno: Bruno Gondim Toledo

Matrícula: 15/0167636

Estatística Computacional

1º/2023

```

if (!require("pacman")) install.packages("pacman")

## Carregando pacotes exigidos: pacman
p_load(knitr,tidyverse,doParallel,furrr,tictoc,ggpubr)

cores <- detectCores()

```

Questão 1

```

dicionario <- read_csv("dados/Dicionario.txt",
  col_names = FALSE)

## Rows: 245366 Columns: 1
## -- Column specification -----
## Delimiter: ","
## chr (1): X1
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
colnames(dicionario) <- "palavras"

```

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 exercício cita que iremos trabalhar apenas com combinações de letras sem inclusão de caractér

pcl <- pcl %>%
  distinct()

# 5481-5427

# Com isso, estaremos trabalhando com 54 palavras a menos (palavras idênticas 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 de

# 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 válida

```

```

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 <- future_map_dbl(1:1000, ~probabilidade())

1/mean(CF)

## [1] 0.0006593583

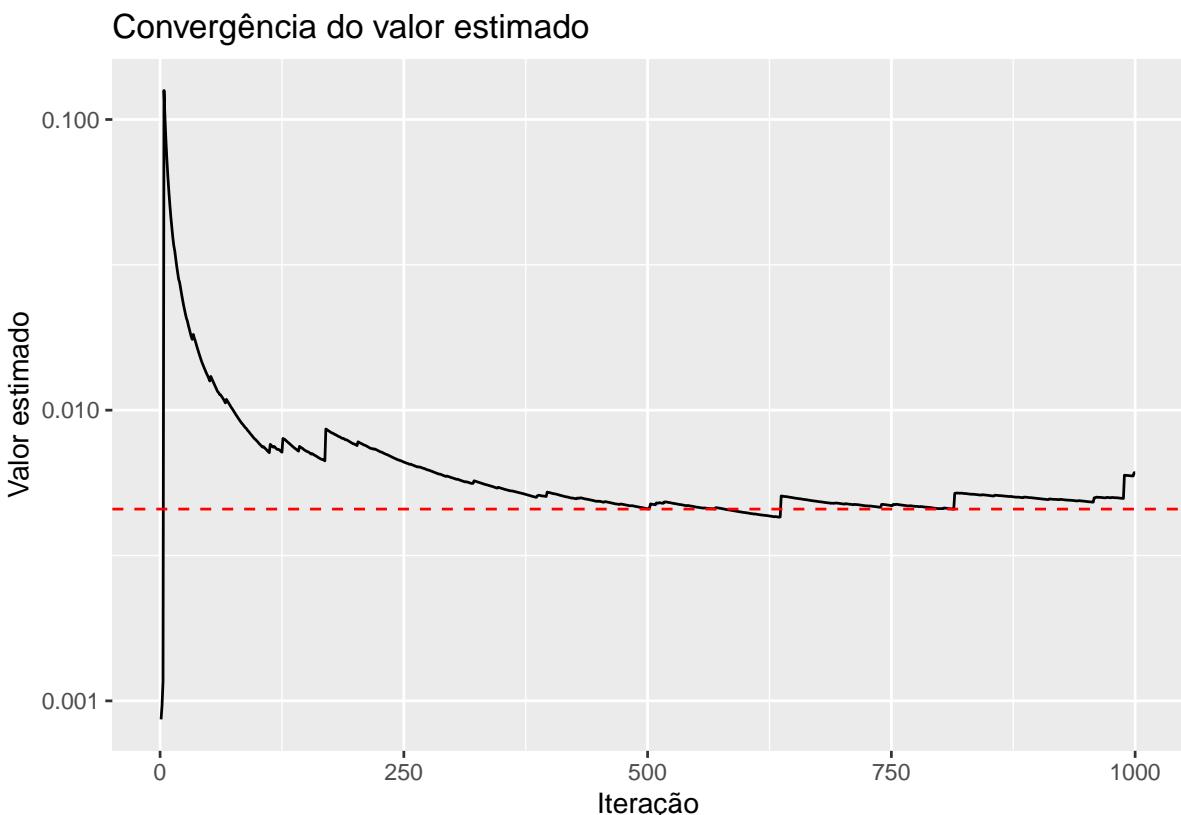
df <- data.frame(iteracao = 1:1000, probabilidade = sapply(CF, function(x) 1/x))
df$media_acumulada <- cumsum(df$probabilidade) / seq_along(df$probabilidade)
df$valor_real <- prob*10

```

```

ggplot(df, aes(iteracao, media_acumulada)) +
  geom_line() +
  geom_hline(yintercept = 0.004567653, linetype = "dashed", color = "red") +
  xlab("Iteração") +
  ylab("Valor estimado") +
  ggtitle("Convergência do valor estimado") +
  scale_y_log10()

```



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()

#tic()
#plan(multisession, workers = cores)
#CF <- mean(future_map_dbl(1:10, ~probabilidade2()))
#toc()

# De fato, roda bem mais rápido.

```

```

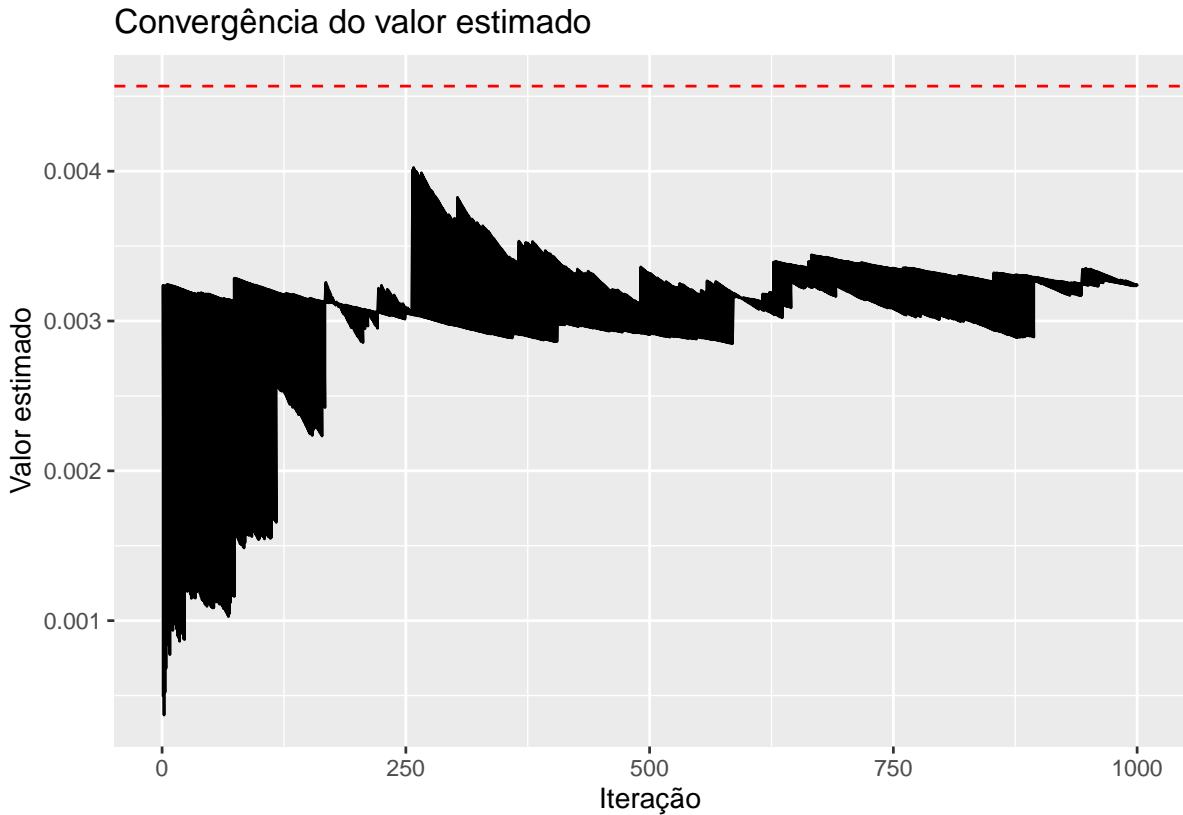
# Testando a convergência desta:
# (como ela é mais eficiente, dá para fazer mais iterações)
plan(multisession, workers = cores)
CF2 <- future_map_dbl(1:2000, ~probabilidade2())
1/mean(CF2)

## [1] 0.0004443064

df2 <- data.frame(iteracao = 1:1000, probabilidade = sapply(CF2, function(x) 1/x))
df2$media_acumulada <- cumsum(df2$probabilidade) / seq_along(df2$probabilidade)
df2$valor_real <- prob*10

ggplot(df2, aes(iteracao, media_acumulada)) +
  geom_line() +
  geom_hline(yintercept = 0.004567653, linetype = "dashed", color = "red") +
  xlab("Iteração") +
  ylab("Valor estimado") +
  ggtitle("Convergência do valor estimado")

```



b)

Resultado analítico:

```

# 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
# = 0.00147929
```

Resultado iterativo:

```
# 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(1:2000, ~palindromo()))
#toc()
1/CFb

## [1] 0.001476006
```

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:
#tic()
plan(multisession, workers = cores)
CF3 <- mean(future_map_dbl(1:2000, ~gerador()))
#toc()
# Resultado:
1/CF3

## [1] 0.006825939

```

d)

```

# Definindo o vetor de probabilidades, segundo as informações da Wikipedia
freq <- read_delim("dados/freq.txt",
  delim = "\t", escape_double = FALSE)

## Rows: 26 Columns: 2
## -- Column specification -----
## Delimiter: "\t"
## chr (2): Letra, Frequência
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
colnames(freq) <- c("letra", "freq")

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

# Amostrando:

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

# acabou_a_criatividade_para_nome_de_funcao()

```

```

# Iterando a função algumas vezes:
#tic()
plan(multisession, workers = cores)
CF4 <- mean(future_map_dbl(1:2000, ~acabou_a_criatividade_para_nome_de_funcao()))
#toc()
# Resultado:
1/CF4

## [1] 0.01012556

```

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

2)

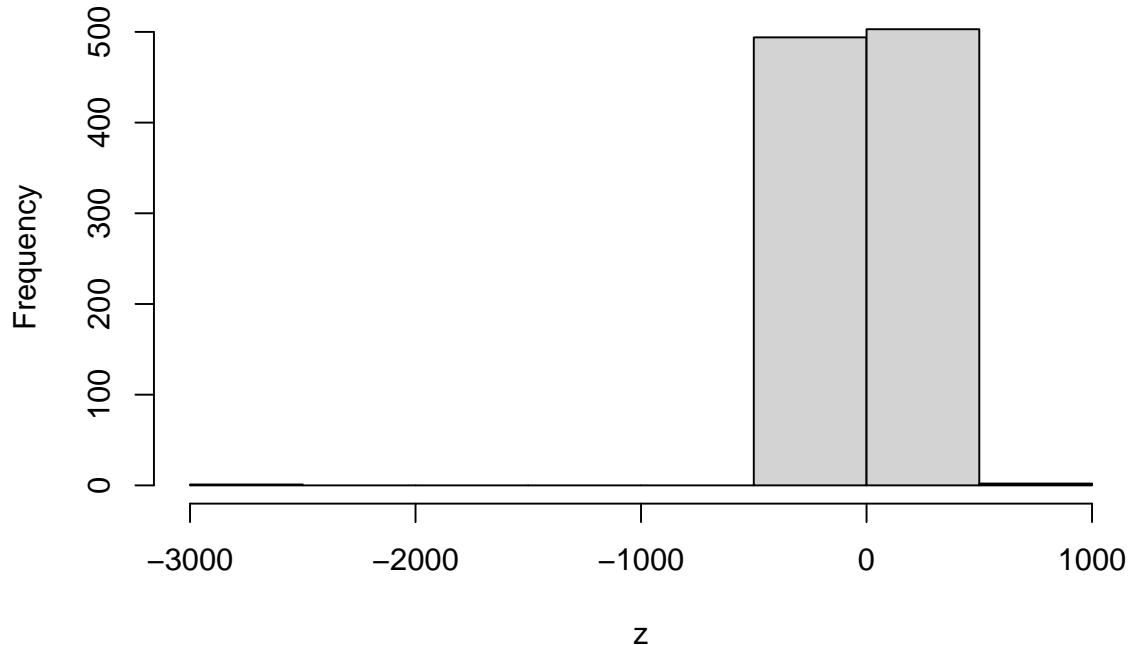
a)

```

# 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)

```

Histogram of z

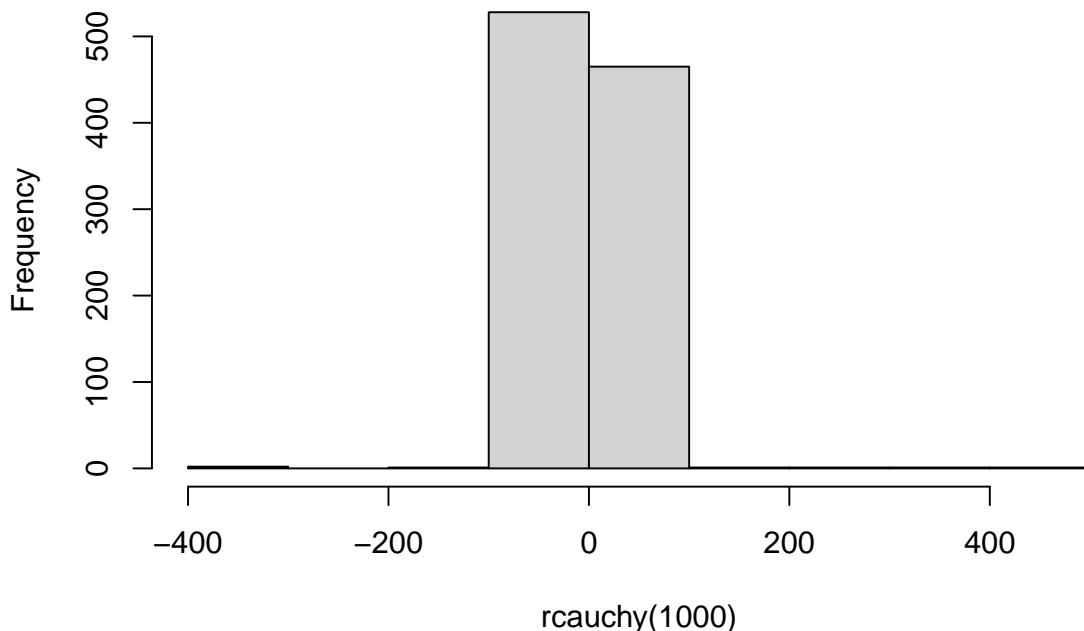


```

#comparando com a distribuição "real"
hist(rcauchy(1000))

```

Histogram of 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	199	0.199	0.2
3	109	0.109	0.1
5	192	0.192	0.2
7	180	0.180	0.2
9	320	0.320	0.3

```
kable(df2)
```

valor	n	freq_observada	freq Esperada
2	213	0.213	0.2
3	99	0.099	0.1
5	199	0.199	0.2
7	173	0.173	0.2
9	316	0.316	0.3

c)

```
# Irei tentar implementar o método de Rejeição de Von Neumann, tentando gerar valores de uma distribuição uniforme entre 0 e 1.
```

```
f <- function(x) dnorm(x, 0, 1)
g <- function(x) dcauchy(x, 0, 1)

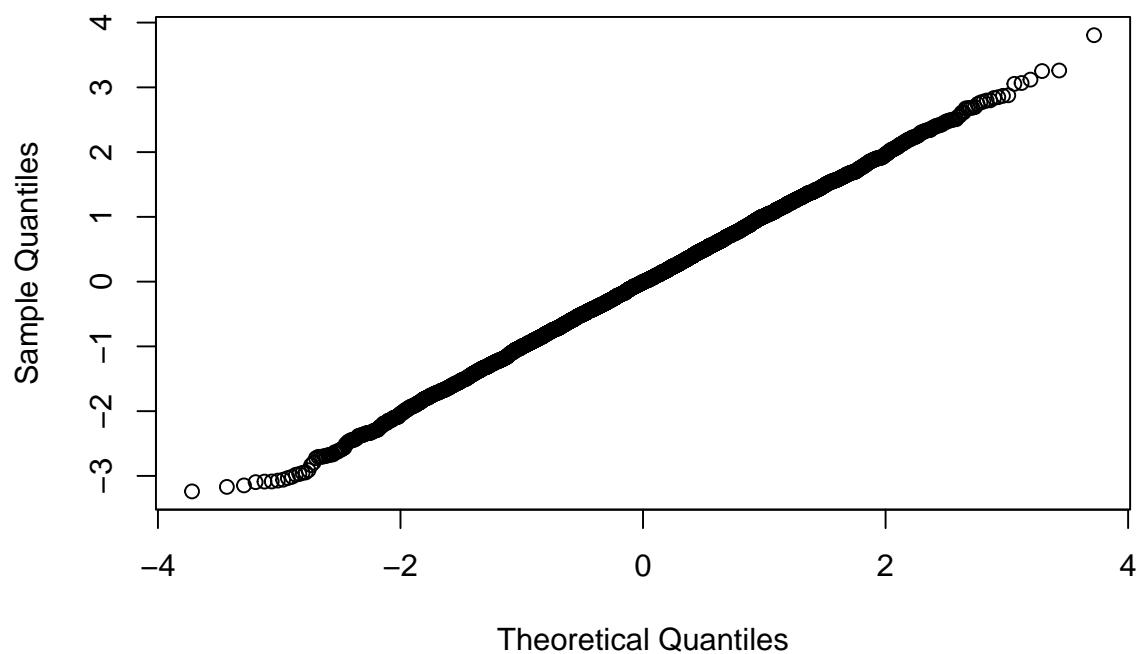
C <- sqrt(2 * pi/exp(1)) # Ajustando uma constante C para ajustar a escala da distribuição cauchy para que f(x) >= g(x)

N <- 5000 # Definindo o número de observações
x <- numeric(N)
i <- 1

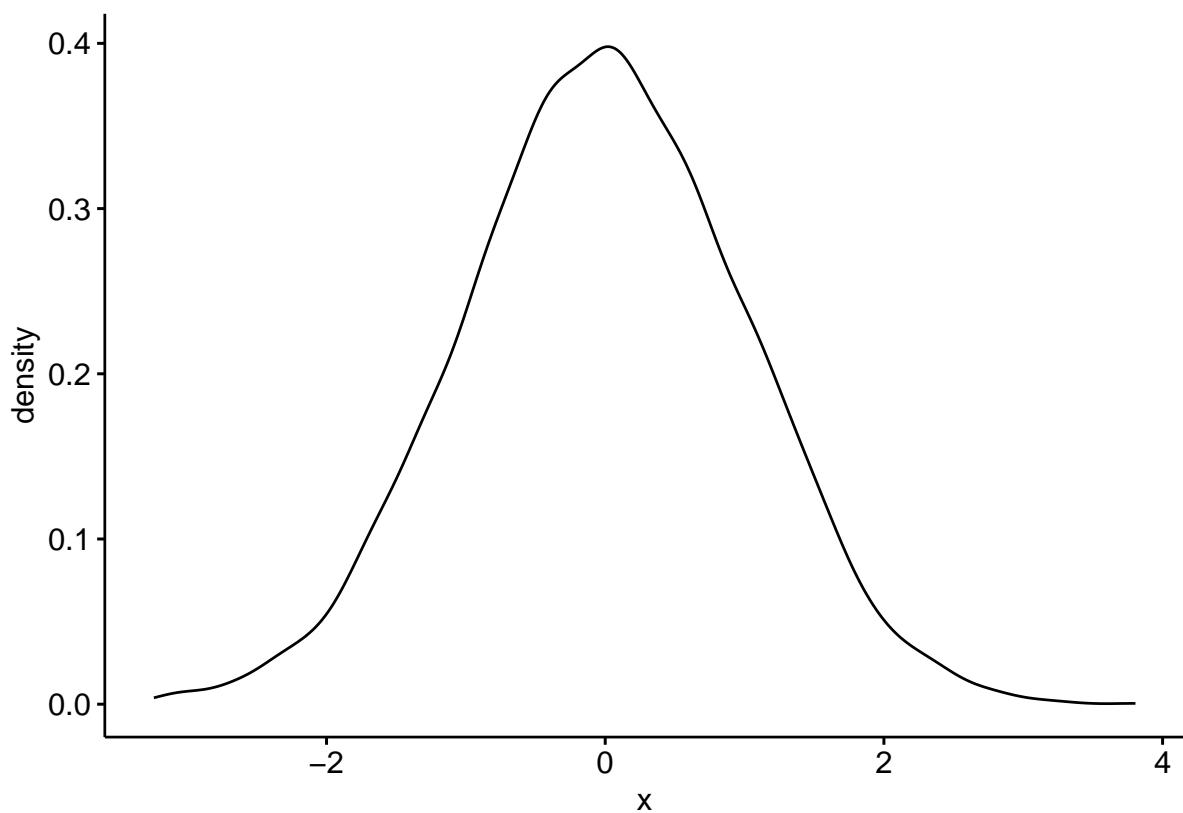
while (i <= N) {
  y <- rcauchy(n = 1)
  u <- runif(n = 1)
  if (u < f(y)/(C * g(y))) {
    x[i] <- y
    i <- i + 1
  }
}

qqnorm(x)
```

Normal Q–Q Plot



```
ggdensity(x)
```



```
shapiro.test(x)
```

```
##
```

```

## Shapiro-Wilk normality test
##
## data: x
## W = 0.9997, p-value = 0.7094
# Os testes não rejeitam a hipótese nula de normalidade da variável.

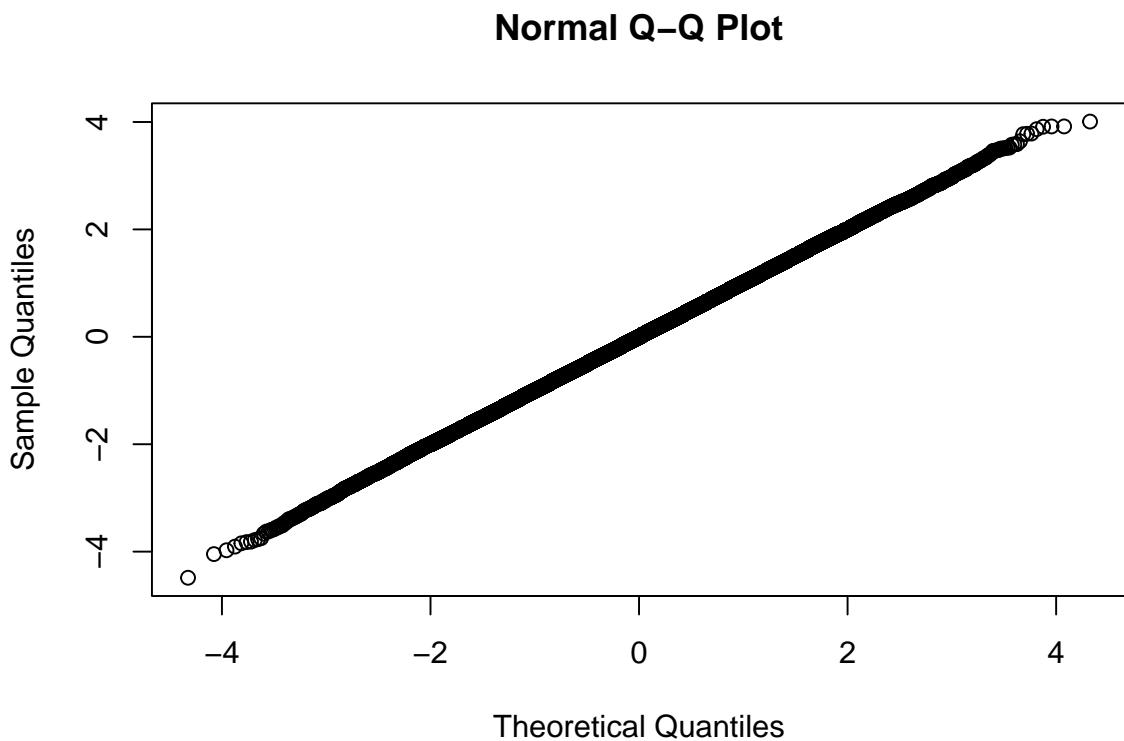
# Fazendo a versão paralelizada, para testar com um conjunto maior de pontos.

set.seed(150167636)
#tic()
x <- future_map_dbl(.x = 1:100000, ~{
  y <- rcauchy(n = 1)
  u <- runif(n = 1)
  if (u < dnorm(y, 0, 1)/(C * dcauchy(y, 0, 1))) {
    y
  } else {
    NA_real_
  }
})
x <- na.omit(x)
#toc()

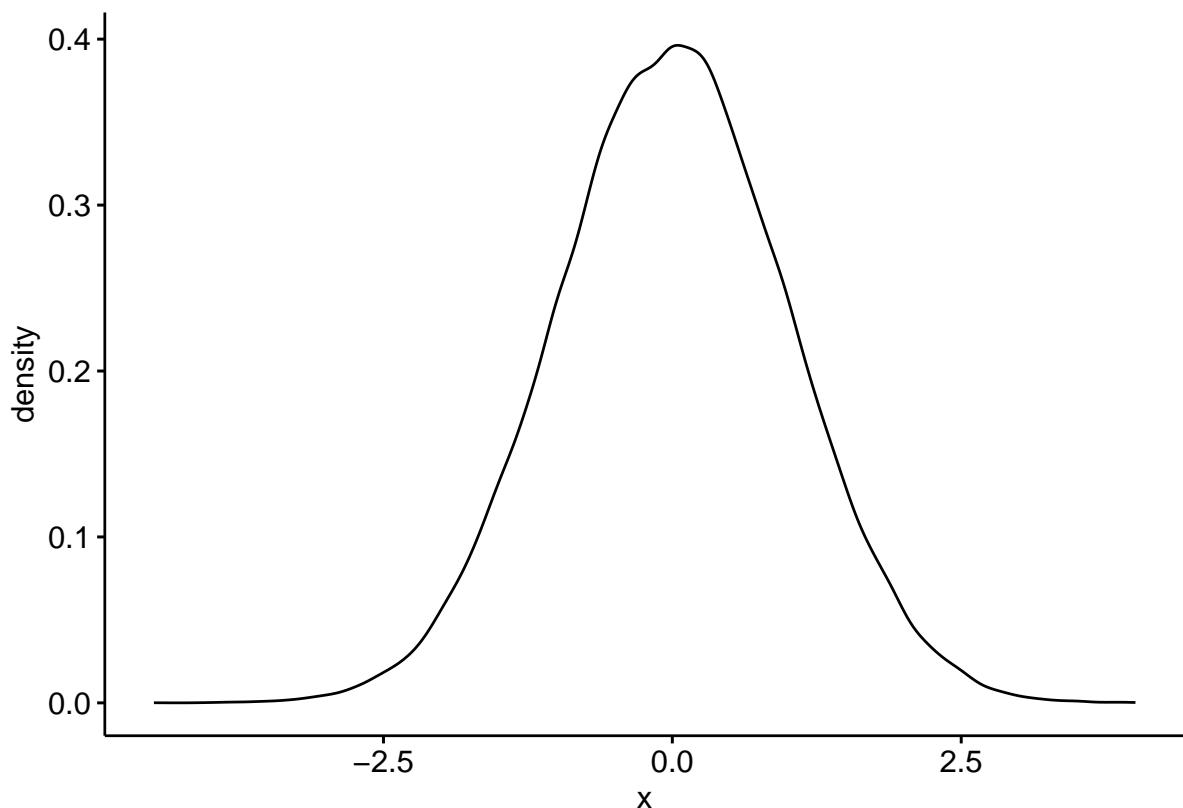
# Neste caso não é possível executar o teste Shapiro-Wilk pela quantidade de
# observações ser muito grande. Então ficaremos apenas com as representações visuais

qqnorm(x)

```



```
ggdensity(x)
```



Pelos gráficos Q-Q e densidade observada, aparenta ter funcionado.