Lista 3 Computacional

Davi Wentrick Feijó

2023-07-19

1) Rizzo – 3.14

```
# Media e matriz de covariancia
mu <- c(0, 1, 2)  # Vetor médio com os valores médios para cada variável
Sigma <- matrix(c(1.0, -0.5, 0.5,
                  -0.5, 1.0, -0.5,
                  0.5, -0.5, 1.0), nrow = 3)  # Matriz de covariância

# Gerando observacoes aleatorias
n <- 1000  # Número de observações a serem geradas


rmvn.cholesky <-  # Definição da função para gerar observações aleatórias
  function(n, mu, Sigma) {
    p <- length(mu)  # Número de variáveis
    Q <- chol(Sigma)  # Decomposição de Cholesky da matriz de covariância
    Z <- matrix(rnorm(n*p), nrow=n, ncol=p)  # Amostras aleatórias da distribuição normal padrão
    X <- Z %*% Q + matrix(mu, n, p, byrow=TRUE)  # Transformação das amostras para a distribuição desej
    X  # Retorna as observações aleatórias
  }

a3 <- rmvn.cholesky(1000, mu, Sigma)  # Geração das observações aleatórias
```

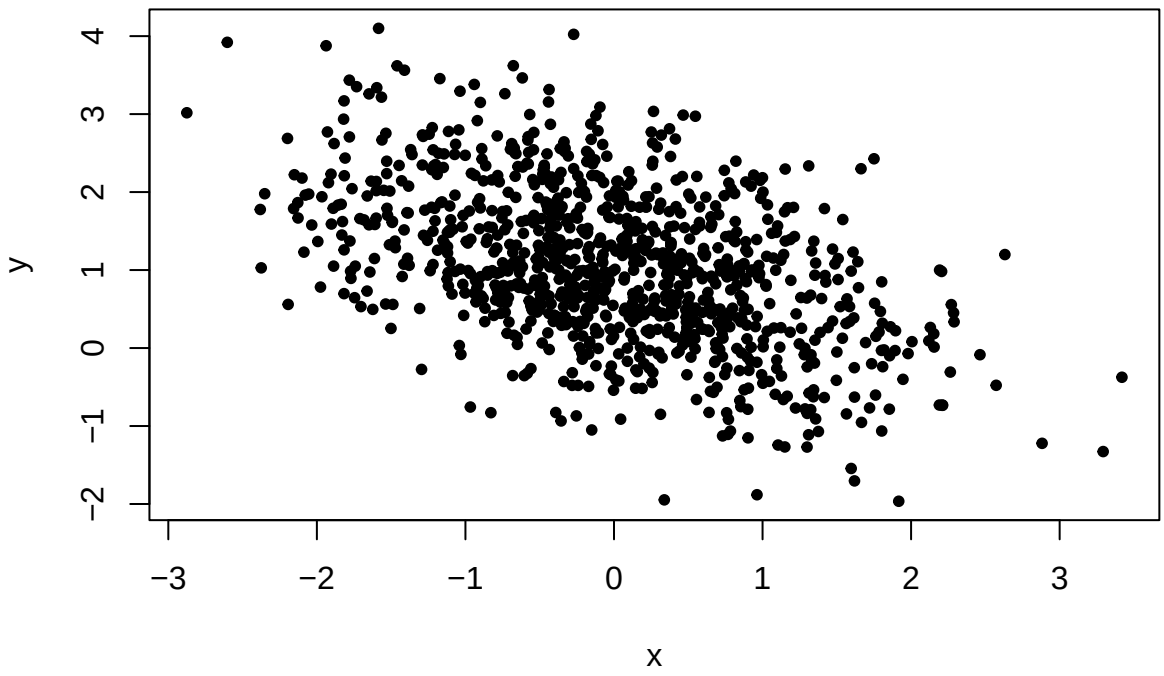
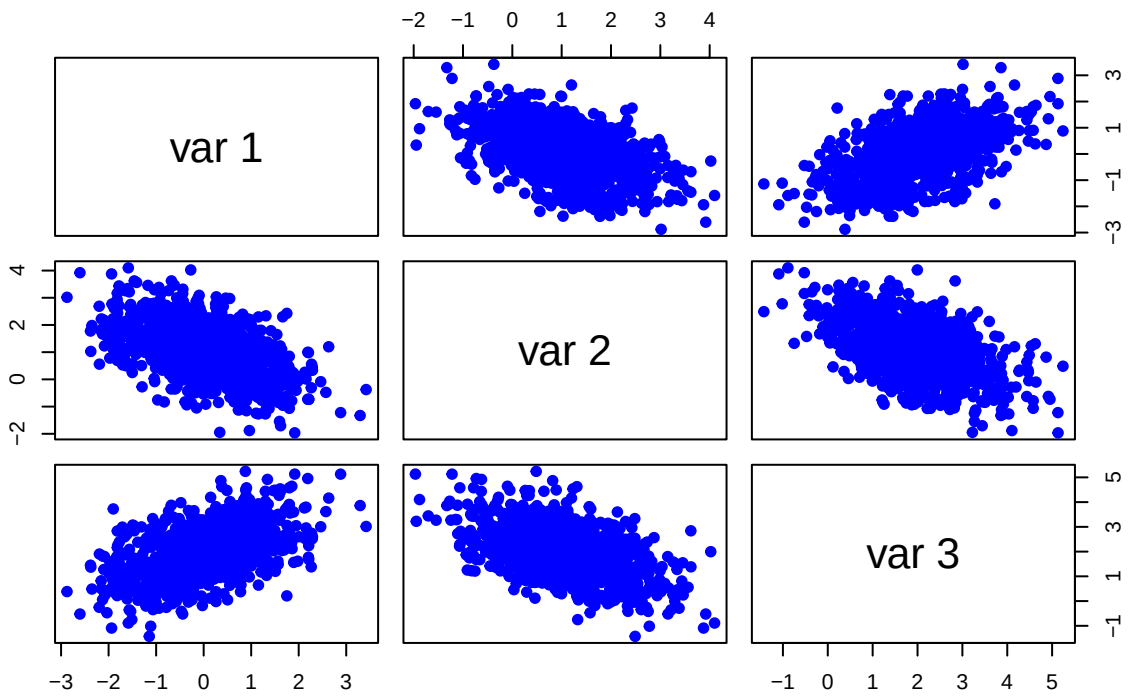

y
4
3
2
1
0
−1
−2
−3
−2
−1
0
1
2
3
x

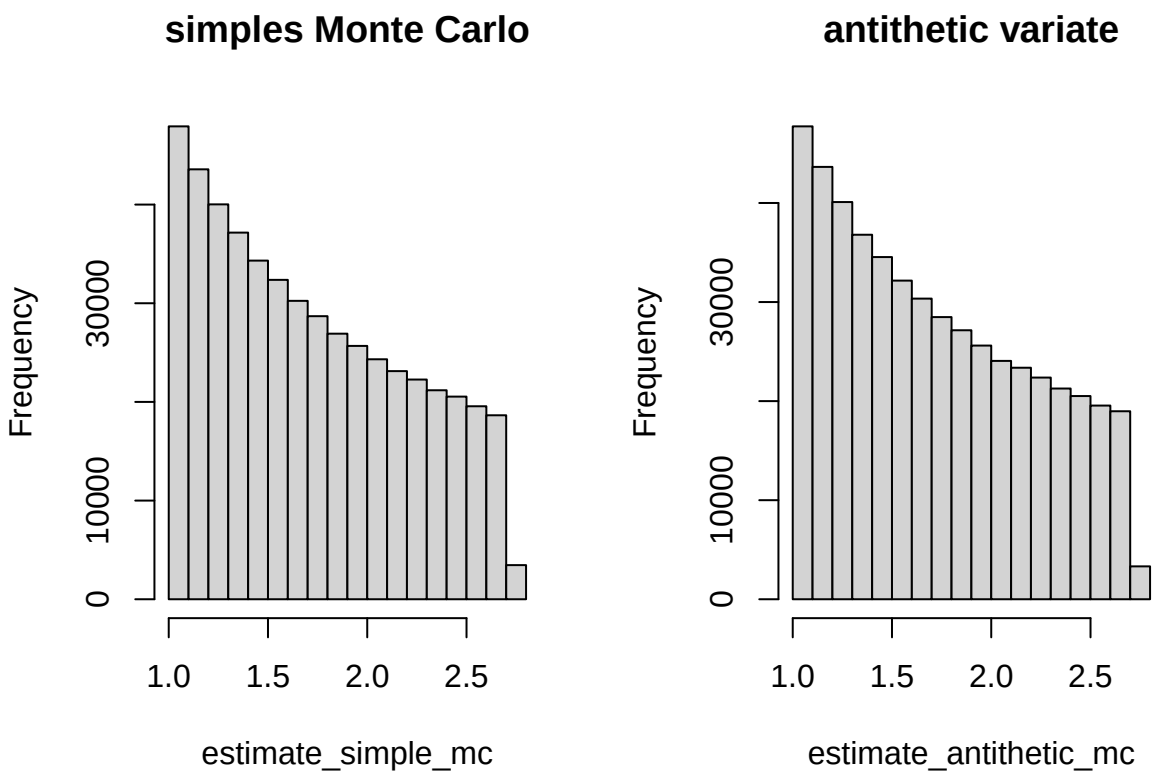
Gráfico de Dispersao




```
## matriz de correlação das observações

##             [,1]       [,2]       [,3]
## [1,]  1.0000000 -0.4843902  0.4975545
## [2,] -0.4843902  1.0000000 -0.5080802
## [3,]  0.4975545 -0.5080802  1.0000000
```

2) Rizzo – 5.7

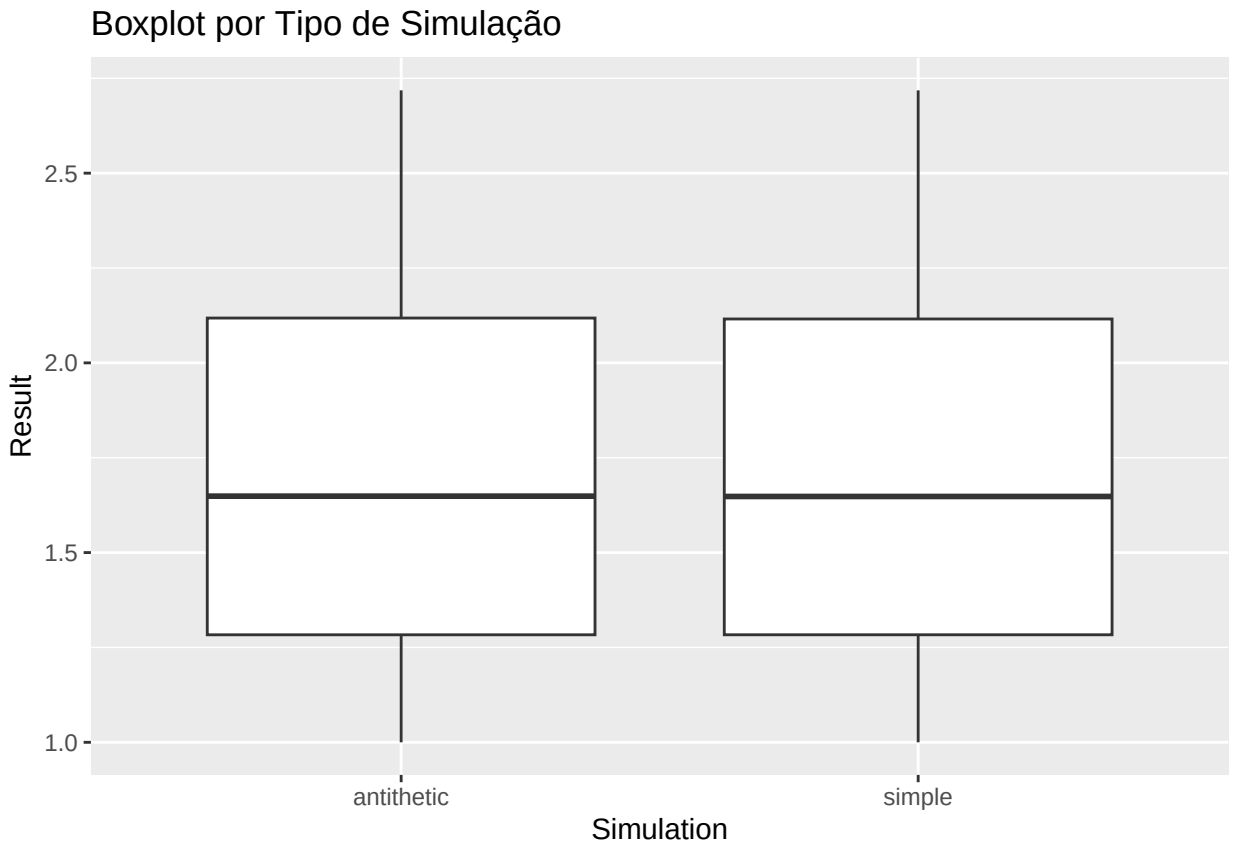
```r
# Set the number of iterations
N <- 100000

# funcao para estimar
funcao <- function(x) {
  return(exp(1)^x)
}

# Simple Monte Carlo method
simple_mc <- function(N) {
  u <- runif(N)  # Gerar N números aleatórios a partir de uma distribuição uniforme
  theta <- funcao(u)  # Estimar  usando a função
  estimate <- mean(theta)  # Calcular a média de
  return(theta)
}

# Antithetic variate method
antithetic_mc <- function(N) {
  u <- runif(N/2)  # Gerar N/2 números aleatórios a partir de uma distribuição uniforme pq vamos estar
  u_antithetic <- 1 - u  # Gerar as variáveis antitéticas
  u_combined <- c(u, u_antithetic)  # Combinar as variáveis
  theta <- funcao(u_combined)  # Estimar usando a função
  estimate <- mean(theta)  # Calcular a média
  return(theta)
}

N=500000

# Estimar usando o método simples de Monte Carlo
estimate_simple_mc <- simple_mc(N)



theta_simple = mean(estimate_simple_mc)
# Estimar usando o método 'antithetic variate'
estimate_antithetic_mc <- antithetic_mc(N)

theta_antithetic = mean(estimate_simple_mc)

# Valor real
valor_real = as.numeric(integrate(funcao, 0, 1)[1])
```




```
## [1] "Theta estimado usando o método simples de Monte Carlo: 1.7174"
```

```
## [1] "Theta  estimado usando o método 'antithetic variate': 1.7174"
```

```
## [1] "Valor real de Theta : 1.7183"
```


Boxplot por Tipo de Simulação
Result
2.5
2.0
1.5
1.0
antithetic
simple
Simulation

3) Validação Cruzada (usa os códigos no final para iniciar)

a. Fixando o cost = 100 (penalidade para furar a margem do SVM), determine via k-fold validação cruzada com k=10, o melhor valor para gamma (a largura de banda do kernel) e reporta esse valor e o erro de teste.

```
# Carregar o conjunto de dados "Glass" do pacote "mlbench"
data(Glass, package = "mlbench")

# Dividir o conjunto de dados em conjunto de treinamento e teste
index <- 1:nrow(Glass)
N <- trunc(length(index) / 3)
set.seed(123)
testindex <- sample(index, N)
testset <- Glass[testindex, ]
trainset <- Glass[-testindex, ]

# Treinar um modelo SVM
svm.model <- svm(Type ~ ., data = trainset, cost = 100, gamma = 0.1)
svm.model
```

```
##
## Call:
## svm(formula = Type ~ ., data = trainset, cost = 100, gamma = 0.1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  100
##
## Number of Support Vectors:  105
```

```
# Fazer previsões nos dados de teste
svm.pred <- predict(svm.model, testset[, -10])
svm.pred
```

```
## 159 207 179  14 195 170  50 118  43 211 213 153  90  91 197 201 185  92 137  99
##   3   7   6   1   7   5   1   2   1   7   7   3   2   3   7   7   2   2   1   2
##  72  26   7 209 196 164  78  81 206 103 117  76 143  32 109 192 190 169  74  23
##   2   1   1   7   7   2   2   2   7   1   2   2   2   1   6   7   2   5   2   1
## 155  53 135 173 174 166  34  69 194 183  63 141  97 199 203  38  21  41 202  60
##   3   1   1   5   5   5   1   1   7   5   1   2   3   7   7   1   2   1   2   1
##  16 116  94   6  86 150  39 204 208 168   4
##   1   2   3   2   3   2   1   7   2   5   2
## Levels: 1 2 3 5 6 7
```

```
# Calcular a matriz de confusão do SVM
matriz_confusao <- table(pred = svm.pred, true = testset[, 10])
matriz_confusao <- as.matrix(matriz_confusao)

# Definir a lista de valores de gamma para avaliar
```

```r
gammas <- c(0.01, 0.1, 1)

# Executar validação cruzada com k-fold
folds <- createFolds(seq(1:6), k = 10)

# Inicializar vetor para armazenar os erros
erros <- numeric()

# Loop sobre os valores de gamma
for (gamma in gammas) {
  erro1 <- numeric()  # Variável para armazenar o erro para cada fold
  erro2 <- numeric()  # Variável para armazenar o erro para cada fold
  erro3 <- numeric()  # Variável para armazenar o erro para cada fold

  # Loop sobre os folds
  for (i in 1:length(folds)) {
    # Dividir os dados em conjunto de treinamento e teste para o fold atual
    train_data <- matriz_confusao[-folds[[i]], ]
    test_data <- t(as.matrix(matriz_confusao[folds[[i]], ]))

    # Extrair as variáveis de entrada (X) e de saída (Y) dos dados de treinamento e teste
    train_X <- train_data[, -ncol(train_data)]
    train_Y <- as.vector(train_data[, ncol(train_data)])
    test_X <- as.vector(test_data[, -ncol(test_data)])
    test_Y <- as.vector(test_data[, ncol(test_data)])

    # Treinar o modelo SVM com o valor de gamma atual
    modelo1 <- svm(Type ~ ., data = trainset, cost = 100, gamma = gammas[1])
    modelo2 <- svm(Type ~ ., data = trainset, cost = 100, gamma = gammas[2])
    modelo3 <- svm(Type ~ ., data = trainset, cost = 100, gamma = gammas[3])

    # Fazer previsões nos dados de teste
    predicao1 <- predict(modelo1)
    predicao2 <- predict(modelo2)
    predicao3 <- predict(modelo3)

    # Calcular a taxa de erro no fold atual
    erro1[i] <- sum(predicao1 != test_Y) / length(test_Y)
    erro2[i] <- sum(predicao2 != test_Y) / length(test_Y)
    erro3[i] <- sum(predicao3 != test_Y) / length(test_Y)

  }

  # Calcular o erro médio para o valor de gamma atual
  avg_erro <- c(mean(erro1), mean(erro2), mean(erro3))
  erros <- c(erros, avg_erro)
}
```

b. Agora otimiza o custo e gamma simultaneamente usando a mesma validação cruzada. Reporta os valores do custo, do gamma e do erro de teste.

```
# Encontrar o valor de gamma com o menor erro
melhor_gamma <- gammas[which.min(erros)]
melhor_erro <- min(erros)
```

```
## Melhor valor de gamma: 0.1
```

```
## Erro de teste correspondente: 140.8333
```

4) Rizzo – 9.3 e 9.7

9.3)

```r
### 9.3)

# Criar o vetor de valores de x
x <- seq(-10, 10, length.out = 1000)

# Definir a função densidade da distribuição Cauchy
densidade_cauchy <- function(x, locacao, escala) {
  return(1 / (pi * escala * (1 + ((x - locacao) / escala)^2)))
}

# Definir os parâmetros da distribuição Cauchy
locacao <- 0  # Parâmetro de localização
escala <- 1  # Parâmetro de escala = teta

# Número de iterações
N <- 10000

# Número de amostras descartadas
descarte <- 1000

# Iniciar a corrente
chain <- numeric(N + descarte)
chain[1] <- 0  # Valor inicial para a corrente

# Distribuição proposta inicialmente (e.g., distribuição normal)
dp_prop <- 1  # Desvio padrão da distribuição proposta inicialmente

# Definir a densidade alvo (Cauchy padrão)
target_density <- function(x) {
  return(1 / (pi * (1 + x^2)))
}

# Algoritmo de Metropolis-Hastings
for (i in 2:(N + descarte)) {
  # Gerar uma amostra candidata da distribuição proposta
  candidato <- rnorm(1, mean = chain[i - 1], sd = dp_prop)

  # Calcular a taxa de aceitação
  acceptance_ratio <- target_density(candidato) / target_density(chain[i - 1])

  # Aceitar ou rejeitar a amostra candidata
  if (runif(1) < acceptance_ratio) {
    chain[i] <- candidato  # Aceitar o candidato
  } else {
    chain[i] <- chain[i - 1]  # Rejeitar o candidato e manter o valor anterior
  }
}

# Descartar as amostras
```

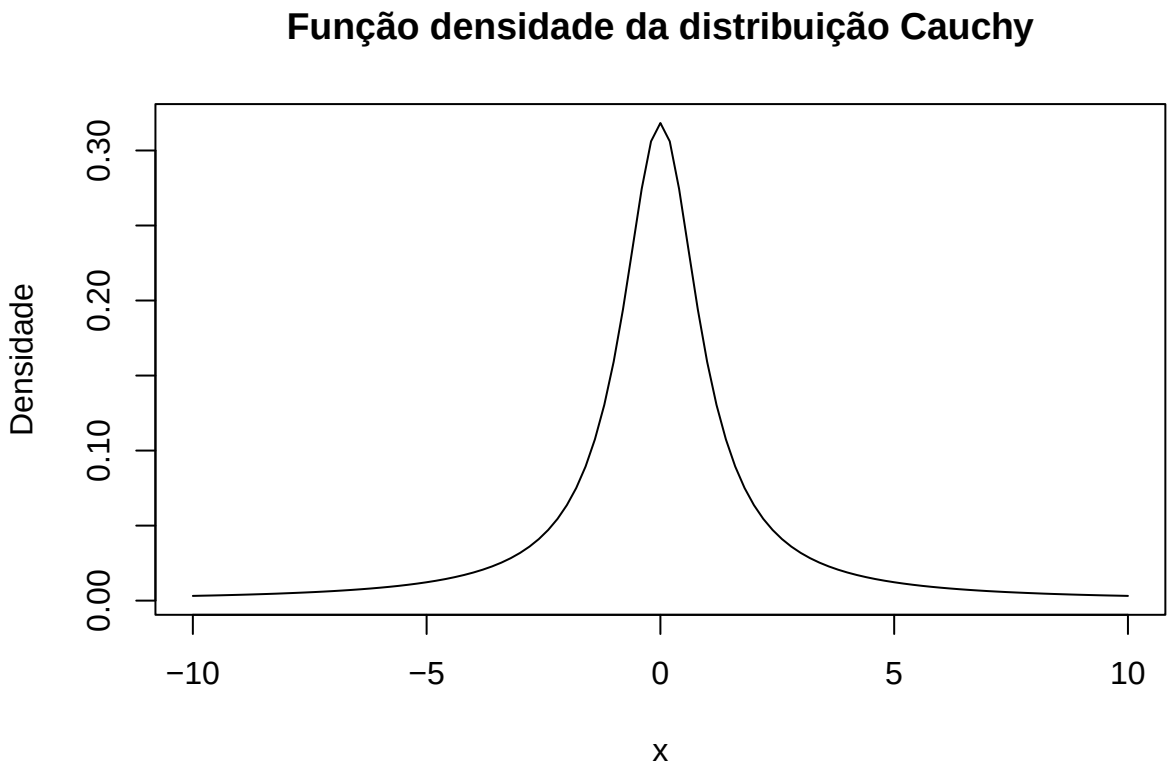
```r
chain <- chain[(descarte + 1):(N + descarte)]

# Calcular os quantis gerados
decis_gerado <- quantile(chain, probs = seq(0.1, 0.9, by = 0.1))

# Calcular os quantis da distribuição Cauchy
decis_cauchy <- qcauchy(seq(0.1, 0.9, by = 0.1))
```

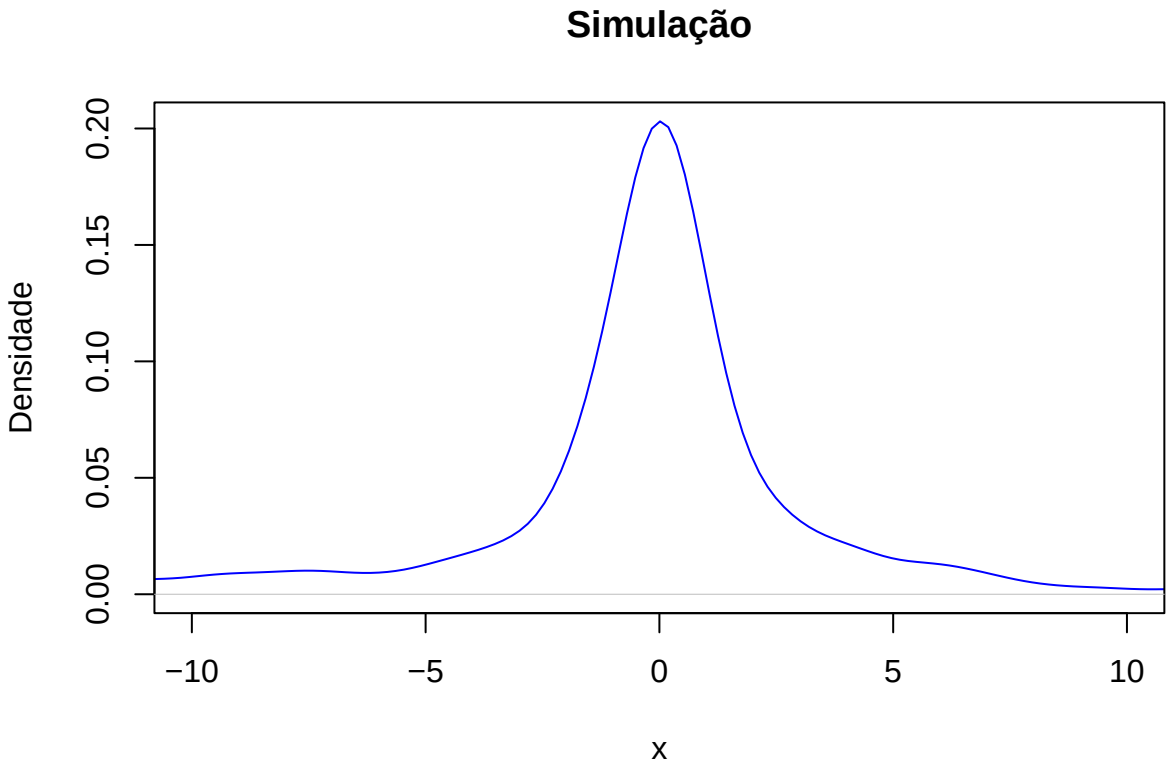
```
## [1] "Decis das observações geradas:"
```

```
##          10%          20%          30%          40%          50%          60%
## -44.61895597  -9.58952623  -2.29295756  -0.96790351  -0.36896839   0.03955857
##          70%          80%          90%
##   0.47221609   1.10019067   2.81545254
```

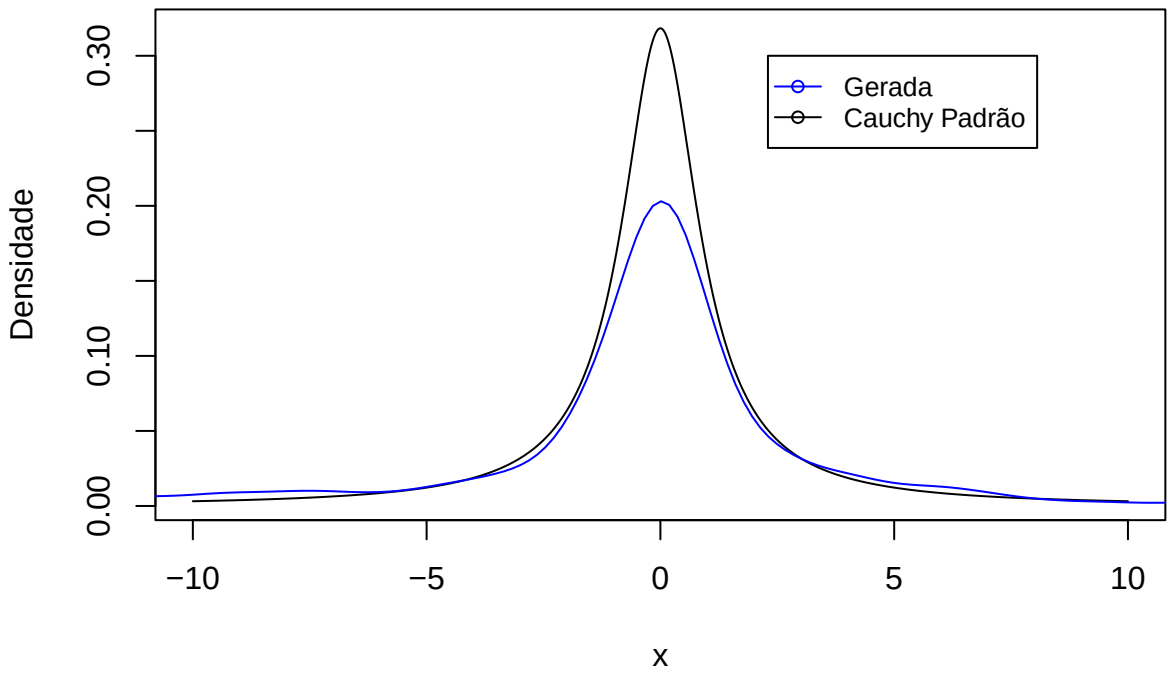
```
## [1] "Decis da Cauchy padrão:"
```

```
## [1] -3.0776835 -1.3763819 -0.7265425 -0.3249197  0.0000000  0.3249197  0.7265425
## [8]  1.3763819  3.0776835
```





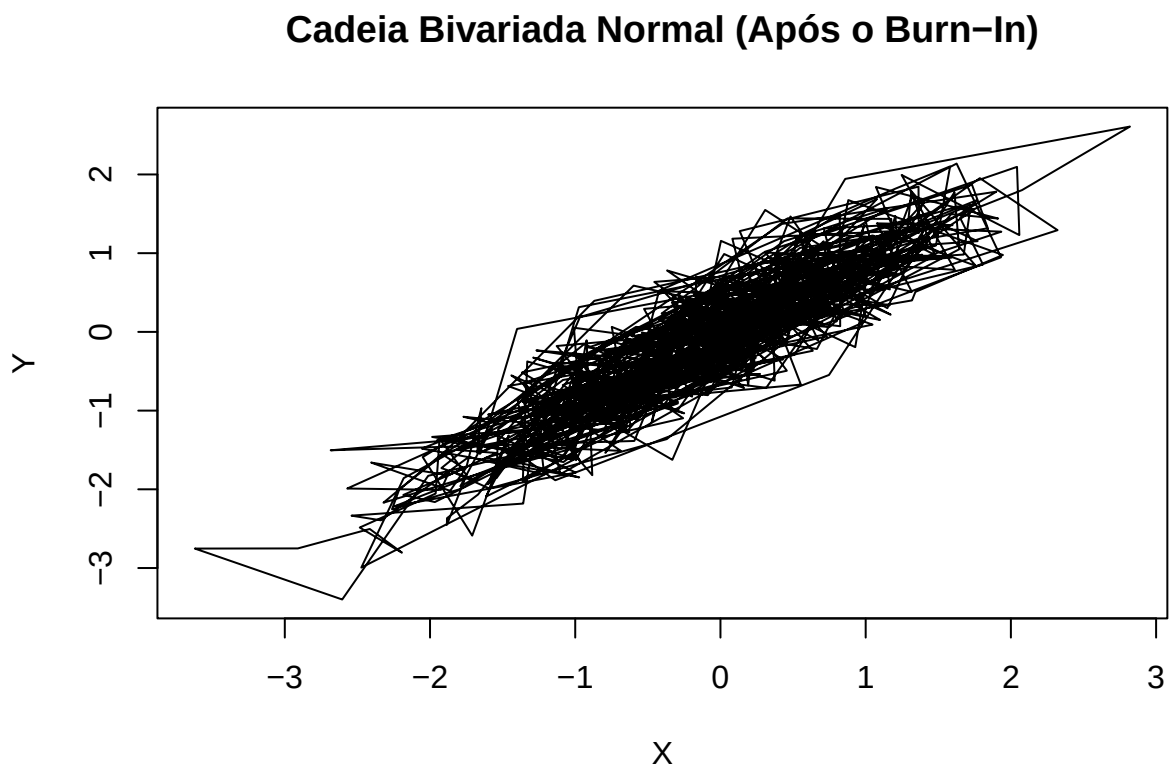

```
# Calcular os valores da função densidade para cada valor de x
densidades <- densidade_cauchy(x, locacao, escala)
```



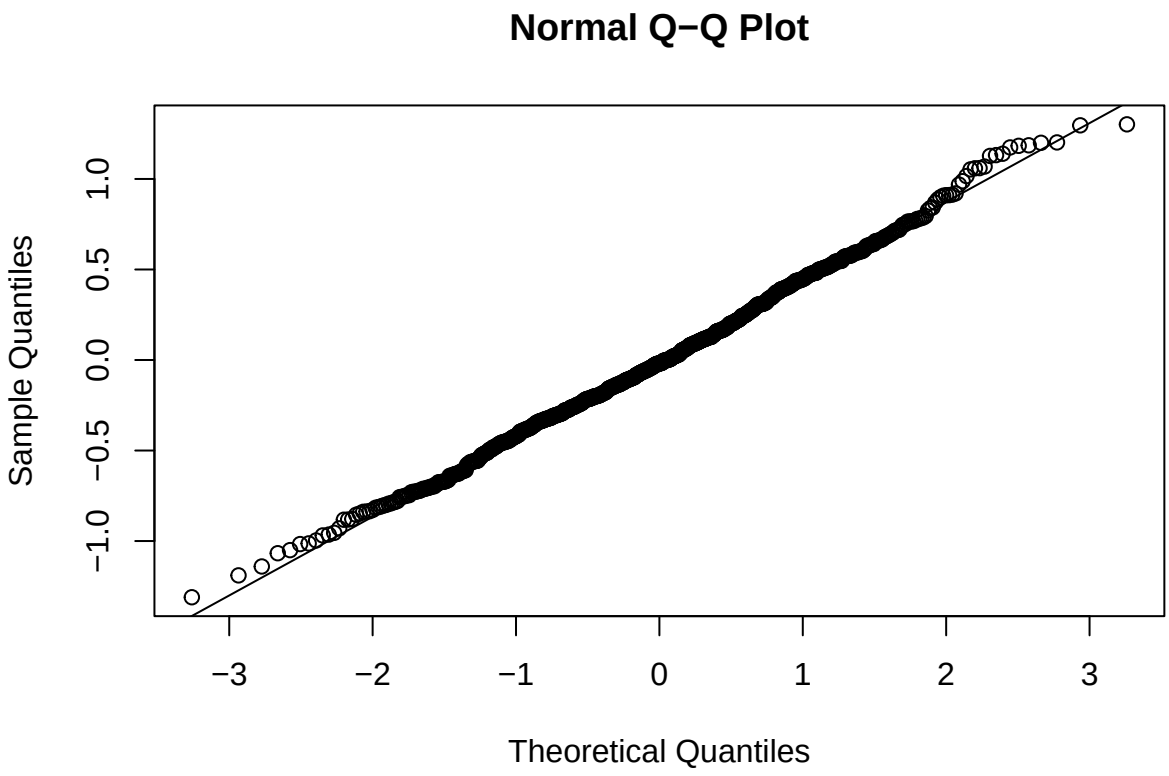
9.7)

```r
# Definir o número de iterações
N <- 1000

# Definir o coeficiente de correlação
rho <- 0.9

# Inicializar as cadeias
X <- numeric(N)
Y <- numeric(N)

# Definir os valores iniciais para X e Y
X[1] <- 0
Y[1] <- 0

# Executar o amostrador Gibbs
for (t in 2:N) {
  # Amostrar X[t] dado Y[t-1]
  X[t] <- rnorm(1, mean = rho * Y[t - 1], sd = sqrt(1 - rho^2))

  # Amostrar Y[t] dado X[t]
  Y[t] <- rnorm(1, mean = rho * X[t], sd = sqrt(1 - rho^2))
}
```

```r
# Definir o comprimento do burn-in
burn_in <- 100

# Descartar as amostras do burn-in
X_discarded <- X[(burn_in + 1):N]
Y_discarded <- Y[(burn_in + 1):N]

# Plotar a cadeia bivariada normal após o burn-in
plot(X_discarded, Y_discarded, type = "l", xlab = "X", ylab = "Y", main = "Cadeia Bivariada Normal (Apó
```

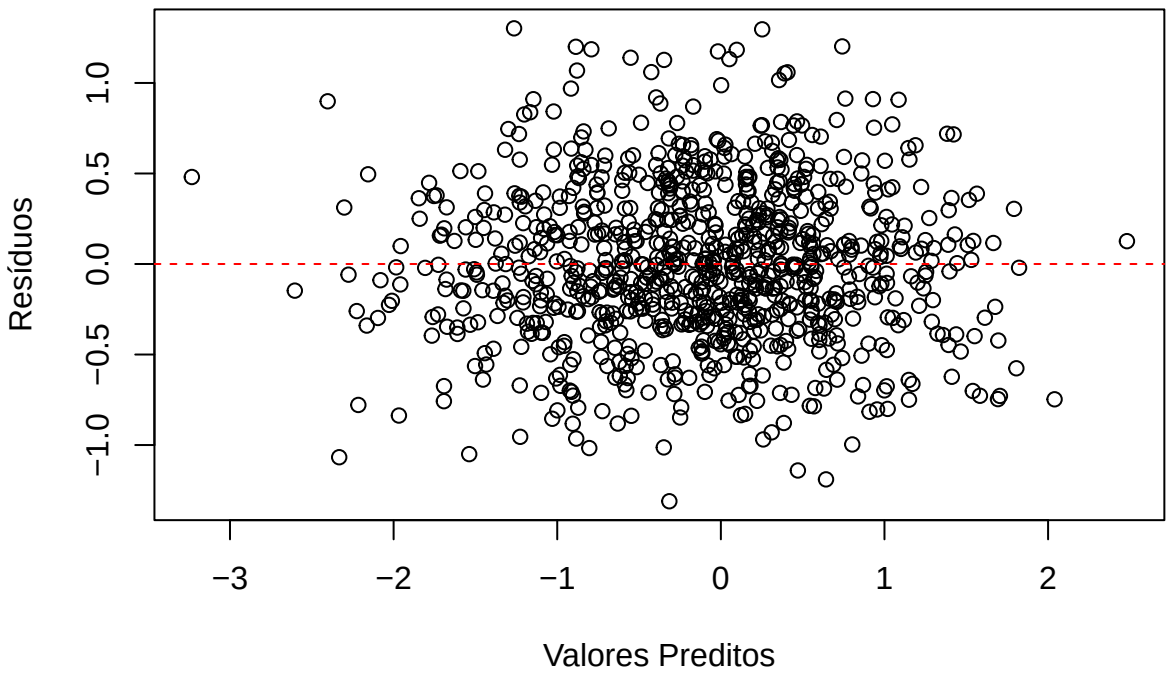


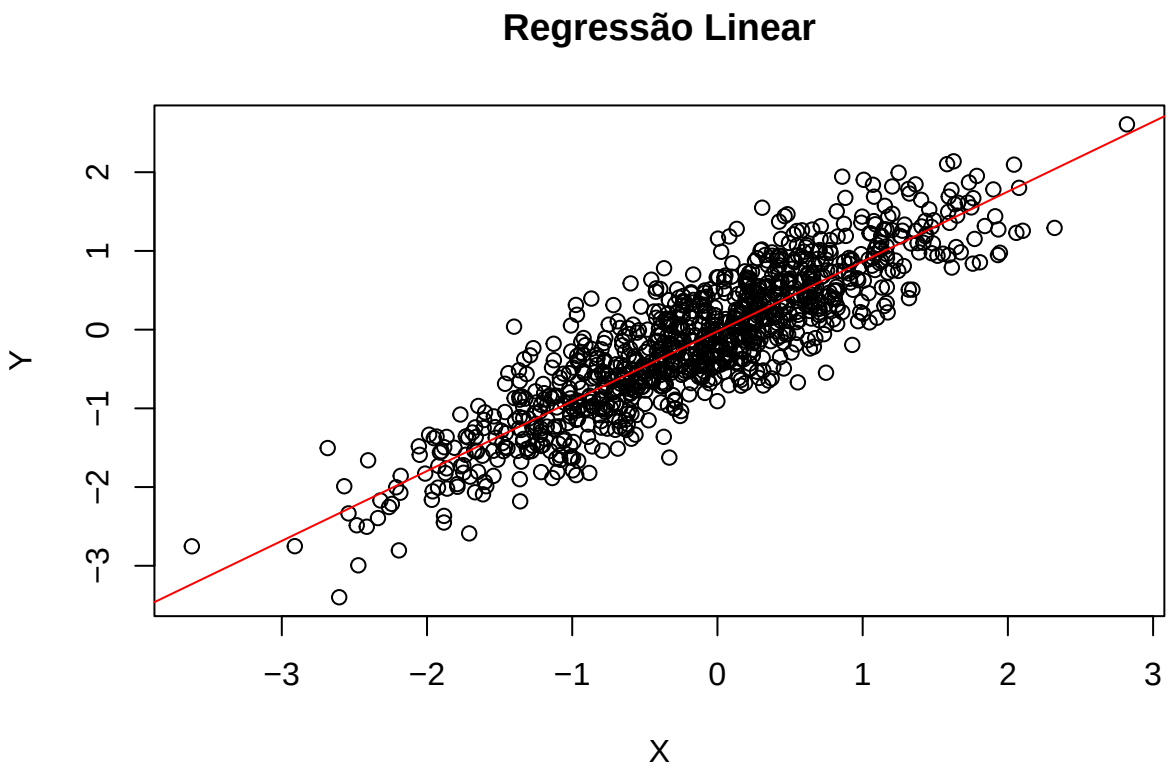

```r
# Aplicar a Regressão Linear

# Criar um data frame com as amostras descartadas
Banco <- data.frame(Y = Y_discarded, X = X_discarded)

# Passo 2: Criar o modelo linear
modelo <- lm(Y ~ X, data = Banco)

# Passo 3: Verificar a normalidade dos resíduos usando quantis
# Q-Q plot
qqnorm(modelo$residuals)
qqline(modelo$residuals)
```




```r
# Teste de Shapiro-Wilk
shapiro.test(modelo$residuals)  # normal
```

```
##
##  Shapiro-Wilk normality test
##
## data:  modelo$residuals
## W = 0.99729, p-value = 0.1392
```


Resíduos
1.0
0.5
0.0
−0.5
−1.0
−3
−2
−1
0
1
2
Valores Preditos


Regressão Linear
Y
X
−3
−2
−1
0
1
2
3