UNIVERSIDADE FEDERAL DE MINAS GERAIS - UFMG
INSTITUTO DE CIÊNCIAS EXATAS - ICEx
DEPARTAMENTO DE ESTATÍSTICA

Métodos Computacionais Aplicados à Estatística

Implementação no Software R

Cristiano de Carvalho Santos

Relatório técnico
Série Ensino
Dezembro de 2022

Prefácio

Este material foi elaborado para o uso como notas de aula do Professor Cristiano de Carvalho Santos na disciplina optativa de Métodos Computacionais Aplicados à Estatística (oferecida pelo Departamento de Estatística da UFMG ao Curso de Graduação em Estatística).

O objetivo principal do texto é apresentar através de exemplos como alguns métodos computacionais podem ser implementados no *software* R (https://cran.r-project.org/) para resolver problemas de inferência estatística, abordando desde a geração de variáveis aleatórias, o uso de métodos Monte Carlo, Bootstrap e Jackknife para estimação de viés e erro padrão, obtenção de intervalos de confiança e cálculo de p-valor em testes de hipóteses, além do uso dos algoritmos de Newton-Raphson, Escore de Fisher e EM para a obtenção de estimadores de máxima verossimilhança. Um pouco da teoria de tais métodos é apresentada brevemente ao longo da apostila, mas o objetivo não é aprofundar em aspectos teóricos e/ou demonstrar que estes métodos são válidos. Por favor, enviem sugestões e correções para ccsgaus@ufmg.br.

Sumário

Capítulo 1

Métodos para geração de variáveis aleatórias

Neste capítulo iremos abordar o conceito de número pseudo-aleatórios e métodos computacionais utilizados para a geração de variáveis aleatórias. O conteúdo apresentado é baseado nos Capítulos 3, 4 e 5 do livro do Ross (2013), no Capitulo 3 do livro de Rizzo (2008) e no Capítulo 6 do livro de Givens and Hoeting (2012).

1.1 Introdução

Ao pensarmos na geração de variáveis aleatórias, a primeira pergunta que pode surgir é: `Por que é necessário saber gerar variáveis aleatórias?`

Algumas das possíveis respostas para essa pergunta são:

- Gerar bancos de dados para avaliar o desempenho de modelos propostos;
- Aproximar integrais. Ex.: algoritmo EM Monte Carlo;
- Implementar métodos de inferência como bootstrap, testes de permutação, entre outros;
- Estatística Bayesiana: Gerar amostras da distribuição *a posteriori* quando a mesma não tem expressão fechada conhecida para sua função densidade ou função massa de probabilidade.

1.2 Geração de números pseudo-aleatórios

Não é possível gerar números aleatórios em um computador, e devido a isto trabalhamos com o conceito de números pseudo-aleatórios.

Definição 1.1. Os números *pseudo-aleatórios* constituem uma sequência de valores que, apesar de serem gerados de forma determinística, tem a aparência de serem variáveis aleatórias independentes com distribuição uniforme entre 0 e 1, isto é, esta sequência de valores atendem a propriedades esperadas por sequencias de variáveis aleatórias com distribuição uniforme entre 0 e 1.

O próximo passo é entender como os números pseudo-aleatórios podem ser gerados. Eles são obtidos por através de expressões matemáticas aplicadas recursivamente, e diferentes métodos podem ser utilizados para isso, abaixo apresentamos alguns deles.

Método Congruente Linear Multiplicativo

Este método para gerar números pseudo aleatórios inicia com valor inicial x_0, conhecido como **semente**, e então calcula recursivamente os valores x_n, para $n \geq 1$. O método consiste em seguir o seguinte algoritmo:

Algoritmo

Passo 1: Definir um valor inicial x_0 (**semente**);
Passo 2: Calcule recursivamente os sucessivos valores x_n, $n \geq 1$, por

$$x_n = ax_{n-1} \text{ mod } m,$$

em que a e m são números positivos inteiros. A operação acima significa que ax_{n-1} é dividido por m e o resto é atribuído a x_n;
Passo 3: A quantidade $y_n = \mathbf{x_n}/\mathbf{m}$ é uma aproximação para uma variável Uniforme(0, 1).

Exemplo 1.1. Se $x_0 = 11, a = 2, m = 16$, então

$$x_1 = 2(11) \text{ mod } 16 = 6 \quad \text{e} \quad y_1 = x_1/m = 6/16 = 0,375.$$

Note que, x_n assume um valor entre $0, 1, \ldots, m-1$, então, após um número finito de valores gerados, a sequência se repete.

As constantes a e m são escolhidas satisfazendo três critérios:

- Para uma semente inicial, a sequência resultante tem a "aparência" de ser uma sequência de variáveis aleatórias independentes com distribuição uniforme no intervalo (0, 1);
- O número de realizações da variável que pode ser gerado antes da repetição do valor inicial é muito grande;
- Os valores podem ser calculados eficientemente em um computador digital.

Observação:
A quantidade m deve ser escolhida como o maior número primo que pode ser guardado no computador, em um computador com 32 bits, $m = 2^{31} - 1$ e $a = 7^5$.

1.2.1 Geradores congruenciais mistos

Outro gerador de números pseudo-aleatórios usa recursões do tipo

$$x_n = (ax_{n-1} + c) \text{ mod } m.$$

Esses geradores envolvem um termo aditivo e um termo multiplicativo. Ao usar geradores deste tipo, muitas vezes escolhe-se m para igualar o comprimento da palavra do computador (*Neste contexto, palavra é a unidade que uma máquina utiliza para trabalhar com memória. Por exemplo, em uma máquina de 32 bits, a palavra tem 32 bits e em uma de 64 bits tem 64 bits.*), uma vez que isso torna o cálculo de $(ax_{n-1} + c)$ mod m - ou seja, a divisão de $ax_{n-1} + c$ por m - bastante eficiente.

O período de um gerador congruencial misto é no máximo m e dependo da escolha do fator a pode ser ainda menor que o modulo. Quando um gerador possui um período completo, todos os valores de 0 a $m-1$ serão gerados, após m números gerados os valores começam a se repetir gerando um ciclo. O gerador só possuirá um período completo para todas as sementes se e somente se:

- o módulo m e o incremento c forem relativamente primos,
- $a - 1$ for divisível por todos os fatores primos de m,
- $a - 1$ for divisível por 4 se m for divisível por 4.

Curiosidades:

- Os softares R (*default*) e Python utilizam um gerador conhecido como *Mersenne Twister*;
- No R, o gerador de número pseudo-aleatório e a semente podem ser alterados com a função `set.seed` (verifique o *help* desta função).

Não iremos explorar as questões teóricas que envolvem a construção de bons geradores de números pseudo-aleatórios. A partir daqui, assumiremos que já possuímos uma função que fornece um número com essas característica quando solicitada.

1.2.2 Exercícios

Exercício 1.1. Implemente algoritmo para método congruencial misto com $x_0 = 5,\ a = 5,\ c = 1$ e $m = 16$. Obtenha os 32 primeiros números da sequência $x_1, x_2, ...$ e observe se existe alguma repetição de série.

Exercício 1.2. (Exercício 1 do Capítulo 3 do Ross (2013)) Se $x_0 = 5$ e $x_n = 3x_{n-1}$ mod 150, encontre $x_1, ..., x_{10}$.

Exercício 1.3. (Adaptado do Exercício 14 do Capítulo 3 do Ross (2013)) Com $x_1 = 23$, $x_2 = 66$ e $x_n = 3x_{n-1} + 5x_{n-2}$ mod 100, $n \geq 3$, podemos calcular a sequência $u_n = x_n/100$, $n \geq 1$.

a. Imprima os 14 primeiros termos da sequência de u_n.
b. Faça um histograma dos 1000 primeiros termos da sequência de u_n.

1.3 Geração de variáveis aleatórias discretas

Nesta seção iremos apresentar alguns métodos para a geração de variáveis aleatórias seguindo distribuições discretas.

1.3.1 Método da transformação inversa: variáveis discretas

Suponha que queremos gerar um valor de uma variável aleatória X tendo função de massa de probabilidade

$$P(X = x_j) = p_j,\ \ j = 0, 1, ...,\ \text{e}\ \sum_j p_j = 1.$$

Para tal, geramos um valor u de uma variável aleatória $U \sim Uniforme(0, 1)$ e obtemos um valor x da variável aleatória X fazendo:

$$x = \begin{cases} x_0 & \text{se } u < p_0; \\ x_1 & \text{se } p_0 \leq u < p_0 + p_1; \\ \vdots & \\ x_j & \text{se } \sum_{l=0}^{j-1} p_l \leq u < \sum_{l=0}^{j} p_l; \\ \vdots & \end{cases}$$

Podemos fazer algumas observações sobre este método:

Observação 1: Para gerar um valor x da variável aleatória X, o procedimento pode ser descrito pelo seguinte algoritmo:

Algoritmo

Passo 1: Gere u de uma distribuição uniforme entre 0 e 1;
Passo 2: Se $u < p_0$ faça $x = x_0$ e pare, caso contrário vá para o próximo passo;
Passo 3: Se $u < p_0 + p_1$ faça $x = x_1$ e pare, caso contrário vá para o próximo passo;
Passo 4: Se $u < p_0 + p_1 + p_2$ faça $x = x_2$ e pare, caso contrário vá para o próximo passo;
$\vdots$
Passo k+2: $u < p_0 + p_1 + p_2 + \cdots + p_k$ faça $x = x_k$ e pare, caso contrário vá para o próximo passo.

Observação 2: Se x_j, $j \geq 0$, estão ordenados tal que $x_0 < x_1 < x_2 < \cdots$, então

$$X = x_j \text{ se } F(x_{j-1}) \leq u < F(x_j),$$

em que $F(x_j)$ é a função de distribuição de X no ponto x_j.

Observação 3: O tempo de algoritmo pode ser otimizado se os possíveis valores x_j de X estiverem em ordem decrescente do p_j.

Observação 4: Se a variável tiver distribuição Uniforme discreta não é necessário buscar a qual intervalo o número aleatório pertence através de condições. Suponha que $P(X = j) = 1/n$ para $j = 1, \ldots, n$. Neste caso,

$$X = x_j \text{ se } \frac{j-1}{n} \leq u < \frac{j}{n},$$

então podemos escrever $X = Int(n \times u) + 1$, em que $Int(x)$ é a parte inteira de x.

Exemplo 1.2. Suponha que queremos simular de uma variável aleatória X tal que

$$p_1 = 0,2, \;\; p_2 = 0,15, \;\; p_3 = 0,25 \;\text{ e }\; p_4 = 0,4,$$

em que $p_j = P(X = j)$.

Podemos gerar um valor u de $U \sim Uniforme(0,1)$ e fazer:

Alternativa 1

Algoritmo

Passo 1: Se $u < 0,20$, faça $x = 1$ e pare, caso contrário vá para o próximo passo;
Passo 2: Se $u < 0,35$, faça $x = 2$ e pare, caso contrário vá para o próximo passo;
Passo 3: Se $u < 0,60$, faça $x = 3$ e pare, caso contrário vá para o próximo passo;
Passo 4: caso contrário faça $x = 4$.

Abaixo é apresentado o código com a implementação da alternativa 1.

```
rcategorica1 <- function(n, x, p){
  am <- numeric(n);  acum <- cumsum(p)
  for(i in 1:n){
    u <- runif(1)
    for(j in 1:length(x)){
      if(u <= acum[j]){
        am[i] <- x[j]
        break
      }
    }
  }
  return(am)
}
```

Note no código acima que x representa o vetor de possíveis valores que a variável pode assumir, enquanto que p representa o respectivo vetor de probabilidades.

Alternativa 2

Algoritmo

Passo 1: Se $u < 0,40$, faça $x = 4$ e pare, caso contrário vá para o próximo passo;
Passo 2: Se $u < 0,65$, faça $x = 3$ e pare, caso contrário vá para o próximo passo;
Passo 3: Se $u < 0,85$, faça $x = 1$ e pare, caso contrário vá para o próximo passo;
Passo 4: caso contrário faça $x = 2$.

Abaixo é apresentado o código com a implementação da alternativa 2.

```
rcategorica2 <- function(n, x, p){
  ordem <- order(p, decreasing = T)
  x <- x[ordem];  p <- p[ordem]
  am <- numeric(n);  acum <- cumsum(p)
  for(i in 1:n){
    u <- runif(1)
    for(j in 1:length(x)){
      if(u <= acum[j]){
        am[i] <- x[j]
        break
      }
    }
  }
  return(am)
}
```

Abaixo avaliamos se as amostras obtidas são coerentes com o resultado esperado.

```
amostra1 <- rcategorica1(10000, 1:4, c(0.2, 0.15, 0.25, 0.4))
amostra2 <- rcategorica2(10000, 1:4, c(0.2, 0.15, 0.25, 0.4))

par( mfrow = c(1,2), mar = c(2, 4, 1, 1) )
barplot(prop.table(table(amostra1)), ylab = "Frequência", xlab = "x",
        main = "Alternativa 1", col = 2:5, ylim = c(0, 0.45))
barplot(prop.table(table(amostra2)), ylab = "Frequência", xlab = "x",
        main = "Alternativa 2", col = 2:5, ylim = c(0, 0.45))
```

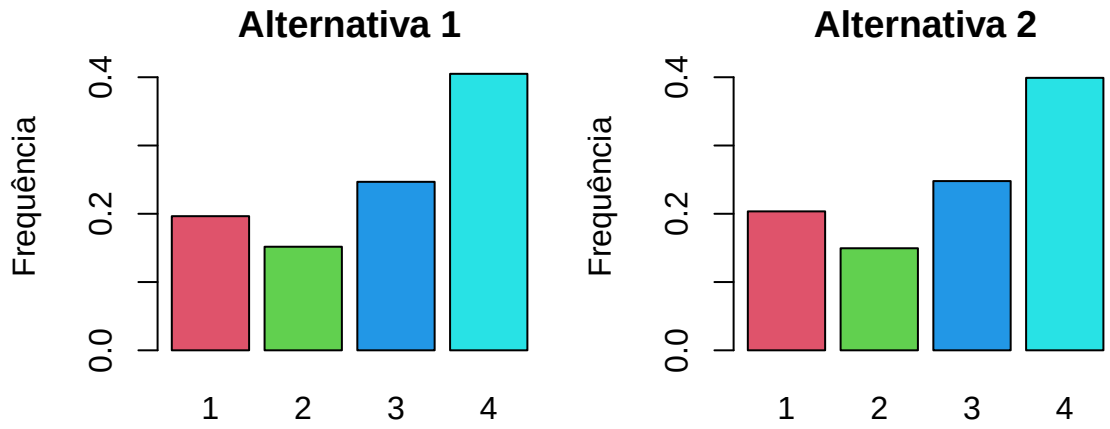

Figura 1.1: Ilustração de simulação com as duas funções construídas

Com o pacote `microbenchmark`, podemos comparar qual das funções é mais rápida e também comparar com a função sample do R.

```
require(microbenchmark)
options(digits = 1)
n <- 1000
pr <- c(0.005, 0.005, 0.01, 0.02, 0.4, 0.56)
x <- 1:length(pr)
resumo <- microbenchmark(
  rcategorica1(n, x, pr), rcategorica2(n, x, pr),
  sample(x, n, replace = T, pr),
  times = 1000, unit = "ms")
print(resumo)
```

```
## Unit: milliseconds
##                           expr  min   lq mean median   uq max neval cld
##         rcategorica1(n, x, pr) 1.55 1.63 2.06   1.74 1.89  41  1000   c
##         rcategorica2(n, x, pr) 1.43 1.52 1.93   1.64 1.82  10  1000  b
##  sample(x, n, replace = T, pr) 0.02 0.02 0.03   0.02 0.03   4  1000 a
```

```
options(digits = 7)
```

Exemplo 1.3. (Permutação) Suponha que queremos gerar uma permutação dos números $1, 2, \ldots, n$ em que todas as $n!$ possíveis ordenações são igualmente prováveis.

Uma alternativa é escolher um número aleatoriamente e colocar na posição n, escolher outro

número entre os $n-1$ números restantes e colocar na posição $n-1$, assim por diante. É conveniente e eficiente colocar os números em um lista ordenada e escolher aleatoriamente a posição do número ao invés do próprio número. O algoritmo é dado por:

Algoritmo

Passo 1: Seja $P_1, P_2, \ldots, P_n$ uma permutação de $1, 2, \ldots, n$;
Passo 2: Faça $k = n$;
Passo 3: Gere um número aleatório I de uma uniforme discreta em $1, \ldots, k$;
Passo 4: Troque os valores de P_I e P_k;
Passo 5: Faça $k = k - 1$ e se $k > 1$ vá para o Passo 3.
Passo 6: A nova ordenação $P_1, P_2, \ldots, P_n$ é a ordenação desejada.

Observações:

- Este algoritmo pode ser utilizado para gerar um subconjunto de tamanho r do conjunto original.
- Para ter maior eficiência podemos sempre considerar $r \leq n/2$, pois se isso não ocorrer podemos escolher os elementos que não pertencem ao subconjunto.

Exemplo 1.4. (Geométrica) A variável aleatória X é dito ter distribuição Geométrica com parâmetro p se

$$P(X = i) = pq^{i-1},\ i \geq 1, \text{ em que } q = 1 - p.$$

Note que

$$\begin{aligned} P(X \leq j-1) = \sum_{i=1}^{j-1} P(X = i) &= 1 - P(X > j-1) \\ &= 1 - q^{j-1},\ j \geq 1. \end{aligned}$$

Geramos X por gerar $U \sim Uniforme(0,1)$ e fazer X igual ao j que

$$1 - q^{j-1} \leq U < 1 - q^j.$$

Equivalentemente,

$$q^j < 1 - U \leq q^{j-1}.$$

Ou ainda,

$$\begin{aligned} X &= min\{j : q^j < 1-U\} \\ &= min\left\{j : j > \frac{log(1-U)}{log(q)}\right\} \\ &= Int\left(\frac{log(1-U)}{log(q)}\right) + 1 = Int\left(\frac{log(U)}{log(q)}\right) + 1. \end{aligned}$$

Obs: O termo $min\{j : q^j < 1-U\}$ é lido como o menor j tal que a condição $q^j < 1-U$ é satisfeita.

1.3.2 Amostrando da distribuição Poisson

Uma variável aleatória X tem distribuição Poisson com parâmetro λ se

$$P(X = x) = \frac{e^{-\lambda}\lambda^x}{x!}, \text{ para } x = 0, 1, ...$$

Para implementar o método da transformada inversa podemos fazer uso da seguinte relação recursiva para calcular as probabilidades a medida que são necessárias:

$$P(X = x+1) = P(X = x) \times \frac{\lambda}{x+1}, \ x \geq 0. \tag{1.1}$$

Com o uso da fórmula recursiva acima, podemos gerar uma amostra da distribuição Poisson seguindo o algoritmo abaixo e otimizando os cálculos computacionais.

Algoritmo

Passo 1: Gere um número aleatório u da variável $U \sim Uniforme(0, 1)$;
Passo 2: Faça $i = 0, pr = e^{-\lambda}$ e $Fx = pr$;
Passo 3: Se $u < Fx$, faça $x = i$ e pare o algoritmo;
Passo 4: Atualize $pr = pr\frac{\lambda}{i+1}$, $Fx = Fx + pr$ e $i = i + 1$;
Passo 5: Vá para o passo 3.

Note que inicialmente o algoritmo compara se $u < e^{-\lambda} = P(X = 0)$ e, caso contrário, segue com a comparação se $u < P(X = 0) + P(X = 1)$. Em cada iteração, $Fx =$

$P(X \leq i)$ e $pr = P(X = i)$. Note que $P(U < e^{-\lambda}) = e^{-\lambda} = P(X = 0)$, visto que U tem distribuição Uniforme entre 0 e 1. Da mesma forma, teremos que $X = 1$ se $e^{-\lambda} \leq u < e^{-\lambda} + e^{-\lambda}\lambda$, isto é,

$$P(X = 1) = P(e^{-\lambda} \leq U < e^{-\lambda} + e^{-\lambda}\lambda) = e^{-\lambda}\lambda.$$

O mesmo procedimento segue sendo feito para dos demais valores possíveis para X.

A seguir implementamos uma função para gerar uma amostra de tamanho n da distribuição Poisson seguindo o algoritmo acima.

```
require(tidyverse); require(kableExtra); require(knitr)

## Implementação do algoritmo para gerar um valor da Poisson
rpois_aux <- function(lambda){
  u <- runif(1, 0, 1)
  i <- 0;  pr <- exp(-lambda);  Fx = pr
  while(u >= Fx){
    pr <- pr*lambda/(i+1)
    Fx <- Fx + pr
    i <- i+1
  }
  return(i)
}

## Replicando para n valores
rpoisson <- function(n, lambda){
  replicate(n, expr = rpois_aux(lambda), simplify = TRUE)
}

## Gerando uma amostra
lambda <- 5
amostra <- rpoisson(10000, lambda)

## Frequencias relativas na amostra gerada e prob teorica
tab <- rbind( round( prop.table( table(amostra) ), 3),
              round(dpois(0:max(amostra),lambda),3))
colnames(tab) <- 0:max(amostra)
rownames(tab) <- c("freq", "prob")
```

```
## imprimindo a tabela formatada com as 10 primeras colunas
tab <- tab[,1:10] %>%
  kbl(align = c(rep("c", 5)),
      caption = "Comparação de frequências relativas com probabilidades teóricas")

tab %>%
  kable_styling(latex_options = "hold_position",
                bootstrap_options = c("bordered"), font_size = 11)
```

Tabela 1.1: Comparação de frequências relativas com probabilidades teóricas

	0	1	2	3	4	5	6	7	8	9
freq	0.007	0.031	0.086	0.138	0.184	0.175	0.141	0.102	0.066	0.038
prob	0.007	0.034	0.084	0.140	0.175	0.175	0.146	0.104	0.065	0.036

```
## Graficamente
freq_rel <- prop.table(table(amostra))
par(mar = c(4, 4, 1, 1))
plot(freq_rel, main = "", xlab = "x", ylab = "Frequência relativa (prob)")
points(0:max(amostra), dpois(0:max(amostra), lambda), col = 2, cex = 1.5)
```

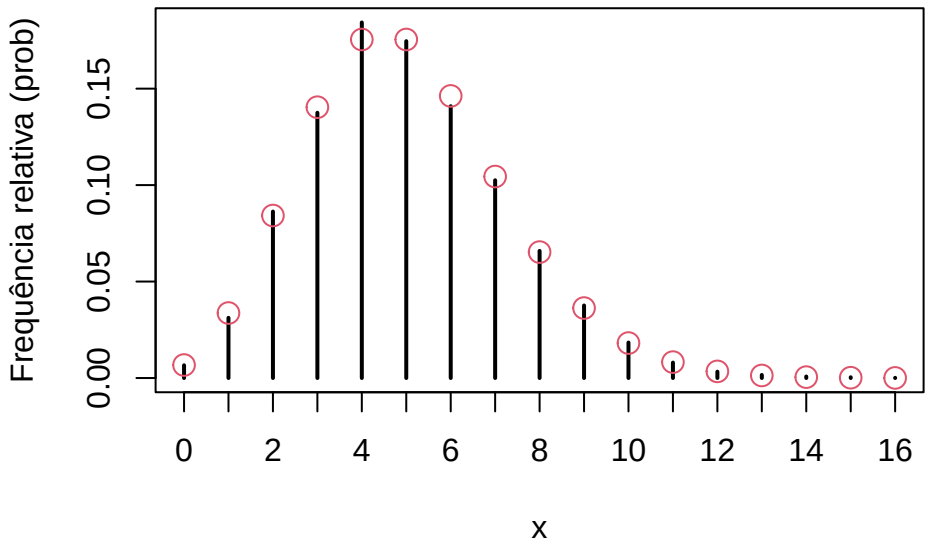

Figura 1.2: Gráfico de frequências de uma amostra gerada da distribuição Poisson e probabilidades teóricas (pontos)

Note que, em média, o algoritmo precisará fazer $\lambda+1$ comparações para amostrar um valor da distribuição poisson, pois ao gerar $x = 3$, por exemplo, o algoritmo verifica se o valor

gerado foi 0, 1, 2 e 3 antes de finalizar. Com isso, o algoritmo pode se tornar computacionalmente ineficiente quando λ é grande. Visto que em uma distribuição Poisson, os dois valores mais prováveis são dois inteiros próximos de λ, o algoritmo pode ser melhorado ao iniciar destes pontos ao invés de 0.

Por exemplo, seja $I = Int(\lambda)$ e use a equação (1.1) para determinar recursivamente $F(I)$, em que $Int(x)$ representa a parte inteira de x. Para gerar, observa-se se $X \leq I$ ou não, vendo se $U \leq F(I)$ ou não. Direciona-se a busca para valores menores a partir de I se $X \leq I$ e para valores maiores a partir de $I + 1$ caso contrário

O número de pesquisas necessárias para este algoritmo é aproximadamente 1 mais a diferença absoluta entre a variável aleatória X e sua média λ. Para um valor grande de λ, utilizando a aproximação da Poisson pela Normal, o número esperado de buscas é aproximadamente $1 + 0.789\sqrt{\lambda}$. Veja Ross para mais detalhes da conta!

1.3.3 Amostrando da distribuição Binomial

Suponha que queremos gerar uma amostra de uma variável aleatória X seguindo distribuição Binomial parâmetros m e p, isto é, tal que

$$P(X = x) = \frac{m!}{x!(m-x)!} p^x (1-p)^{m-x}, \text{ para } x = 0, 1, ..., m.$$

Para amostrar da distribuição Binomial podemos utilizar diferentes estratégias. Uma delas é, para cada observação da Binomial, gerar uma sequência de m variáveis aleatórias Bernoulli independentes com probabilidade de sucesso p e calcular o número de sucessos. Outra estratégia é utilizando o método de transformação inversa diretamente com a função de probabilidade acumulada da binomial. Para isso, note que

$$P(X = x + 1) = \frac{m-x}{x+1} \frac{p}{1-p} \times P(X = x), \ 0 \leq x \leq m - 1. \tag{1.2}$$

Com o uso da fórmula recursiva acima, podemos gerar uma amostra da distribuição Binomial seguindo o algoritmo abaixo.

Algoritmo

Passo 1: Gere um número aleatório u da variável $U \sim Uniforme(0, 1)$;
Passo 2: Faça $c = p/(1-p)$, $i = 0$, $pr = (1-p)^m$ e $Fx = pr$;
Passo 3: Se $u < Fx$, faça $x = i$ e pare o algoritmo;
Passo 4: Atualize $pr = pr\frac{c(m-i)}{i+1}$, $Fx = Fx + pr$ e $i = i + 1$;
Passo 5: Vá para o passo 3.

Em cada iteração do algoritmo, $Fx = P(X \leq i)$ e $pr = P(X = i)$.

```
## Implementação do algoritmo para gerar um valor
rbinom_aux1 <- function(m, p){
  u <- runif(1, 0, 1)
  c <- p/(1-p);  i <- 0;  pr <- (1-p)^m;  Fx = pr
  while(u >= Fx){
    pr <- pr*c*(m-i)/(i+1)
    Fx <- Fx + pr
    i <- i+1
  }
  return(i)
}

## Replicando para n valores - algoritmo
rbinomial <- function(n, m, p){
  replicate(n, expr = rbinom_aux1(m, p), simplify = TRUE)
}

## Gerando um valor da binomial a partir de bernoullis
rbinom_aux2 <- function(m, p){
  u <- runif(m, 0, 1)
  x <- sum( u <= p )
  return(x)
}

## Replicando para n valores - a partir de bernoullis
rbinomial2 <- function(n, m, p){
  replicate(n, expr = rbinom_aux2(m, p), simplify = TRUE)
}

## Gerando uma amostra
m <- 20;  p <- 0.3

amostra <- rbinomial(10000, m, p)
amostra2 <- rbinomial2(10000, m, p)
```

```
matriz <- cbind( c(p*m, p*(1-p)*m),
                 c(mean(amostra),  var(amostra)),
                 c(mean(amostra2),  var(amostra2)) )

rownames(matriz) <- c("Média", "Variância")
colnames(matriz) <- c("Teórica", "Amostra1", "Amostra2")

matriz
```

```
##           Teórica Amostra1 Amostra2
## Média         6.0 5.998500 6.022700
## Variância     4.2 4.208919 4.190804
```

```
## Prob teorica
probs <- round(dbinom(0:max(amostra), m, p), 3)
probs2 <- round(dbinom(0:max(amostra2), m, p), 3)

## Graficamente
par( mfrow = c(1,2), mar = c(4, 4, 1, 1) )
plot(prop.table(table(amostra)), main = "Algoritmo 1", xlab = "x",
     ylab = "Frequência relativa (prob)")
points(0:max(amostra), probs, col = 2, cex = 1.5)
plot(prop.table(table(amostra2)), main = "Algoritmo 2", xlab = "x",
     ylab = "Frequência relativa (prob)")
points(0:max(amostra2), probs2, col = 2, cex = 1.5)
```

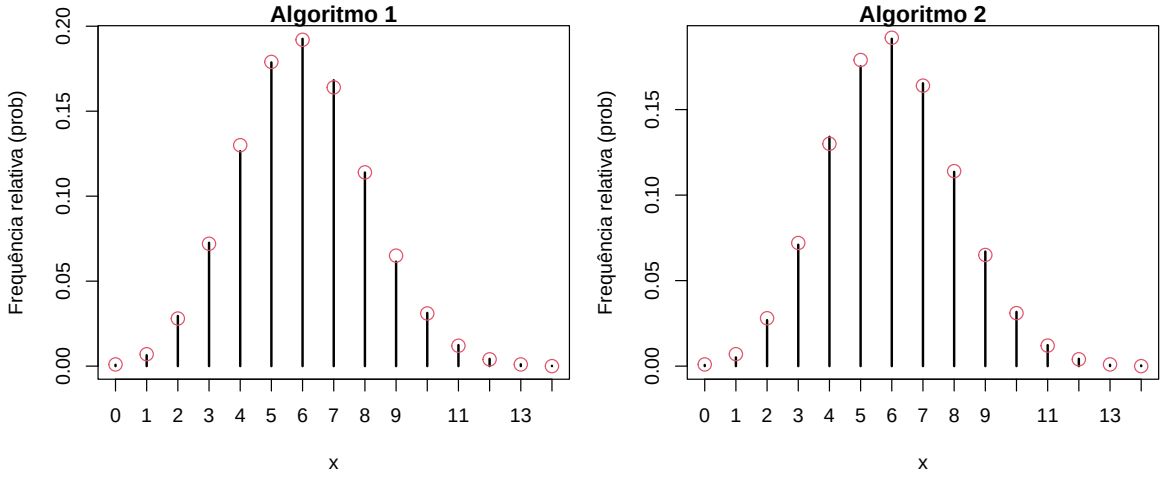

Figura 1.3: Gráfico de frequências des amostras geradas da distribuição Binomial com os dois algoritmos e probabilidades teóricas (pontos)

O algoritmo pode ser melhorado seguindo duas diferentes estratégias:

Estratégia 1: No algoritmo anterior, o número de pesquisas realizadas para gerar um número é 1 a mais que o valor de x gerado. Portanto, em média, serão necessárias 1 + mp pesquisas para gerar X. Quando $p > 0,5$, podemos gerar X subtraindo m do valor de uma variável aleatória $Y \sim Binomial(m, 1-p)$.

Estratégia 2: Da mesma forma que ocorre com a Poisson, quando mp resulta em um valor grande, podemos iniciar a buscar perto dos valores mais prováveis, isto é, próximos da média da distribuição. Primeiro determinamos se o valor gerado a partir do u será menor ou igual a $I = Int(mp)$ ou se será maior que I. No primeiro caso, deve-se então começar a pesquisa com I, então $I-1, ...$, e assim por diante; enquanto no último caso, deve-se começar a pesquisar com $I+1$ e ir aumentando.

1.3.4 Método da aceitação e rejeição

Suponha que queremos amostrar valores da variável aleatória X com função massa de probabilidade dada por

$$p_j = P(X = x_j),\ j \geq 1.$$

O método da rejeição poderá ser útil em casos que não se tem expressão para a função de probabilidade acumulada F e em casos em que mesmo que se conheça $F(x)$, não possível chegar à uma expressão para a função F^{-1}. Isto ocorre em situações em que só conhecemos o núcleo da função massa de probabilidade, mas não conhecemos a constante normalizadora. Por exemplo, em algumas situações pode ser necessário gerar da distribuição condicional de uma variável discreta que pode assumir infinitos valores (Quando o suporte tem valores finitos é mais simples de obter a constante normalizadora).

A ideia deste método é amostrar de uma variável aleatória W com função massa de probabilidade $q_j = P(W = x_j)$, para $j \geq 1$, que seja simples de amostrar e decidir se o valor gerado deve ser aceito ou não para a amostra de X. Deve-se aceitar o valor simulado com probabilidade proporcional a p_j/q_j.

Seja c uma constante tal que $\frac{p_j}{q_j} \leq c,$ para todo j tal que $p_j > 0$, o método da aceitação e rejeição gera valores da variável X através dos algoritmo a seguir.

Algoritmo

Passo 1: Gere um valor de w da v.a W com função massa de probabilidade $q_j = P(W = w_j)$, para $j = 1, 2, ...$;
Passo 2: Gere um número aleatório u da variável $U \sim Uniforme(0, 1)$;
Passo 3: Se $u < p_j/(cq_j)$, faça $x = w$ e pare o algoritmo. Caso contrário, retorne ao Passo 1.

O seguinte resultado mostra que o algoritmo funciona.

Teorema 1.1. *O algoritmo aceitação e rejeição gera uma variável aleatória X tal que $P(X = x_j) = p_j$, $j = 1, 2, ...$ Além disso, o número de iterações do algoritmo necessárias para obter um valor x é uma variável aleatória geométrica com média c.*

Demonstração. Incialmente, calculamos

$$\begin{aligned} P(W = x_j, \text{o valor ser aceito}) &= P(W = x_j)P(\text{Ser aceito} \mid W = x_j) \\ &= q_j \times \frac{p_j}{cq_j} \\ &= \frac{p_j}{c}. \end{aligned}$$

Somando para todo j resulta que

$$P(\text{o valor ser aceito}) = \sum_j \frac{p_j}{c} = c^{-1}.$$

Visto que cada iteração resulta de forma independente em uma probabilidade c^{-1} de um valor ser aceito, o número de iterações necessárias para gerar um valor tem distribuição geométrica com média c.

Além disso,

$$\begin{aligned} P(X = x_j) &= \sum_{i=1}^{\infty} P(x_j \text{ ser aceito na iteração } i) \\ &= \sum_{i=1}^{\infty} \left(1 - \frac{1}{c}\right)^{i-1} p_j/c \\ &= p_j. \end{aligned}$$

O último resultado é devido a $\sum_{i=1}^{\infty}(1-1/c)^i = c-1$. □

Exemplo 1.5. Suponha que queremos simular de uma variável aleatória X que assume os valores $1, 2, \ldots, 10$ com respectivas probabilidades ($p_j = P(X = j)$) iguais a

$$0.11, 0.12, 0.09, 0.08, 0.12, 0.10, 0.09, 0.09, 0.10, 0.10.$$

Para isto podemos utilizar o método da transformação inversa, no entanto iremos utilizar o algoritmo de aceitação e rejeição como ilustração de tal método. Consideramos a distribuição uniforme discreta em $(1, \ldots, 10)$ para gerar os valores propostos, isto é, $q_j = 1/10,\ j = 1, \ldots, 10$.

Neste caso, escolhemos c talque $c = \max\{p_j/q_j\} = 1,2$ e o algoritmo é dado por:

Algoritmo

Passo 1: Gere um valor de u_1 da distribuição $Uniforme(0,1)$ e faça $y = Int\{10u_1\} + 1$;
Passo 2: Gere um valor de u_2 da distribuição $Uniforme(0,1)$;
Passo 3: Se $u_2 < p_y/0.12$, faça $x = y$ e pare o algoritmo. Caso contrário, retorne ao Passo 1.

Em média, o algoritmo requer apenas 1,2 iterações para gerar um valor de X.

No R temos:

```
rreject_categorica <- function(n){
  x <- numeric(n)
  py <- c(0.11, 0.12, 0.09, 0.08, 0.12, 0.10, 0.09, 0.09, 0.10, 0.10)
  cont <- 0
  while(cont < n){
    u1 <- runif(1); y <- as.integer(10*u1)+1
    u2 <- runif(1)
    if(u2< py[y]/0.12){
      cont <- cont + 1; x[cont] <- y
    }
```

```
  }
  return(x)
}

par(mar = c(4, 4, 1, 1))
amostra <-rreject_categorica(100000)
barplot(prop.table(table(amostra)), ylab = "Freq relativa", xlab = "x")
```

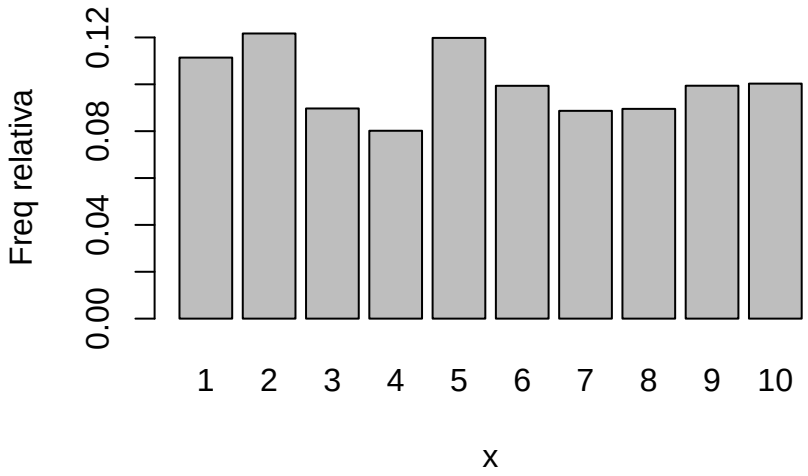

Figura 1.4: Gráfico de barras para ilustração do algoritmo de rejeição no caso discreto

1.3.5 Exercícios

Exercício 1.4. Implemente um algoritmo para fazer permutação utilizando o algoritmo para gerar uma amostra de uma variável categórica. Não utilize a função sample do R.

Exercício 1.5. Implemente o algoritmo para gerar da distribuição geométrica.

Exercício 1.6. Implemente a versão otimizada do algoritmo para gerar da distribuição Poisson e faça comparações do tempo computacional com o algoritmo inicial. Utilize o pacote microbenchmark para comparar.

Exercício 1.7. Reescreva a função do Exemplo 2.6 permitindo que os possíveis valores de X e as probabilidades sejam passadas como argumentos. Faça um estudo de simulação demonstrando a efetividade da função implementada para vários cenários de distribuições para variáveis categóricas.

1.4 Geração de variáveis aleatórias contínuas

Nesta seção iremos apresentar alguns métodos para a geração de variáveis aleatórias seguindo distribuições discretas.

1.4.1 Método da transformação inversa para variáveis contínuas

O método da tranformação inversa no caso de variáveis contínuas é baseado na seguinte proposição.

Proposição 1.1. *Seja U uma variável aleatória Uniforme(0, 1). Para qualquer função de distribuição contínua F a variável aleatória X definida por*

$$X = F^{-1}(U)$$

tem distribuição F.

Demonstração. Seja F_X denotando do função de distribuição da variável definida por $X = F^{-1}(U)$. Então,

$$\begin{aligned} F_X(x) &= P(X \leq x) \\ &= P(F^{-1}(U) \leq x). \end{aligned}$$

Visto que F é uma função de distribuição, segue que $F(x)$ é uma função **monótona crescente** de x e a desigualdade $a \leq b$ implica em $F(a) \leq F(b)$. Assim,

$$\begin{aligned} F_X(x) &= P(\ F(F^{-1}(U)) \leq F(x)\ ) \\ &= P(U \leq F(x)) \\ &= F(x). \end{aligned}$$

□

Então, o procedimento para a geração da variável aleatória é dado por:

Algoritmo

Passo 1: Gerar um valor u da distribuição uniforme(0, 1);
Passo 2: Obter o valor x através da transformação $F^{-1}(u)$;

Na gráfico abaixo ilustramos a ideia do método da transformação inversa no caso contínuo. Gerando um valor para a probabilidade acumulada, podemos encontrar qual o valor de x em que a função de distribuição acumulada será igual ao valor gerado.

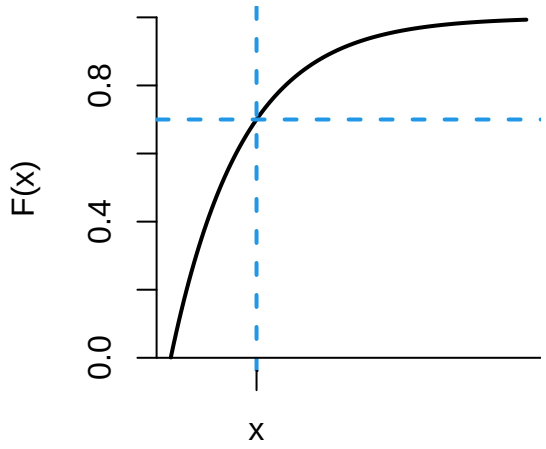

Figura 1.5: Ilustração do método da transformada inversa

Exemplo 1.6. Como amostrar de uma distribuição Normal truncada no intervalo (a, b) com parâmetros μ e σ^2 ?

Esta distribuição possui função densidade dada por:

$$f(x \mid \mu, \sigma^2, a, b) = \frac{\phi(x \mid \mu, \sigma^2)}{\Phi(b \mid \mu, \sigma^2) - \Phi(a \mid \mu, \sigma^2)},$$

em que $\phi(x \mid \mu, \sigma^2)$ e $\Phi(x \mid \mu, \sigma^2)$ são as funções densidade de probabilidade e função de probabilidade acumulada da distribuição normal com parâmetro Normal com parâmetros μ e σ^2 avaliadas no ponto x.

Esta função pode ser implentada com o código:

```
dnormtrunc <- function(x, mu, sigma, a, b){
  d <- dnorm(x, mu, sigma)/( pnorm(b, mu, sigma) - pnorm(a, mu, sigma) )
}
```

Neste caso, a função de distribuição inversa $F^{-1}(u)$ é dada por

$$F^{-1}(u) = \Phi^{-1}\left(\Phi(a \mid \mu, \sigma^2) + [\, \Phi(b \mid \mu, \sigma^2) - \Phi(a \mid \mu, \sigma^2) \,] \times u \;\mid \mu, \sigma^2 \right).$$

Com isto, podemos implementar uma função para gerar uma amostra da distribuição normal truncada com o código a seguir.

```
rnormtrunc <- function(n, mu, sigma, a, b){
  us <- runif(n)
  amostra <- qnorm( pnorm(a, mu, sigma) +
                      ( pnorm(b, mu, sigma) - pnorm(a, mu, sigma) )*us, mu, sigma )
}
```

Abaixo, ilustramos o uso da função construída.

```
a <- 6
b <- 13
amostra <- rnormtrunc(10000, 10, 2, a, b)

par(mar = c(4, 4, 1, 1))
hist(amostra, xlab = expression(x), ylab = "Densidade", prob = T, main = "")
grid <- seq(a, b, by = 0.01)
lines(grid, dnormtrunc(grid, 10, 2, a, b), col = "red", lwd = 2 )
```

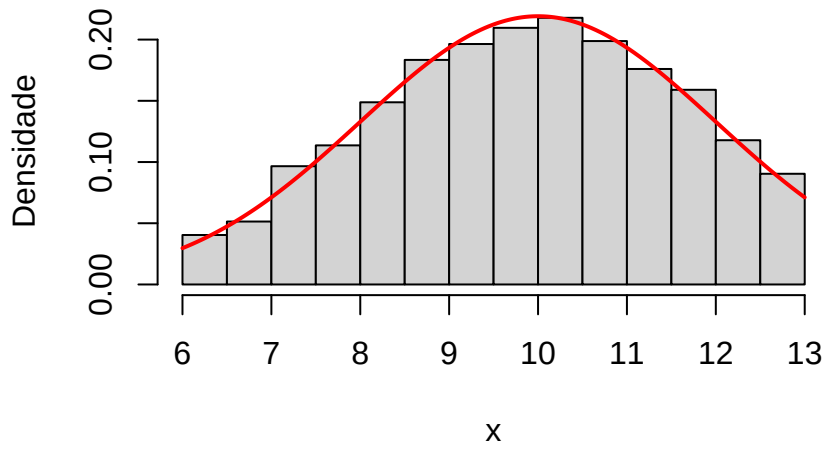

Figura 1.6: Histograma de uma amostra gerada da distribuição Normal truncada entre 6 e 10

1.4.2 Método da transformação

Se uma variável aleatória Y é obtida partir de uma transformação g de outra variável aleatória X, então podemos amostrar de X e obter uma amostra de Y apenas aplicando a transformação $Y = g(X)$.

Exemplo 1.7. Suponha que desejamos gerar uma amostra aleatória de tamanho n da distribuição Qui-quadrado com 1 grau de liberdade e dispomos de uma função que gera da distribuição normal padrão. Além disso, temos que se $Z \sim N(0, 1)$ então a variável aleatória $X = Z^2 \sim \chi^2_1$.

Podemos implementar no R com o seguinte código.

```
n <- 1000
z <- rnorm(n); x <- z^2
par(mar = c(4, 4, 1, 1))
hist(x, xlab = expression(x), ylab = "Densidade", prob = T, main = "", breaks = 50)
grid <- seq(min(x), max(x), by = 0.01)
lines(grid, dchisq(grid, 1), col = "red", lwd = 2 )
```

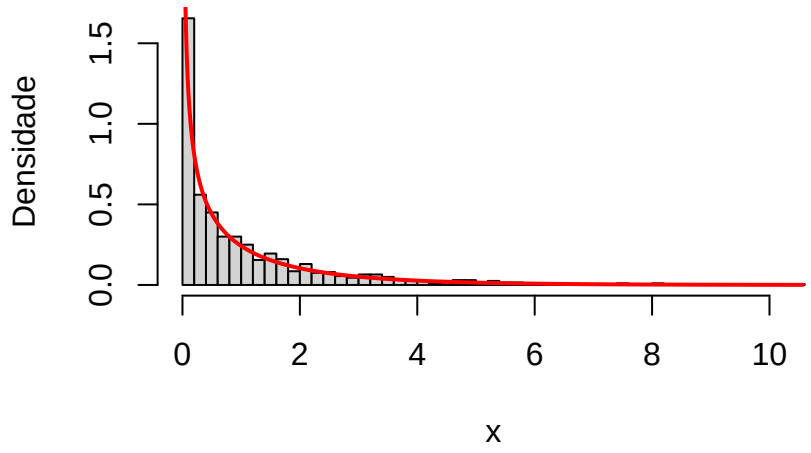

Figura 1.7: Histograma da amostra gerada da distribuição Qui-quadrado com 1 grau de liberdade

Exemplo 1.8. Suponha que desejamos gerar uma amostra aleatória de tamanho n da distribuição Normal assimétrica, com parâmetros μ, σ^2 e λ, cuja densidade é dada por

$$f(x|\mu,\sigma^2,\lambda) = \frac{2}{\sigma}\phi\left(\frac{x-\mu}{\sigma}\right)\Phi\left(\lambda\left(\frac{x-\mu}{\sigma}\right)\right), \quad -\infty < x < \infty.$$

O cálculo da função densidade pode ser feito com o seguinte código.

```
dSN <- function(x, mu, sigma, lambda){
  dens <- (2/sigma)*dnorm((x-mu)/sigma)*pnorm(lambda*(x-mu)/sigma)
  return(dens)
}
```

A seguir ilustramos a função densidade da distribuição Normal assimétrica padrão com três valores diferentes do parâmetro de assimetria.

```
grid <- seq(-2, 4, by =0.01)
d1 <- dSN(grid, mu = 0, sigma = 1, lambda = 1)
d2 <- dSN(grid, mu = 0, sigma = 1, lambda = 3)
d3 <- dSN(grid, mu = 0, sigma = 1, lambda = 10)
```

```
par(mar = c(4, 4, 1, 1))
plot(grid, d1, type = "l", ylim = c(0, 0.8), ylab = "Densidade", xlab = "x")
lines(grid, d2, lty= 2, lwd = 2, col = 2)
lines(grid, d3, lty= 4, lwd = 2, col = 4)
legend("topright", expression(lambda == 1, lambda == 3,lambda == 10),
       lwd = 2,lty = c(1,2,4), col = c(1,2,4), bty = "n")
```

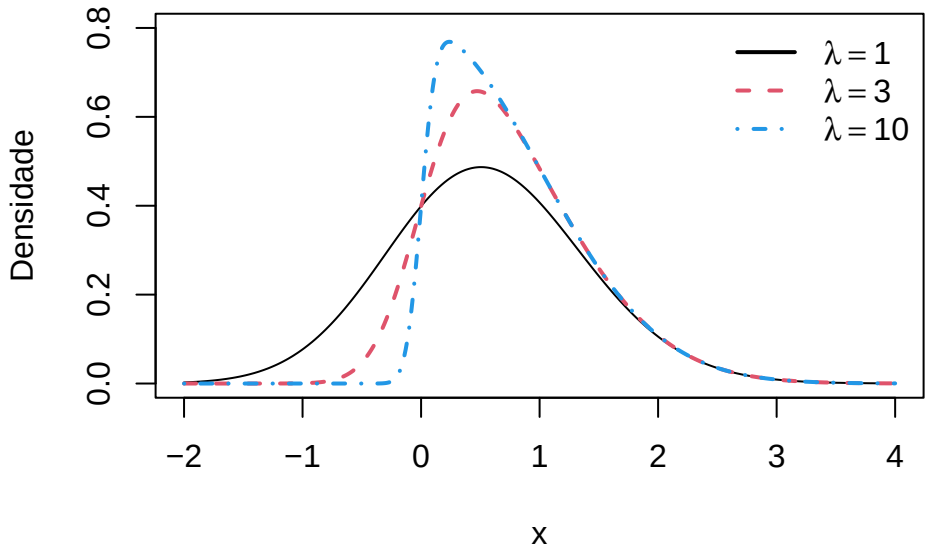

Figura 1.8: Função densidade da distribuição Normal Assimétrica padrão para diferentes valores do parâmetro de assimetria

A representação estocástica de Henze para a distribuição SN estabelece que, se $X \sim SN(\mu, \sigma^2, \lambda)$ então

$$X \overset{d}{=} \mu + \sigma \left[\delta|V| + (1-\delta^2)^{1/2} W\right],$$

em que $\delta = \lambda(1+\lambda^2)^{-1/2}$, V e W são variáveis aleatórias i.i.d com distribuição Normal padrão. Podemos amostrar da distribuição Normal assimétrica a partir do algoritmo a seguir.

Algoritmo

Passo 1: Gerar amostras $v_1, \ldots, v_n$ e $w_1, \ldots, w_n$ da distribuição normal padrão;
Passo 2: Calcular $x_i = \mu + \sigma \left[\delta|v_i| + (1-\delta^2)^{1/2} w_i\right]$, para $i = 1, \ldots, n$.

O código a seguir apresenta a implementação do método da transformação para a geração de uma amostra da distribuição normal assimétrica e a ilustração com um histograma, com a curva de densidade teórica,de uma amostra gerada com o código implementado.

```
rSN <- function(n, mu, sigma, lambda){
  V <- rnorm(n); W <- rnorm(n)
  delta <- lambda/sqrt(1 + lambda^2)
  Z <- delta*abs(V) + sqrt(1 - delta^2)*W
  X <- mu + sigma*Z
  return(X)
}

n <- 10000
mu <- 3; sigma <- 1; lambda <- 10
amostra <- rSN(n, mu, sigma, lambda)

par(mar = c(4, 4, 1, 1))
hist(amostra, prob = T, main = "", xlab = "x", col = "gray",
     breaks = 25, ylab = "Densidade")

grid <- seq(-5, 10, by = 0.01)
densidade <- dSN(grid, mu, sigma, lambda)
lines(grid, densidade, col = 2, lty = 1, lwd = 2)
```

Figura 1.9: Histograma de uma amostra simulada da distribuição normal assimétrica com $\mu = 3$, $\sigma = 1$ e $\lambda = 10$

1.4.3 Método polar para gerar vetores aleatórios normais

Sejam X e Y variáveis aleatórias com distribuição Normal padrão e sejam R e θ denotando as coordenadas polares do vetor (X, Y), isto é, $R^2 = X^2 + Y^2$ e tan$(\theta) = \frac{Y}{X}$. Para melhor entendimento, relembre os seguintes resultados:

- **Teorama de Pitágoras:** (hipotenusa)^2 = (cateto oposto)^2 + (cateto adjacente)^2;
- seno(θ) = (cateto oposto)$/$(hipotenusa);
- cosseno(θ) = (cateto adjacente)$/$(hipotenusa);
- seno$^2(\theta)$ + cosseno$^2(\theta) = 1$;
- tangente(θ) = (cateto oposto)$/$(cateto adjacente) = seno$(\theta)/$cosseno(θ).

Na figura abaixo é apresenada a representação em coordenadas polares do vetor (X, Y).

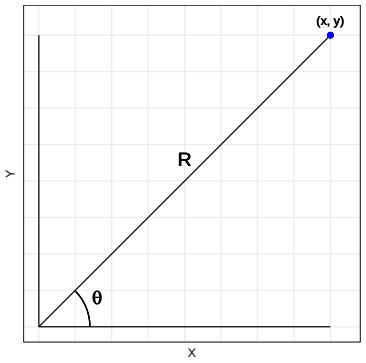

Figura 1.10: Coordenadas polares

A ideia deste método é gerar das variáveis R^2 e θ de uma forma simples e depois obter as variáveis X e Y por transformação. Visto que X e Y são independentes, a sua fdp conjunta é o produto das fdps individuais, ou seja,

$$f(x, y) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \frac{1}{\sqrt{2\pi}} e^{-y^2/2} = \frac{1}{2\pi} e^{-(x^2+y^2)/2}.$$

Podemos obter fdp conjunta de R^2 e θ através do método do Jacobiano

$$d = r^2 = x^2 + y^2, \quad \theta = \tan^{-1}\left(\frac{y}{x}\right).$$

Visto que $x = \sqrt{d}\cos\theta$ e $y = \sqrt{d}\sin\theta$, o Jacobiano da tranformação é ,

$$J = det\begin{pmatrix} \frac{\partial x}{\partial d} & \frac{\partial x}{\partial \theta} \\ \frac{\partial y}{\partial d} & \frac{\partial y}{\partial \theta} \end{pmatrix} = det\begin{pmatrix} \frac{d^{-1/2}}{2}\cos\theta & -d^{1/2}\sin\theta \\ \frac{d^{-1/2}}{2}\sin\theta & d^{1/2}\cos\theta \end{pmatrix} = \frac{1}{2}(\cos\theta)^2 + \frac{1}{2}(\sin\theta)^2 = \frac{1}{2}.$$

A fdp conjunta de R^2 e θ é igual a

$$f(d,\theta) = \frac{1}{2\pi}\frac{e^{-d/2}}{2}, \ 0 < d < \infty, \ 0 < \theta < 2\pi.$$

Note que a fdp conjunta é o produto de duas fdp's marginais de R^2 e θ, isto é, R^2 e θ são independentes, tal que $R^2 \sim Exponencial(1/2)$ e $\theta \sim Uniforme(0, 2\pi)$. Sendo assim, podemos gerar as variáveis R^2 e θ, e posteriormente obter X e Y através do seguinte algoritmo que considera as conhecidas transformações de Box-Muller.

Algoritmo

Passo 1: Gere números aleatórios u_1 e u_2 da distribuição Uniforme de 0 a 1;
Passo 2: Obtenha $R^2 = -2\log u_1$ e $\theta = 2\pi u_2$;
Passo 3: Faça

$$X = R\cos\theta = \sqrt{-2\log u_1}\cos(2\pi u_2);$$
$$Y = R\sin\theta = \sqrt{-2\log u_1}\sin(2\pi u_2).$$

Este método não é computacionalmente muito efeciente devido ao cálculo de seno e cosseno. Outra alternativa é gerar de variáveis aleatórias V_1 e V_2 independentes com distribuição Uniforme(-1, 1). Fazemos isto gerando amostras de $U_1 \sim Uniforme(0,1)$ e $U_2 \sim Uniforme(0,1)$, e calculando $V_1 = 2U_1 - 1$ e $V_2 = 2U_2 - 1$. Assim, o ponto (V_1, V_2) é uniformemente distribuido no quadrado de área igual a 4 centrado em (0,0). Geramos valores (v_1, v_2) do par (V_1, V_2) até obter valores tal que $v_1^2 + v_2^2 \leq 1$. Este par será uniformemente distribuído dentro do círculo de raio igual a 1.

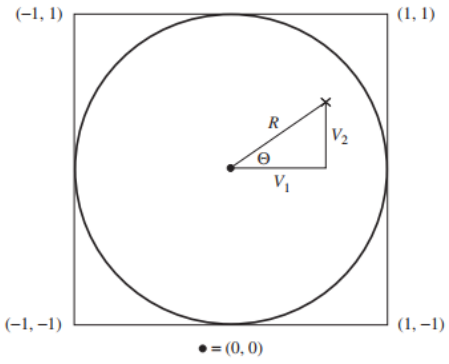

Figura 1.11: Ilustração método polar

Novamente denotando R e θ como as coordenadas polares deste par, podemos verificar que R^2 e θ são independentes com $R^2 \sim Uniforme(0,1)$ e $\theta \sim Uniforme(0, 2\pi)$. Então, após ter gerado um valor do ângulo θ, podemos calcular

$$\begin{aligned} \cos\theta &= V_1/R = V_1/\sqrt{V_1^2 + V_2^2}; \\ \sin\theta &= V_2/R = V_2/\sqrt{V_1^2 + V_2^2}. \end{aligned}$$

Este resultado é utilizado na transformação de Box-Muller. Além disso, dado que $R^2 = V_1^2 + V_2^2 \sim Uniforme(0,1)$ e é independente de θ, pode ser usado no lugar da variável U necessária na transformação. Então, fazendo $S = R^2$, obtemos

$$\begin{aligned} X &= \sqrt{-2\log S}\,(V_1/\sqrt{S}); \\ Y &= \sqrt{-2\log S}\,(V_2/\sqrt{S}). \end{aligned}$$

Desta forma, temos o seguinte algoritmo.

Algoritmo

Passo 1: Gere números aleatórios u_1 e u_2 da distribuição Uniforme(0, 1);
Passo 2: Obtenha $v_1 = 2u_1 - 1$, $v_2 = 2u_2 - 1$ e $s = v_1^2 + v_2^2$;
Passo 3: Se $s > 1$, retorne para o Passo 1;
Passo 4: Retorne os valores

$$\begin{aligned} x &= \sqrt{-2\log s}\,(v_1/\sqrt{s}); \\ y &= \sqrt{-2\log s}\,(v_2/\sqrt{s}). \end{aligned}$$

Dado que a probabilidade de um ponto gerado estar dentro do círculo de raio 1 é igual a $\pi/4$ (área do círculo dividido pela área do quadrado), segue que, em média, o método requer $4/\pi = 1,273$ iterações do Passo 1, e logo, requer a geração de 2,546 números aleatórios para gerar dois valores independentes da Normal padrão.

No R podemos implementar com o seguinte código.

```
rnormal_padrao <- function(n){
  itera <- round(n/2, 0) + 1;    amostra <- NULL
  for(i in 1:itera){
    s <- 1.5
    while(s > 1){
      u1 <- runif(1); u2 <- runif(1)
      v1 <- 2*u1-1; v2 <- 2*u2-1; s <- v1^2 + v2^2
    }
    x <- sqrt(-2*log(s)/s)*v1;    y <- sqrt(-2*log(s)/s)*v2
    amostra[i*2-1] <- x;  amostra[i*2] <- y
  }
  return(amostra[1:n])
}

dnormal <- function(x){ return((2*pi)^(-0.5)*exp(-x^2/2)) }

amostra <- rnormal_padrao(10000)
par(mar = c(4, 4, 1, 1))
hist(amostra, prob = T, main = "", ylab = "Densidade", xlab = "x",
     ylim = c(0, 0.5), breaks = 30)
grid <- seq(min(amostra), max(amostra), length.out = 100)
lines(grid, dnormal(grid), col = 2 )
```

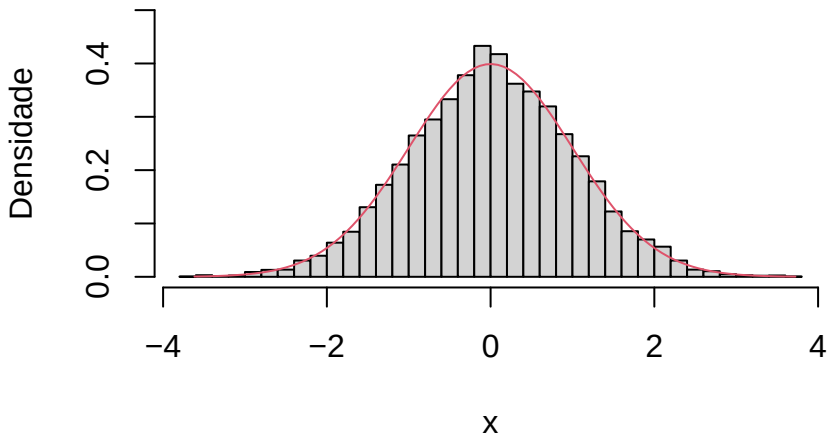

Figura 1.12: Gerando da Normal padrão

Podemos gerar da Normal bivariada com uma matriz de covariâncias igual a identidade com

o mesmo algoritmo. No R podemos implementar com o seguinte código.

```
rnormal_bivariada <- function(n){
  amostra <- matrix(0, nrow = n, ncol = 2)
  for(i in 1:n){
    s <- 1.5
    while(s > 1){
      u1 <- runif(1); u2 <- runif(1)
      v1 <- 2*u1-1; v2 <- 2*u2-1; s <- v1^2 + v2^2
    }
    x <- sqrt(-2*log(s)/s)*v1
    y <- sqrt(-2*log(s)/s)*v2
    amostra[i,] <- c(x, y)
  }
  return(amostra)
}

amostra <- rnormal_bivariada(500)
par(mar = c(4, 4, 1, 1))
plot(amostra, pch = 16, xlab = "x", ylab = "y", cex=0.5, col = 4)
```

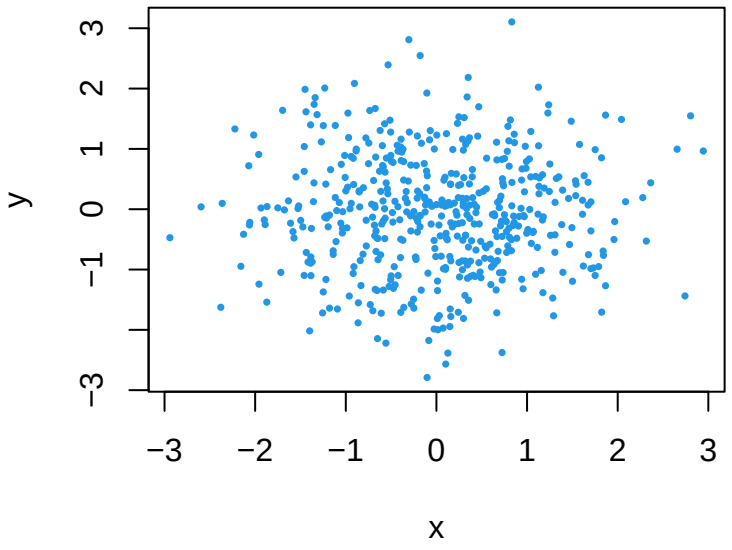

Figura 1.13: Gerando da Normal bivariada

Podemos gerar da distribuição normal com média μ e variância σ^2 aplicando uma transformação de locação e escala, isto é, se $Z \sim N(0,1)$, então $X = \mu + \sigma Z \sim N(\mu, \sigma^2)$. De forma similar pode ser feito para a distribuição Normal multivariada.

1.4.4 Algoritmo da aceitação-rejeição

O **Objetivo** é gerar uma amostra de uma distribuição cuja função densidade $f(x)$ pode ser calculada (no pior caso, sem a constante de proporcionalidade). A estratégia é amostrar candidatos de uma distribuição mais simples e então corrigir a probabilidade de amostragem através da rejeição de alguns candidatos.

Seja g uma função densidade da qual sabemos amostrar (e fácil de calcular $g(x)$) e $e(x)$ um envelope definido por ter a propriedade

$$e(x) = g(x)/\alpha = cg(x) \geq f(x),$$

$\forall x$ em que $f(x) > 0$ e para constantes positivas $c \geq 1$ e $\alpha \leq 1$. A função envelope $e(x)$, obtido multiplicando a função $g(x)$ por uma constante c, é um limite superior para a função $f(x)$ para todo valor de x no suporte de $f(x)$.

A seguir apresentamos o algoritmo, cuja demostração de seu funcionamento pode ser encontrada na página 147 do livro do *Givens.*

Algoritmo

Passo 1: Gere um valor y de $Y \sim g$;
Passo 2: Gere um valor u de $U \sim Uniforme(0, 1)$;
Passo 3: Rejeite y se $u > f(y)/e(y)$ e retorne ao passo 1;
Passo 4: Caso contrário, faça $x = y$ e considere x como um elemento da amostra aleatória da distribuição alvo. Retorne ao passo um até ter a amostra do tamanho desejado.

Observe no algoritmo e na figura abaixo que o método da rejeição elimina um valor candidato y com probabilidade proporcional ao comprimento da barra vertical acima de $f(y)$ em relação ao comprimento total da barra até $e(y)$. Note que o método pode ser visto como uma amostragem uniforme na região bidimensional sob a curva e e, em seguida, descartamos os valores amostrados que estão acima de f e abaixo de e. Amostrar de f é equivalente a amostrar uniformemente da região bidimensional sob a curva rotulada $f(x)$ e ignorar a coordenada vertical.

Propriedades:

- Em cada iteração, a probabilidade de valor gerado ser aceito é α e, assim, α pode ser visto como uma medida de eficiência do algoritmo;

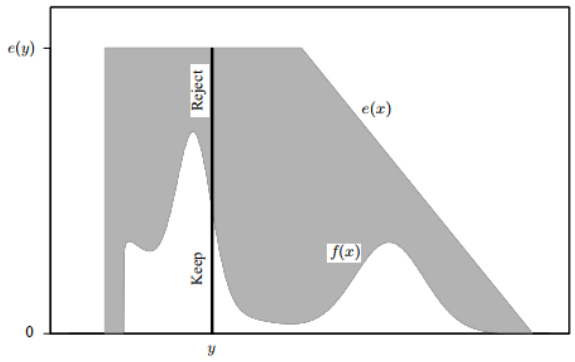

Figura 1.14: Ilustração do algoritmo de rejeição

- O número de iterações do algoritmo necessários para obter um valor de X é uma variável Geométrica com média $1/\alpha$;
- Suponha que f é conhecida sem a constante de proporcionalidade k. Isto é, sabemos calcular apenas $q(x) = f(x)/k$, em que k é desconhecido. Então, um valor candidato y deve ser rejeitado quando

$$U > q(y)/e(y).$$

- Após escolher uma g, podemos otimizar a eficiência do algoritmo encontrando o maior α tal que $g(x)/\alpha \geq f(x)$. Isto é equivalente a $c = 1/\alpha = \max\{f(x)/g(x)\}$.

Exemplo 1.9. Suponha que desejamos simular de uma distribuição Beta$(\alpha = 2, \beta = 2)$ (função de densidade na Figura 1.15) através do método de rejeição. Em média, quantas iterações serão necessárias para gerar 1000 valores por este método?

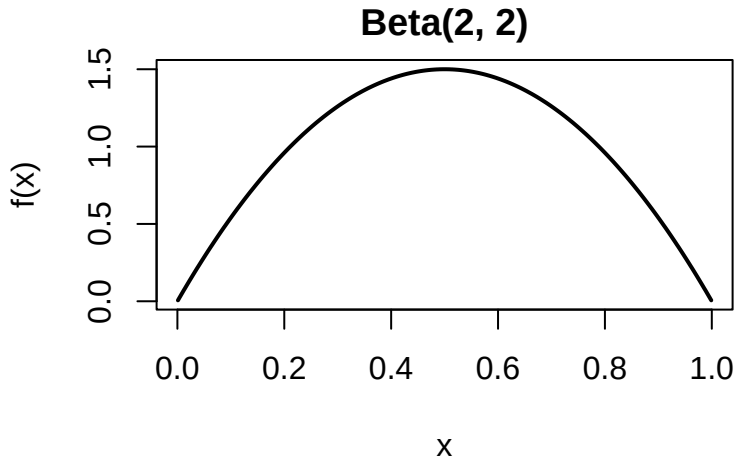

Figura 1.15: Ilustração do algoritmo de rejeição

A densidade da Beta(2, 2) é dada por

$$f(x) = 6x(1-x), \;\; 0 < x < 1.$$

Se escolhemos $g(x)$ como a distribuição Uniforme(0, 1), então $f(x)/g(x) \leq 6$, escolhemos $\alpha = 1/6$.

Um candidato y gerado de $g(y)$ é aceito se o valor amostrado u da variável aleatória $U \sim Uniforme(0, 1)$ for tal que

$$u \leq \frac{f(y)}{g(y)/\alpha} = \frac{6y(1-y)}{6} = y(1-y).$$

Em média, $n/\alpha = 6000$ iterações são necessárias!

Uma escolha mais eficiente para α seria maximizar $\frac{f(x)}{g(x)} = 6x(1-x)$ e desta forma obtemos $\alpha = 2/3$.

Neste caso, um candidato y gerado de $g(y)$ é aceito se

$$u \leq \frac{f(y)}{g(y)/\alpha} = \frac{6y(1-y)}{3/2} = 4y(1-y).$$

Em média, $n/\alpha = 1500$ iterações são necessárias!

Algoritmo no R:

```
## com alpha = 1/6
# Guardando valores aceitos e rejeitados para ilustração

n <- 10000
k <- 0; k2 <- 0
j <- 0 # iterações
x <- numeric(n); u1 <- numeric(n)
z <- NULL; u2 <- NULL

while (k < n) {
  u <- runif(1)
  j <- j+1
```

```
  y <- runif(1) # gerando valor aleatório de g
  if (y * (1-y) >= u) { # valor aceito
    k <- k+1; x[k] <- y; u1[k] <- u
  }else{
    k2 <- k2+1; z[k2] <- y; u2[k2] <- u
  }
}
amostra1 <- x
abscissas <- c(x, z); ordenadas <- c(u1, u2)
cores <- c( rep(4, length(x)), rep(2, length(z)) )
cat("Número de iterações: ", j, "\n")
```

```
## Número de iterações:  60197
```

```
## com alpha = 2/3
n <- 10000; k <- 0; j <- 0
x <- numeric(n)
while (k < n) {
  u <- runif(1); y <- runif(1)
  j <- j+1
  if (4*y * (1-y) >= u) {
    k <- k+1
    x[k] <- y
  }
}
amostra2 <- x
cat("Número de iterações: ", j, "\n")
```

```
## Número de iterações:  15105
```

Abaixo ilustramos o funcionamento do algoritmo com os dois diferentes valores de α considerados.

```
par( mfrow = c(1,3), mar = c(4, 4, 3, 1) )
hist(amostra1, prob = T, col = "gray", xlab = "x",
     ylab = "Densidade", main = expression(alpha == 1/6))

plot(abscissas, ordenadas, main = expression("Candidatos com " ~ alpha == 1/6),
     xlab = "valor x gerado", ylab = "valor u gerado", col = cores, pch=16)

hist(amostra2, prob = T, col="gray", xlab = "x",
```

```
     ylab = "Densidade", main = expression(alpha == 1/3))
```

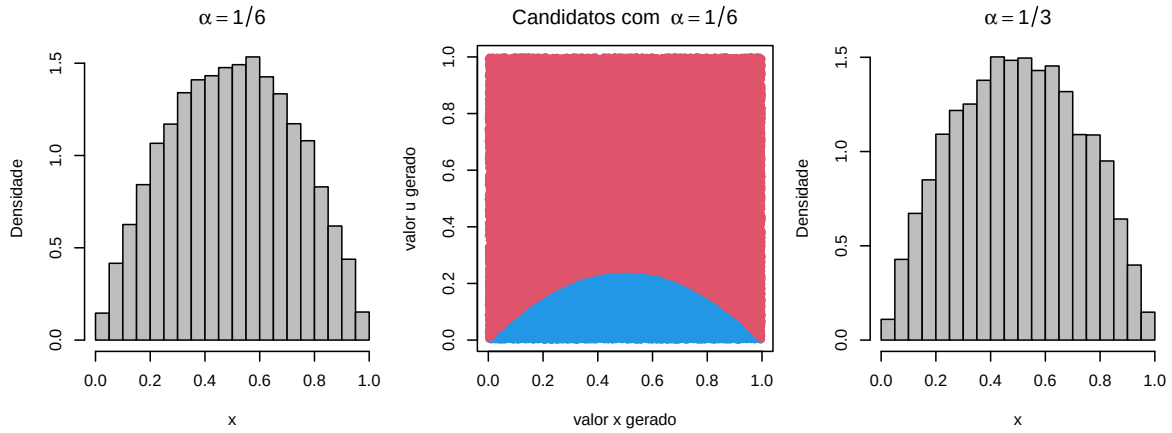

Figura 1.16: Ilustração do algoritmo de rejeição (n=10000)

Implementamos o código abaixo considerando a melhor escolha de α.

```
## Para quaisquer valores dos parâmetros da beta e
## maximizando numericamente para encontrar o melhor alpha
rbeta_rej <- function(n, a, b){
  k <- 0; j <- 0; x <- numeric(n)
  maximo <- optim(0.5, dbeta, method = "L-BFGS-B",
                      lower = 0.00001, upper = 0.99999, control=list(fnscale=-1),
                      shape1 = a, shape2 = b)$value
  alpha = 1/maximo
  while (k < n) {
    u <- runif(1); y <- runif(1); j <- j+1
    if (dbeta(y, a, b) * alpha >= u) {
      k <- k+1; x[k] <- y
    }
  }
  cat("Número de iterações: ", j, "\n")
  return(x)
}

a <- 4; b <- 2
amostra <- rbeta_rej(1000, a, b)
```

```
## Número de iterações:  2079
```

Ilustrando com um gráfico temos:

```
par(mar = c(4, 4, 1, 1))
hist(amostra, prob = T, main = "", col = "lemonchiffon4", breaks = 25,
     xlab = "x", ylab = "Densidade")
grid <- seq(0, 1, by = 0.01)
lines(grid, dbeta(grid, a, b), col = "midnightblue", lwd = 2)
```

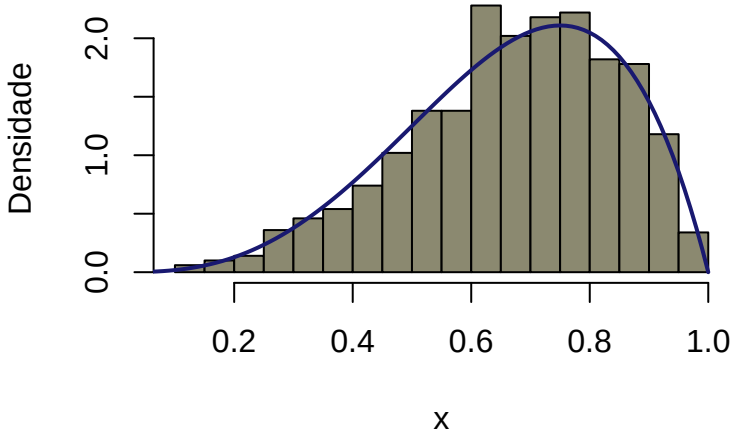

Figura 1.17: Ilustração do algoritmo de rejeição otimizado

1.4.5 Método da rejeição adaptativa

A amostragem de rejeição comum requer uma avaliação de f para cada sorteio de candidato y. Nos casos em que avaliar f é computacionalmente caro, mas a amostragem de rejeição é atraente, a velocidade de simulação melhorada é alcançada por **amostragem de rejeição comprimida**. Esta estratégia evita a avaliação de f em alguns casos, empregando uma função de compressão não negativa s. Para que s seja uma função de compressão adequada, $s(x)$ não deve exceder $f(x)$ em qualquer lugar no suporte de f. Um envelope, e, também é usado; como na amostragem de rejeição comum, $e(x) = g(x)/\alpha \geq f(x)$ no suporte de f.

Vantagem

Na abordagem amostragem de rejeição adaptativa, o envelope e a função de compressão são refinados iterativamente durante a geração de amostras. Com isso, a quantidade de candidatos descartados e a frequência com que f deve ser avaliada diminuem à medida que as iterações aumentam. Gilks and Wild (1992) propuseram uma estratégia de geração

automática de envelope para amostragem de rejeição comprimida para uma densidade log-côncava contínua, diferenciável em uma região contínua de suporte.

Envelope

Seja $l(x) = \log f(x)$, e assuma $f(x) > 0$ em um intervalo (possivelmente infinito) da reta real. Seja f log-côncava no sentido de que

$$[\, l(c) - l(b) \,] - [\, l(b) - l(a) \,] = l(a) - 2l(b) + l(c) < 0$$

para quaisquer três pontos na região de suporte de f para os quais $a < b < c$. Sob as premissas adicionais que f é contínua e diferenciável, observe que $l'(x)$ existe e diminui monotonicamente com o aumento de x, mas pode ter descontinuidades.

O algoritmo é iniciado avaliando $l(x)$ e $l'(x)$ nos pontos, $x_1 < x_2 < \cdots < x_k$. Seja $T_k = \{x_1, \ldots, x_k\}$. Se o suporte de f se estende a $-\infty$, escolha x_1 tal que $l'(x_1) > 0$. Da mesma forma, se o suporte de f se estende a ∞, escolha x_k tal que $l'(x_k) < 0$. Defina o envelope de rejeição em T_k como sendo o exponencial do limite superior, linear por partes, formado pelas tangentes em cada ponto em T_k (Veja a figura abaixo). Se denotarmos o limite superior de l como e_k^*, então o envelope de rejeição é $e_k(x) = \exp\{e_k^*\}$.

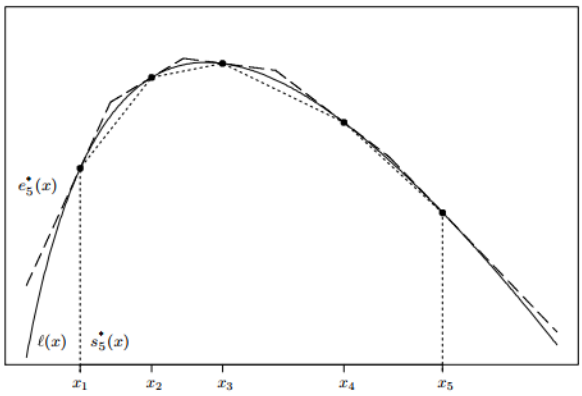

Figura 1.18: Limite externo e interno para $l(x)$ com $k = 5$

Pode-se mostrar que as tangentes em x_i e x_{i+1} se cruzam em

$$z_i = \frac{l(x_{i+1}) - l(x_i) - x_{i+1}l'(x_{i+1}) + x_i l'(x_i)}{l'(x_i) - l'(x_{i+1})},$$

para $i = 1, \ldots, k-1$. Portanto,

$$e_k^*(x) = l(x_i) + (x - x_i)l'(x_i), \quad \text{para } x \in [z_{i-1}, z_i]$$

e $i = 1, \ldots, k$, com z_0 e z_k definidos, respectivamente, para serem iguais aos limites inferior e superior do suporte para f.

Função de compressão

Defina a função de compressão em T_k como sendo o exponencial do limite inferior linear por partes formado pelas cordas entre pontos adjacentes em T_k. Este limite inferior é dado por

$$s_k^*(x) = \frac{(x_{i+1} - x)l(x_i) + (x - x_i)l(x_{i+1})}{x_{i+1} - x_i} \text{ para } x \in [x_i, x_{i+1}]$$

e $i = 1, ..., k-1$.

Quando $x < x_1$ ou $x > x_k$, seja $s_k^*(x) = -\infty$. Assim, a função de compressão é $s_k(x) = \exp\{s_k^*(x)\}$. Tanto o envelope de rejeição quanto a função de compressão são funções exponenciais por partes. O envelope possui caudas exponenciais que ficam acima das caudas de f. A função de compressão possui suporte limitado.

O algoritmo

A amostragem de rejeição adaptativa é inicializada pela escolha de um k pequeno e uma grade adequada T_k correspondente. A primeira iteração do algoritmo inicia com a geração de um valor candidato y. Os candidatos sorteados são obtidos a partir da densidade obtida escalando o envelope exponencial por partes e_k para que se integre a 1. Isto é, calculamos

$$u_k = \frac{\exp\{e_k^*(x)\}}{\sum_{i=0}^{k} \int_{z_i}^{z_{i+1}} \exp\{e_k^*(x')\}dx'}.$$

Podemos inicialmente, amostrar de um dos intervalos com probabilidade $p_k = \int_{z_{i-1}}^{z_i} u_k dx$ e posteriormente gerar um valor candidato de uma exponencial truncada no intervalo sorteado. Este candidato é aceito com probabilidade $s(y)/e(y)$. Caso não seja aceito no primeiro momento, ele poderá ser aceito em um segundo momento com probabilidade $f(y)/e(y)$. Note que a ideia inicial desta estratégia é que o função s é mais simples de ser avaliada do que a função f. Quando um candidato é aceito neste segundo estágio, o ponto aceito é adicionado ao conjunto T_k, criando T_{k+1}. As funções atualizadas e_{k+1} e s_{k+1} também são calculadas. Quando um sorteio de candidato é rejeitado, nenhuma atualização para T_k, e_k ou s_k é feita. Além disso, vemos agora que um novo ponto que corresponde a qualquer membro existente de T_k não fornece nenhuma atualização significativa para T_k, e_k ou s_k.

Uma vez que cada sorteio aceito é feito usando uma abordagem de amostragem de rejeição, os valores amostrados são uma amostra i.i.d. de f. Se f for conhecido apenas com o núcleo da densidade, a abordagem de amostragem de rejeição adaptativa ainda pode ser usada, uma vez que a constante de proporcionalidade simplesmente muda l, e_k^* e s_k^*.

1.4.5.1 Abordagem livre de derivadas

Gilks (1992) desenvolveu uma abordagem semelhante que não requer avaliação de l'. Mantemos as suposições de que f é log-côncava com uma região de suporte contínua.

Para o conjunto de pontos T_k, defina $L_i(\cdot)$ como a função de linha reta conectando $(x_i, l(x_i))$ e $(x_{i+1}, l(x_{i+1}))$ para $i = 1, \ldots, k-1$. Defina

$$e_k^*(x) = \begin{cases} \min\{L_{i-1}(x),\ L_{i+1}(x)\} & \text{para } x \in [x_i, x_{i+1}]; \\ L_1(x) & \text{para } x < x_1; \\ L_{k-1}(x) & \text{para } x > x_k, \end{cases}$$

com a convenção de que $L_0(x) = L_k(x) = \infty$.

Então e_k^* é um limite superior linear por partes porque a concavidade de garante que $L_i(x)$ esteja abaixo de $l(x)$ em (x_i, x_{i+1}) e acima de $l(x)$ quando $x < x_i$ ou $x > x_{i+1}$. O envelope de amostragem de rejeição é então $e_k(x) = \exp\{e_k^*(x)\}$.

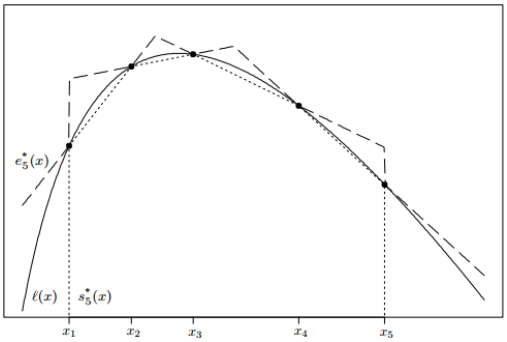

Figura 1.19: Limite externo e interno para $l(x)$ com $k = 5$

A função de compressão permanece a mesma. As iterações do algoritmo de amostragem de rejeição adaptativa livre de derivadas procedem de forma análoga à abordagem anterior, com T_k, o envelope e a função de compressão atualizados cada vez que um novo ponto é mantido. O envelope não é tão eficiente como quando é usado l'.

Independentemente do método usado para construir e_k, observe que seria preferível que a grade T_k fosse mais densa nas regiões onde $f(x)$ é maior, próximo a moda de f. Felizmente, isso acontecerá automaticamente, uma vez que esses pontos provavelmente serão mantidos em iterações subsequentes e incluídos nas atualizações do T_k. Pontos de grade muito distantes nas caudas de f não são muito úteis.

1.4.6 Reamostragem Ponderada

O algoritmo *sampling importance resampling (SIR)* simula realizações aproximadas de alguma distribuição alvo. O processo de geração tem dois pontos principais:

- Amostras são geradas de uma função de amostragem por importância g;
- Cada ponto na amostra é ponderado para corrigir a probabilidade de amostragem de tal forma que a amostra ponderada seja relacionada da densidade alvo f.

Para a densidade alvo f, os pesos usados para corrigir as probabilidades de amostragem são chamados pesos de importância padronizados e são definidos por

$$\omega(y_i) = \frac{f(y_i)/g(y_i)}{\sum_{j=1}^{m} f(y_j)/g(y_j)},$$

para uma coleção de valores $x_1, \dots, x_m$ gerados de g.

Podemos ver este método como uma aproximação de f por uma distribuição discreta tendo massa $\omega(y_i)$ em cada ponto observado y_i para $i = 1, \dots, m$.

Algoritmo

Passo 1: Gere candidatos $Y_1, \dots, Y_m$ iid de g.
Passo 2: Calcule os pesos de importância padronizados $\omega(Y_1), \dots, \omega(Y_m)$.
Passo 3: Reamostre, com reposição, $X_1, \dots, X_n$ de $Y_1, \dots, Y_m$ com probabilidades $\omega(Y_1), \dots, \omega(Y_m)$.

Convergência:

A variável aleatória X amostrada com o algoritmo SIR tem distribuição que converge para f quando $m \to \infty$. Note que:

- $n/m \to 0$ para garantir a convergência em distribuição da distribuição da amostra e, para n fixo, a convergência ocorre quando $m \to \infty$. A tolerância máxima para a razão n/m depende da qualidade da densidade g. Algumas vezes $n/m \leq 1/10$ é suficiente para resultar em uma reamostra que não contenha muitas replicações de valores.
- O suporte de g deve incluir todo o suporte de f e g deve ter caudas mais pesadas que f, ou de forma mais geral, g ser escolhida garantindo que $f(x)/g(x)$ nunca cresça demais. Se $g(x)$ está próximo de zero em um local onde $f(x)$ é positivo, então um valor gerado desta região acontecerá muito raramente, mas quando isso acontecer receberá um enorme peso.

Exemplo 1.10. Sejam X e U variáveis aleatórias independentes, X com distribuição N(0,1) e U com distribuição Uniforme(0,1). Dizemos que $Y = X/U$ tem distribuição de Slash e sua função densidade de probabilidade é dada por

$$f(y) = \begin{cases} \frac{1-\exp\{-y^2/2\}}{y^2\sqrt{2\pi}}, & y \neq 0 \\ \frac{1}{2\sqrt{2\pi}}, & y = 0. \end{cases}$$

Esta densidade tem caudas muito pesadas.

Para entendermos melhor a vantagem do SIR, veja abaixo a geração de amostra da distribuição Slash diretamente por transformação. Note que são gerados alguns valores muito atípicos.

```
dslash <- function(y){
  (y!=0)*(1-exp(-y^2/2))/(y^2*sqrt(2*pi)) + (y==0)*(1-exp(-y^2/2))/(y^2*sqrt(2*pi))
}

rslash <- function(n){
  u <- runif(n); x <- rnorm(n)
  y <- x/u
  return(y)
}

par(mar = c(4, 4, 1, 1))
hist(rslash(500), ylab = "Densidade", xlab = "x", main = "", breaks = 30, prob = T)
```

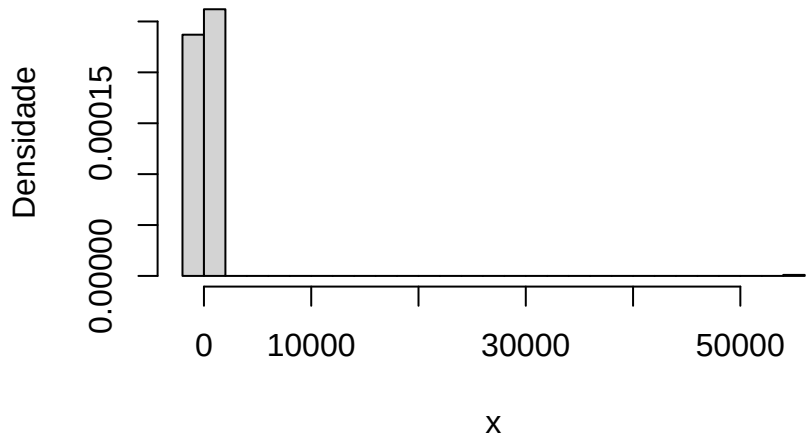

Figura 1.20: Geração da distribuição Slash diretamente

Agora ilustramos o uso do SIR, considerando:

- Uso do SIR com proposta Slash para gerar da Normal padrão.
- Uso do SIR com proposta Normal padrão para gerar da Slash.

No R, temos:

```
## Para gerar de uma normal padrao com proposta slash
Pesos1 <- function(y){
  df <- dnorm(y); dg <- dslash(y)
  p <- df/dg
  pesos <- p/sum(p)
}
rnorm_sir <- function(n, m){
  y <- rslash(m)
  pesos <- Pesos1(y)
  amostra <- sample(x = y, size = n, replace = T, prob = pesos)
  return(amostra)
}


## Para gerar de uma  slash com proposta normal padrao
Pesos2 <- function(y){
  df <- dslash(y);  dg <- dnorm(y)
  p <- df/dg
  pesos <- p/sum(p)
}
rslash_sir <- function(n, m){
  y <- rnorm(m)
  pesos <- Pesos2(y)
  amostra <- sample(x = y, size = n, replace = T, prob = pesos)
  return(amostra)
}

n <- 5000
m <- 100000
amostra1 <- rnorm_sir(n, m)
amostra2 <- rslash_sir(n, m)
```

```
## Gráficos
par(mar = c(4, 4, 3, 2), mfrow = c(1, 2))

hist(amostra1, prob = T, main = "", xlab = "x",
     col = "darkgoldenrod", breaks = 50, ylab = "Densidade" )
grid <- seq(-5, 5, by = 0.01)
densidade <- dnorm(grid)
lines(grid, densidade, col = 2, lty = 1, lwd = 2)

hist(amostra2, prob = T, main = "", xlab = "x",
     col = "aquamarine1", breaks = 50, xlim = c(-7,7), ylab = "Densidade")
grid <- seq(-7, 7, by = 0.01)
densidade <- dslash(grid)
lines(grid, densidade, col = 2, lty = 1, lwd = 2)
```

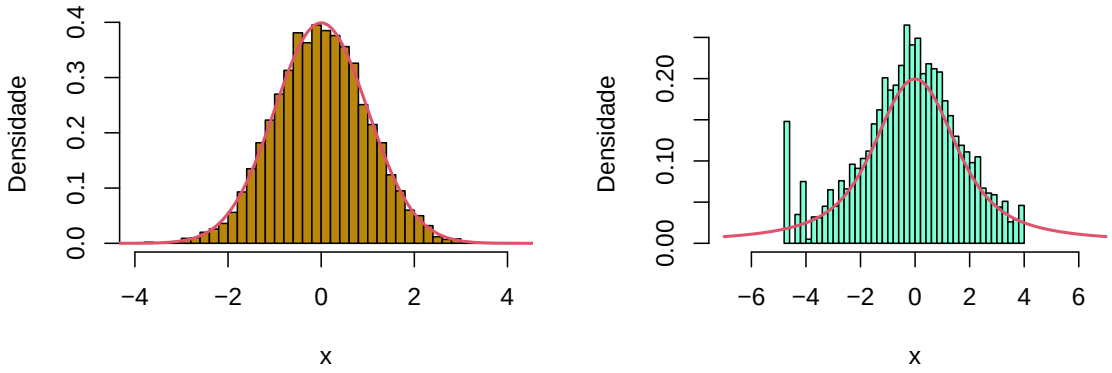

Figura 1.21: Geração de normal padrão com proposta slash e geração de slash com proposta normal padrão

Note que o resultado é bem ruim quando tentamos amostrar da Slash partindo de um proposta Normal padrão, mas o inverso funciona bem.

1.4.7 Exercícios

Exercício 1.8. Reimplemente o método da transformação inversa para gerar da distribuição Normal truncada sem utilizar as funções pnorm e dnorm. Utilize a função integrate para calcular a função de distribuição acumulada em um ponto. Use a função qnorm, visto que neste caso fica complicado implementar este método sem o uso desta função.

Exercício 1.9. Implemente uma função para gerar amostras de tamanho n da variável ale-

atória com função densidade de probabilidade é dada por

$$f(x) = \begin{cases} \frac{x-2}{2} & \text{se } 2 \leq x \leq 3; \\ \frac{2-x/3}{2} & \text{se } 3 \leq x \leq 6. \end{cases}$$

Apresente as contas necessárias.

Exercício 1.10. Implemente uma função para gerar amostras de tamanho n da variável aleatória com função de distribuição é dada por

$$F(x|\alpha, \beta) = 1 - \exp\{-\alpha x^{\beta}\},\ 0 < x < \infty.$$

Apresente as contas necessárias.

Exercício 1.11. Escreva uma função para gerar variáveis aleatórias com distribuição Lognormal(μ, σ^2) usando o método polar e o método da transformação. Gere amostras de tamanho 1000 e compare através de histogramas com a função densidade da distribuição lognormal dada pela função dlnorm do R.

Exercício 1.12. Suponha que é fácil gerar de variáveis aleatórias X_i com função de distribuição F_i, $i = 1, \ldots, n$. Considere que $Y = \max\{X_1, \ldots, X_n\}$. Temos que

$$F_Y(x) = \prod_{i=1}^{n} F_i(x).$$

a. Como podemos gerar uma amostra aleatória da variável Y?
b. Usando a ideia do item anterior, implemente uma função para gerar da variável aleatória que tem função de distribuição dada por

$$F(x) = x^n,\ \ 0 \leq x \leq 1.$$

c. Refaça o item anterior utilizando o método de aceitação e rejeição.
d. Como podemos amostrar de uma variável aleatória que possui função de distribuição dada por $F_Y(x) = 1 - \prod_{i=1}^{n}[1 - F_i(x)]$? Dica: Qual a variável aleatória Y possui esta função de distribuição?

Exercício 1.13. Usando o método da rejeição, gere da distribuição Normal truncada entre a e b. Apresente as contas para encontrar a probabilidade de aceitação.

Exercício 1.14. Pesquise algum pacote do R em que o método da rejeição adaptativa está implementado e gere amostras das distribuições Gama e Beta. Considere diferentes valores para os parâmetros destas distribuições.

Exercício 1.15. Implemente o algoritmo SIR para amostrar da distribuição Slash utilizando a distribuição $t - student$ com grau de liberdade pequeno.

Exercício 1.16. Implemente o algoritmo SIR para amostrar da distribuição normal assimétrica.

1.5 Misturas de distribuições

Podemos classificar as misturas de distribuições em discretas ou contínuas de acordo com a distribuição misturadora considerada. O método para gerar de misturas é conhecido como método da composição.

Mistura discreta

Uma variável aleatória X é uma mistura discreta de distribuições se

$$F_X(x) = \sum_i \omega_i F_{X_i}(x)$$

para alguma sequência de variáveis aleatórias $X_1, X_2, ...$ e ω_i é tal que $\sum_i \omega_i = 1$. As constantes ω_i são chamadas pesos das misturas.

Exemplo 1.11. Seja a mistura dada por

$$F_X(x) = \sum_{i=1}^{5} \omega_i F_{X_i}(x),$$

em que $X_i \sim Gama(3, \lambda_i = 1/i)$ são independentes e $\omega_i = i/15, i = 1, ..., 5$.

Para simular um valor fazemos:

Algoritmo

Passo 1: Geramos um inteiro $i \in \{1, 2, 3, 4, 5\}$ da variável aleatória K, em que $P(K = i) = \omega_i$, para $i = 1, ..., 5$.
Passo 2: Gere um valor aleatório de $Gama(3, \lambda_i)$.

No R temos:

```
n <- 5000
k <- sample(1:5, size = n, replace = TRUE, prob = (1:5)/15)
rate <- 1/k
```

```r
x <- rgamma(n, shape = 3, rate = rate)

# Gráfico da densidade estimada com densidade dos componentes da mistura
par(mar = c(4, 4, 1, 1))
plot(density(x), xlim = c(0, 40), ylim = c(0, .3), lwd = 3, xlab = "x",
     ylab = "Densidade", main = "", col = 2)

grid <- seq(0.01, max(x), length.out = 500)
for (i in 1:5) lines(grid, dgamma(grid, 3, 1/i), col= gray(0.6))
```

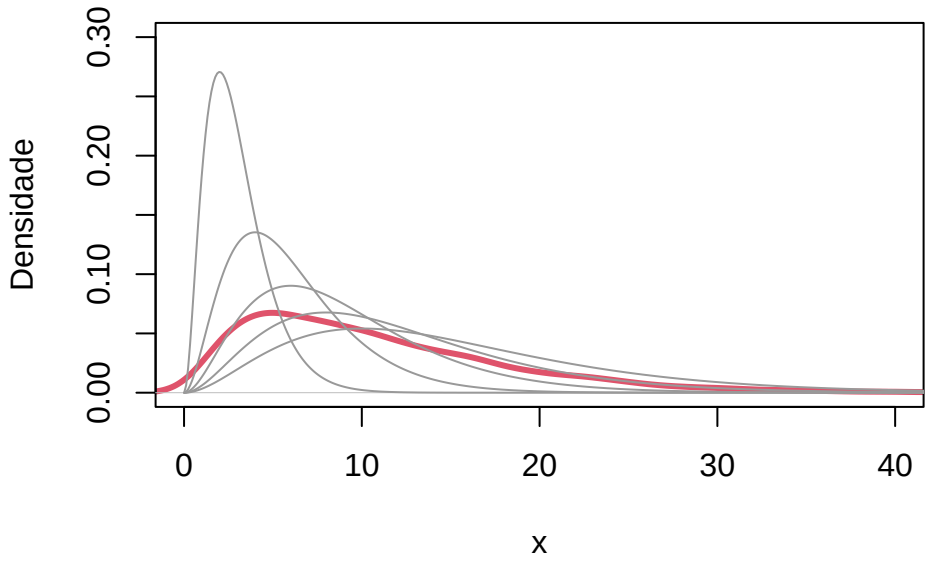

Figura 1.22: Mistura finita de distribuições Gama

Mistura contínua

Uma variável aleatória X é uma mistura contínua de distribuições se a distribuição de X é

$$F_X(x) = \int_{-\infty}^{\infty} F_{X|Y=y}(x) f_Y(y) dy,$$

para uma família de distribuições $F_{X|Y=y}(x)$ indexada por números reais y e uma função de pesos f_Y tal que $\int_{-\infty}^{\infty} f_Y(y) dy = 1$.

Exemplo 1.12. A distribuição binomial negativa é uma mistura de distribuições Poisson(λ), em que λ tem distribuição Gama. Se

$$\begin{aligned} X \mid \lambda &\sim Poisson(\lambda) \\ \lambda &\sim Gama(r, \beta) \end{aligned}$$

então, X tem distribuição binomial negativa com parâmetros r e $p = \beta/(1+\beta)$.

Para gerar uma amostra de X basta fazer:

```
n = 1000
r = 4; beta = 3
lambda <- rgamma(n, r, beta)
x <- rpois(n, lambda)

par(mar = c(4, 4, 1, 1))
plot(prop.table(table(x)), ylab = "Frequência relativa", xlab = "x")
```

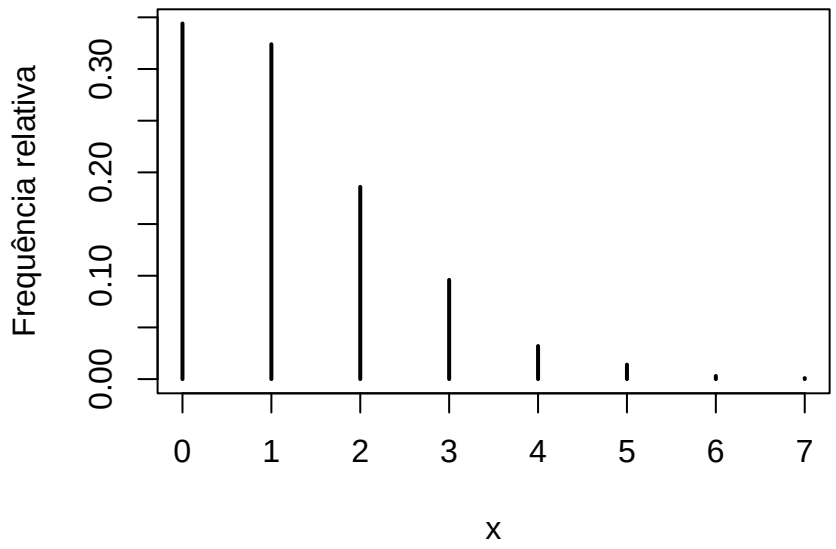

Figura 1.23: Mistura infinita de distribuições Poisson

```
# comparando proporções obtidas com as probabilidades teóricas
mix <- tabulate(x+1) / n
negbin <- round(dnbinom(0:max(x), r, beta/(1+beta)), 3)
tab <- round(rbind(mix, negbin), 3)
rownames(tab) <- c("Freq", "Prob")
```

```
tab <- tab %>%
  kbl(align = c(rep("c", 5)),
      caption = "Comparação de frequências relativas com probabilidades teóricas")

tab %>%
  kable_styling(latex_options = "hold_position",
                bootstrap_options = c("bordered"), font_size = 11)
```

Tabela 1.2: Comparação de frequências relativas com probabilidades teóricas

Freq	0.344	0.324	0.186	0.096	0.032	0.014	0.003	0.001
Prob	0.316	0.316	0.198	0.099	0.043	0.017	0.006	0.002

1.5.1 Exercícios

Exercício 1.17. Usando o método da composição, implemente uma função para gerar de uma variável aleatória com função de distribuição:

a. $F(x) = \frac{x+x^2}{2}, \ 0 \leq x \leq 1;$

b. $F(x) = \frac{x+x^3+x^5}{3}, \ 0 \leq x \leq 1;$

c. $F(x) = \begin{cases} \frac{1-e^{-2x}+2x}{3}, & 0 \leq x \leq 1 \\ \frac{3-e^{-2x}}{3}, & 1 \leq x \leq \infty. \end{cases}$

Exercício 1.18. Implemente um algoritmo para gerar uma amostra de tamanho n da variável aleatória com função de distribuição

$$F(x) = \int_0^\infty x^y e^{-y} dy, \ 0 \leq x \leq 1.$$

Dica: Suponha que a função de distribuição condicional de X dado $Y = y$ é $P(X \leq x|Y = y) = x^y, \ 0 \leq x \leq 1.$

1.6 Geração de trajetórias de processos de Poisson e suas extensões.

Nesta seção aborda-se os Processos de Poisson homogêneo e não-homogêneo unidimensional, além do Processo de Poisson homogêneo bidimensional.

1.6.1 Processo de Poisson Homogêneo

Suponha que “eventos” estejam ocorrendo em momentos aleatórios e que $N(t)$ indique o número de eventos que ocorrem no intervalo de tempo $[0, t]$. Estes eventos constituem um processo de Poisson com taxa $\lambda > 0$ se

1. $N(0) = 0$.
2. O número de eventos que ocorrem em intervalos disjuntos são independentes.
3. A distribuição do número de eventos que ocorrem em um dado intervalo depende do tamanho do intervalo, mas não da sua localização.
4. $\lim_{h\to 0} \frac{P(N(h)=1)}{h} = \lambda$ e $\lim_{h\to 0} \frac{P(N(h)\geq 2)}{h} = 0$.

As suposições acima podem ser reescritas como:

- O processo começa no tempo 0.
- Incremento independente: afirma que o número $N(t)$ de eventos no tempo t é independente do número de eventos que ocorrem entre t e $t+s$ dado por $N(t+s)-N(t)$.
- A distribuição de probabilidade de $N(t+s)-N(t)$ é a mesma para todos os valores de t.
- A probabilidade de ocorrer um evento em um pequeno intervalo de comprimento h é aproximadamente λh, enquanto a probabilidade de ocorrência de dois ou mais eventos é aproximadamente 0.

Estas suposições implicam que o número de eventos em um intervalo de tamanho t é uma variável aleatória Poisson com média $t\lambda$. Para mostrar isto, considere o intervalo $[0, t]$ e divida-o em n subintervalos não sobrepostos de comprimento t/n.

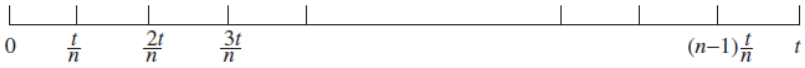

Figura 1.24: Subintervalos em [0, t].

Considere primeiro o número desses subintervalos que contêm um evento:

- Como cada subintervalo independentemente contém um evento com a mesma probabilidade ($\approx \lambda t/n$), segue que o número de tais intervalos é uma variável aleatória binomial com parâmetros n e $p \approx \lambda t/n$.
- Se $n \to \infty$, o número destes subintervalos converge para uma variável aleatória de Poisson com média λt.
- A probabilidade de que qualquer um desses subintervalos conter dois ou mais eventos vai para 0 quando $n \to \infty$.

Então, segue que $N(t)$ é uma variável aleatória Poisson com média λt.

Para um processo de Poisson, seja X_1 o tempo de ocorrência do primeiro evento. Para $n > 1$, seja X_n o tempo decorrido entre $(n-1)$-ésimo evento e n-ésimo evento. A sequência $\{X_n, n = 1, 2, ...\}$ é chamada a sequência de tempos entre chegadas.

Note que o evento $\{X_1 > t\}$ ocorre se e somente se nenhum evento do processo de Poisson ocorrer no intervalo $[0, t]$; portanto

$$P(\{X_1 > t\}) = P(N(t) = 0) = e^{-\lambda t}.$$

Logo, X_1 tem distribuição exponencial com média $1/\lambda$. Note também que

$$\begin{aligned} P(X_2 > t \mid X_1 = s) &= P(\text{0 eventos em } (s, s+t] \mid X_1 = s) \\ &= P(\text{0 eventos em } (s, s+t]) \\ &= e^{-\lambda t}. \end{aligned}$$

Então X_2 é uma variável exponencial com média $1/\lambda$ e além disso, X_2 é independente de X_1. Repetindo os argumentos obtemos que os tempos $X_1, X_2, ...$ entre eventos são variáveis aleatórias independentes e identicamente distribuídas com distribuição exponencial com parâmetro λ.

Gerando um processo de Poisson - Abordagem 1

Suponha que desejamos gerar os primeiros n tempos de eventos de um processo de Poisson com taxa λ. Para fazer isso, fazemos uso do resultado de que os tempos entre os eventos sucessivos para tal processo são variáveis aleatórias exponenciais independentes, cada uma com a taxa λ. Note que:

1. Se geramos n números aleatórios $X_1, ..., X_n$ da distribuição exponencial com parâmetro λ, então X_i será considerado o tempo entre os $(i-1)$-ésimo e i-ésimo eventos do processo de Poisson.
2. Os tempos de ocorrência dos n primeiros eventos são dados por

$$\sum_{i=1}^{j} X_i, \text{ para } j = 1, ..., n.$$

Algoritmo

Gerando Processo de Poisson com tempo de acompanhamento T.
Passo 1: Faça $t = 0, I = 0$;
Passo 2: Gere um número aleatório u de $U \sim Uniforme(0, 1)$;
Passo 3: Faça $t = t - (1/\lambda) \log u$. Se $t > T$, pare (Alterar condição de parada para $I > n$ se tivermos interessado em gerar do processo até ocorrerem n eventos);
Passo 4: Faça $I = I + 1, S(I) = t$;
Passo 5: Vá para o Passo 2.

No R podemos implementar como a seguir.

```
pph <- function(lambda = 0.5, T = 10){
  t <- 0; I <- 0
  S = NULL
  repeat{
    u <- runif(1)
    t <- t - (1/lambda)*log(u)
    if(t > T) break
    I <- I+1; S[I] <- t
  }
  cat("Número de eventos:", I,"\n")
  return(S)
}

set.seed(1256)

tempos1 <- pph(lambda = 0.5)
```

```
## Número de eventos: 2
```

```
tempos2 <- pph(lambda = 1)
```

```
## Número de eventos: 12
```

```
tempos3 <- pph(lambda = 2)
```

```
## Número de eventos: 23
```

```
par(mar=c(5,0,0,0))
plot(c(tempos1, tempos2, tempos3),
     c(rep(2, length(tempos1)), rep(1, length(tempos2)),
       rep(0, length(tempos3))), axes = F, ylim = c(-1, 3), xlim = c(0, 10),
     ylab = "", xlab = "Tempo de ocorrência", pch = 16)
segments(x0 = 0, y0 = 2, x1 = 10, y1 = 2, col = 4)
segments(x0 = 0, y0 = 1, x1 = 10, y1 = 1, col = 4)
segments(x0 = 0, y0 = 0, x1 = 10, y1 = 0, col = 4)
axis(1, at = c(0,2,4,6,8,10))
```

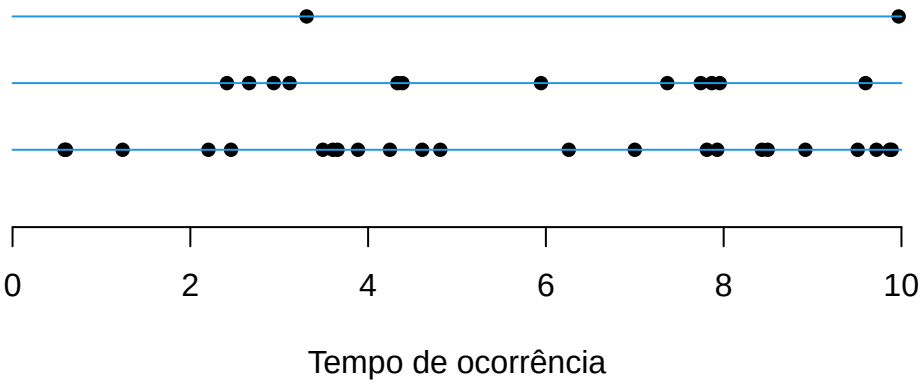

Figura 1.25: Geração do Processo de Poisson com taxa igual a 0,5, 1 e 2, respectivamente nas linhas.

Gerando um processo de Poisson - Abordagem 2

Outra maneira de simular um processo de Poisson com taxa λ durante um tempo T, em que λ representa o número médio de eventos em uma unidade de tempo, é iniciar simulando o número total $N(T)$ de eventos que ocorrem até o tempo T. Note que $N(T) \sim Poisson(\lambda T)$. Esta abordagem é mais eficiente computacionalmente do que simular os tempos entre as chegadas a partir da distribuição Exponencial.

Se o valor simulado de $N(T)$ for n, então n números aleatórios $U_1, \ldots, U_n$ são gerados de uma Uniforme(0, 1) e $\{TU_1, \ldots, TU_n\}$ será o conjunto de tempos de ocorrência de eventos até o tempo T. Isto ocorre pois, condicional a $N(T) = n$, o conjunto de tempos não ordenados até os eventos tem distribuição Uniforme(0, T). Posteriormente, precisamos ordenar os valores $TU_i, i = 1, \ldots, n$, para obter os tempos de ocorrência dos eventos.

Para melhor entendimento, seja $N(t)$ o número de valores em $\{TU_1, \ldots, TU_{N(T)}\}$ que são menores do que t. Vamos mostrar que $N(t)$, para $0 \leq t \leq T$, é um Processo de

Poisson.

Para mostrar que o processo tem incrementos **estacionários e independentes**, definimos $I_1, ..., I_r$ subintervalos disjuntos em $[0, T]$ e dizemos que:

- Um evento j é um evento do tipo i se TU_j pertence ao i-ésimo intervalo disjunto, para $i = 1, ..., r$;
- O evento do tipo $r + 1$ se não pertence a nenhum dos r intervalos.

Visto que U_i, para $i \geq 1$, são independentes, cada um dos $N(T)$ eventos são classificados de forma independente nos tipos $1, ..., r, r + 1$ com respectivas probabilidades iguais a $p_1, ..., p_r, p_{r+1}$, que são calculadas como o tamanho do intervalo I_i dividido por T quando $i \leq r$ e $p_{r+1} = 1 - \sum_{i=1}^{r} p_i$. De resultados de probabilidade, segue que os números de eventos nos intervalos disjuntos, definidos por $N_1, ..., N_r$, são variáveis aleatórias Poisson independentes, com $E[N_i]$ igual a λ multiplicado pelo comprimento do intervalo I_i; que estabelece que $N(t), 0 \leq t \leq T$, tem incrementos estacionários e independentes. Como o número de eventos em qualquer intervalo de comprimento h é Poisson com média λh, temos

$$\lim_{h\to 0} \frac{P(N(h) = 1)}{h} = \lim_{h\to 0} \frac{\lambda h e^{-\lambda h}}{h} = \lambda$$

e

$$\lim_{h\to 0} \frac{P(N(h) \geq 2)}{h} = \lim_{h\to 0} \frac{1 - e^{-\lambda h} - \lambda h e^{-\lambda h}}{h} = 0.$$

1.6.2 Processo de Poisson não-homogêneo

Do ponto de vista da modelagem, a maior fraqueza do processo de Poisson é a suposição de que os eventos têm a mesma probabilidade de ocorrer em todos os intervalos de mesmo tamanho. Uma generalização, que relaxa essa suposição, leva ao processo não-homogêneo ou não estacionário.

Se $N(t)$ indica o número de eventos que ocorrem até o tempo t, dizemos que $\{N(t), t > 0\}$ constitui um processo de Poisson não-homogêneo com função de intensidade $\lambda(t)$, $t > 0$, se

- $N(0) = 0$.
- O número de eventos que ocorrem em intervalos disjuntos são independentes.
- $\lim_{h\to 0} \frac{P(\text{exatamente 1 evento entre } t \text{ e } t+h)}{h} = \lambda(t)$.
- $\lim_{h\to 0} \frac{P(\text{2 ou mais eventos entre } t \text{ e } t+h)}{h} = 0$.

Proposição 1.2. *A variável aleatória $N(t+s)-N(t)$ é uma variável Poisson com média $m(t+s)-m(t)$, em que $m(t)$ é definido por*

$$m(t)=\int_0^t \lambda(s)ds,\ \ t\geq 0$$

e é chamada função do valor médio.

A quantidade $\lambda(t)$, chamada de intensidade no tempo t, indica quão provável é que um evento ocorra ao redor do tempo t. A proposição abaixo é um caminho para interpretar o processo de Poisson não-homogêneo.

Proposição 1.3. *Suponha que os eventos estejam ocorrendo de acordo com um processo de Poisson com taxa λ, e suponha que, independentemente de qualquer coisa que aconteceu antes, um evento que ocorreu no tempo t é contado com a probabilidade $p(t)$. Então o processo de contagem de eventos constitui um processo de Poisson não-homogêneo com função de intensidade $\lambda(t)=\lambda p(t)$.*

Demonstração. Esta proposição é provada observando que as condições previamente dadas são todas satisfeitas. As condições (a), (b) e (d) seguem uma vez que o resultado correspondente é verdadeiro para todos os eventos (não apenas os contados).

A condição (c) segue desde que

$$\begin{aligned}&P(\text{1 evento contado entre t e t+h})\\&=P(\text{1 evento e ele é contado})\\&+P(\text{2 ou mais eventos e exatamente 1 é contado})\\&\approx \lambda h p(t).\end{aligned}$$

□

Geralmente, é muito difícil obter resultados analíticos para um modelo matemático que pressuponha um processo de Poisson não-homogêneo e, como resultado, tais processos não são aplicados com a frequência que deveriam.

Gerando do Processo de Poisson não-homogêneo - Alternativa 1

Suponha que desejamos simular os primeiros eventos, até o tempo T, de um processo de Poisson não-homogêneo com função de intensidade $\lambda(t)$. O primeiro método que apresentamos, chamado de abordagem de refinamento ou amostragem aleatória, começa escolhendo um valor λ tal que

$$\lambda(t)\leq\lambda,\ \ \forall t\leq T.$$

Se um evento de um processo de Poisson com taxa λ que ocorre no tempo t é contado (independentemente do que aconteceu anteriormente) com probabilidade $p(t) = \lambda(t)/\lambda$, então o processo de eventos contados é um processo de Poisson não-homogêneo com função intensidade $\lambda(t)$, $0 \leq t \leq T$. Assim, simulando um processo de Poisson e, em seguida, contando aleatoriamente seus eventos, podemos gerar o processo de Poisson não-homogêneo desejado.

Algoritmo

Passo 1: Faça $t = 0$, $I = 0$;
Passo 2: Gere um número aleatório u_1 de $U \sim Uniforme(0, 1)$;
Passo 3: Faça $t = t - (1/\lambda) \log u_1$. Se $t > T$, pare;
Passo 4: Gere um número aleatório u_2 de $U \sim Uniforme(0, 1)$;
Passo 5: Se $u_2 \leq \lambda(t)/\lambda$, faça $I = I + 1$, $S(I) = t$;
Passo 6: Vá para o Passo 2.

Exemplo 1.13. Gere de um Processo de Poisson não-homogêneo com

$$\lambda(t) = \begin{cases} t, & 0 < t < 5; \\ 10, & \text{caso contrário.} \end{cases}$$

```
# Função para gerar do PPNH com taxa lambda_t até o tempo T
ppnh <- function(lambda_t, lambda, T = 10){
  t <- 0; I <- 0; S = NULL
  repeat{
    u1 <- runif(1)
    t <- t - (1/lambda)*log(u1)
    if(t > T) break
    u2 <- runif(1)
    if(u2 <= lambda_t(t)/lambda){
      I <- I+1; S[I] <- t
    }
  }
  cat("Número de eventos:", I,"\n")
  return(S)
}
```

```r
# Função lambda(t)
lambdat <- function(t){
  if(t < 5){ lambda <- t }else{ lambda <- 10 }
  return(lambda)
}

# Escolha obvia é lambda = 10
tempos1 <- ppnh(lambda_t = lambdat, lambda = 10)
```

```
## Número de eventos: 59
```

```r
tempos2 <- ppnh(lambda_t = lambdat, lambda = 10)
```

```
## Número de eventos: 62
```

```r
tempos3 <- ppnh(lambda_t = lambdat, lambda = 10)
```

```
## Número de eventos: 60
```

```r
par(mar=c(5,0,0,0))
plot(c(tempos1, tempos2, tempos3),
     c(rep(2, length(tempos1)), rep(1, length(tempos2)),
       rep(0, length(tempos3))), axes = F, ylim = c(-1, 3), xlim = c(0, 10),
     ylab = "", xlab = "Tempo de ocorrência", pch = 16)
segments(x0 = 0, y0 = 2, x1 = 10, y1 = 2, col = 4)
segments(x0 = 0, y0 = 1, x1 = 10, y1 = 1, col = 4)
segments(x0 = 0, y0 = 0, x1 = 10, y1 = 0, col = 4)
axis(1, at = c(0,2,4,6,8,10))
```

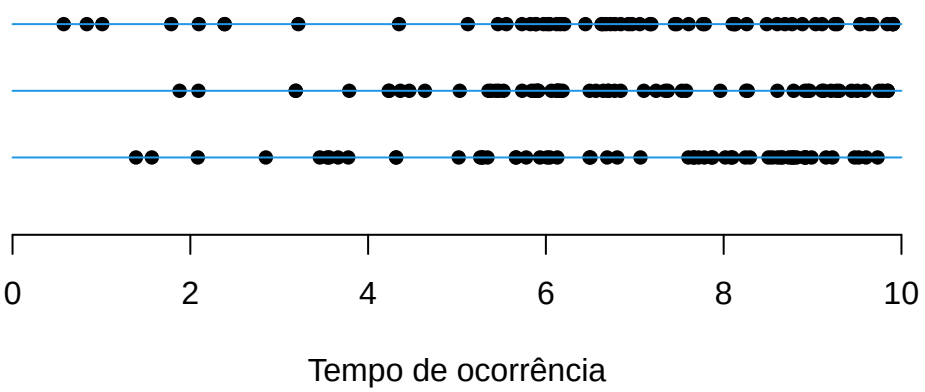

Figura 1.26: Geração do Processo de Poisson não homogêneo - 3 simulações.

O procedimento acima é claramente mais eficiente quando $\lambda(t)$ está próximo λ. Assim, uma forma de melhorar é dividir o intervalo em subintervalos e, em seguida, usar o procedimento em cada subintervalo. Ou seja, determine os valores k apropriados, $0 = t_0 < t_1 < t_2 < ... < t_k < t_{k+1} = T, \lambda_1, ..., \lambda_{k+1}$ tal que

$$\lambda(s) \leq \lambda_i \text{ se } t_{i-1} \leq s < t_i, \ \ i = 1, ..., k+1.$$

Então, basta gerar o processo de Poisson não-homogêneo sobre o intervalo (t_{i-1}, t_i) gerando variáveis aleatórias exponenciais com taxa λ_i, e aceitando o evento gerado no tempo s, $s \in (t_{i-1}, t_i)$, com probabilidade $\lambda(s)/\lambda_i$.

Por causa da propriedade de falta de memória da exponencial e o fato de que a taxa de uma exponencial pode ser alterada por multiplicação por uma constante, segue-se que não há perda de eficiência na passagem de um subintervalo para o seguinte. Se estamos em $t \in (t_{i-1}, t_i)$ e geramos X, uma exponencial com taxa λ_i, tal que $t + X > t_i$, então podemos usar $\lambda_i[X - (t_i - t)]/\lambda_{i+1}$ como a próxima exponencial com taxa λ_{i+1}.

Algoritmo

Supondo que $J + 1$ intervalos tal que $0 = t_0 < t_1 < t_2 < ... < t_J < t_{J+1} = T$.
Passo 1: $t = 0, j = 1, I = 0$;
Passo 2: Gere um número aleatório u de $U \sim Uniforme(0, 1)$ e faça $x = -(1/\lambda_j) \log u$;
Passo 3: Se $t + x > t_j$, vá para o Passo 8; Se $t + x > T$ pare o algoritmo; Caso contrário, vá para o Passo 4;
Passo 4: Faça $t = t + x$;
Passo 5: Gere um número aleatório u_2 da Uniforme(0,1);
Passo 6: Se $u_2 \leq \lambda(t)/\lambda_j$, faça $I = I + 1, S(I) = t$;
Passo 7: Vá para o Passo 2;
Passo 8: Se $j = J + 1$, pare;
Passo 9: $x = (x - t_j + t)\lambda_j/\lambda_{j+1}, t = t_j, j = j + 1$;
Passo 10: Vá para o Passo 3.

A geração do processo de Poisson não-homogêneo pode ser mais eficiente em algumas situações. Por exemplo, considere o caso onde $\lambda(s) = 10 + s, 0 < s < 1$. Temos que:

- Usar o método de refinamento com $\lambda = 11$ geraria um número esperado de 11 eventos, cada um dos quais requereria um número aleatório para determinar se deveria ou não ser aceito.

- Por outro lado, para gerar um processo de Poisson homogêneo com taxa 10 e depois fundi-lo com um processo de Poisson não-homogêneo com taxa $\lambda(s) = s, 0 < s < 1$ produziria um número igualmente distribuído de tempos de evento, mas com o número esperado que precisa ser verificado igual a 1.

Suponha que para algum subintervalo (t_{i-1}, t_i) temos que $\lambda_i > 0$, onde

$$\lambda_i \equiv \text{Infimo}\{\lambda(s) : t_{i-1} \leq s < t_i\}.$$

Nessa situação, não devemos usar diretamente o algoritmo de refinamento, mas sim simular um processo de Poisson homogêneo com taxa λ_i no intervalo desejado e depois simular um processo de Poisson não-homogêneo com a função de intensidade $\lambda^*(s) = \lambda(s) - \lambda_i$ quando $s \in (t_{i-1}, t_i)$.

Gerando do Processo de Poisson não-homogêneo - Alternativa 2

Um segundo método para simular um processo de Poisson não-homogêneo com função de intensidade $\lambda(t)$, $t > 0$, é gerar diretamente os sucessivos tempos de evento. Então, seja $S_1, S_2, ...$ denotando os sucessivos tempos de eventos de tal processo. Como essas variáveis aleatórias são claramente dependentes, geramos em sequência, começando com S_1 e usando o valor gerado de S_1 para gerar S_2, e assim por diante.

Note que se um evento ocorrer no tempo s, então, independente do que ocorreu antes de s, o tempo adicional até o próximo evento tem a distribuição F_s, dada por

$$\begin{aligned}
F_s(x) &= P(\text{ próximo evento ocorre antes de } x+s \mid \text{ evento em } s) \\
&= P(\text{ evento ocorrer entre } s \text{ e } s+x \mid \text{ evento em } s) \\
&= P(\text{evento entre } s \text{ e } s+x) \\
&= 1 - P(0 \text{ eventos em } (s, s+x)) \\
&= 1 - \exp\left(-\int_s^{s+x} \lambda(y)dy\right) = 1 - \exp\left(-\int_0^x \lambda(s+y)dy\right).
\end{aligned}$$

Podemos simular os tempos do evento $S_1, S_2, ...$, gerando S_1 a partir da distribuição F_0. Se o valor simulado de S_1 for s_1, geramos S_2 adicionando s_1 a um valor gerado a partir da distribuição F_{s_1}. Se essa soma é s_2, geramos S_3 adicionando s_2 a um valor gerado a partir da distribuição F_{s_2} e assim por diante.

Exemplo 1.14. Suponha que $\lambda(t) = 1/(t+a), t \geq 0$, para alguma constante positiva a. Então

$$\int_0^x \lambda(s+y)dy = \int_0^x \frac{1}{s+y+a}dy = \log\left(\frac{x+s+a}{s+a}\right).$$

Assim,

$$F_s(x) = 1 - \frac{s+a}{x+s+a} = \frac{x}{x+s+a}.$$

Supondo que $x = F_s^{-1}(u)$, então, usando o método da transformação inversa,

$$u = F_s(x) = \frac{x}{x+s+a} \text{ ou } F_s^{-1}(u) = x = (s+a)\frac{u}{1-u}.$$

Podemos então gerar números aleatórias $U_1, U_2, \ldots$ da distribuição uniforme(0, 1) e, então, fazemos

$$\begin{aligned} S_1 &= \frac{aU_1}{1-U_1}; \\ S_2 &= S_1 + (S_1+a)\frac{U_2}{1-U_2} = \frac{S_1 + aU_2}{1-U_2}; \\ &\vdots \\ S_j &= \frac{S_{j-1} + aU_j}{1-U_j},\ j \geq 2. \end{aligned}$$

1.6.3 Simulação do Processo de Poisson homogêneo bidimensional

Um processo consistindo da ocorrência aleatória de pontos no plano constitui um processo de Poisson bidimensional com taxa λ, $\lambda > 0$, se

1. O número de pontos que ocorrem em qualquer região da área A é Poisson distribuído com média λA;
2. O número de pontos que ocorrem em regiões disjuntas é independente.

Para um dado ponto fixo **0** no plano, agora mostramos como simular pontos, de acordo com um processo de Poisson bidimensional com taxa λ, que ocorre em uma região circular de raio r centrada em **0**.

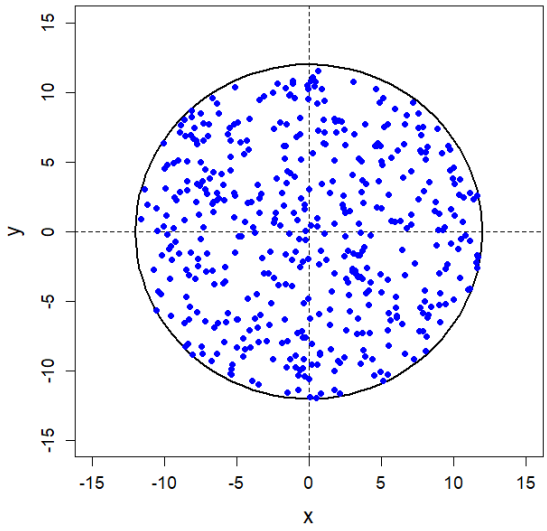

Figura 1.27: Processo de Poisson Homogêneo em uma região circular

Seja $C(a)$ denotando o círculo de raio a centrado em **0** e observe que, da condição 1, o número de pontos em $C(a)$ é Poisson distribuído com média $\lambda\pi a^2$. Seja $R_i, i \geq 1$, denote a distância da origem **0** ao seu i-ésimo ponto mais próximo. Note que a área do círculo com raio R_1 é dada por πR_1^2 e

$$\begin{aligned} P(\pi R_1^2 > x) &= P(R_1 > \sqrt{x/\pi}) \\ &= P\left(\text{nenhum ponto em } C\left(\sqrt{\frac{x}{\pi}}\right)\right) \\ &= e^{-\lambda x}. \end{aligned}$$

Seja $C(b) - C(a)$ denotando a região entre $C(b)$ e $C(a)$, $a < b$. Temos o seguinte resultado:

$$\begin{aligned} & P(\pi R_2^2 - \pi R_1^2 > x \mid R_1 = a) \\ & P\left(R_2 > \sqrt{\frac{x + \pi R_1^2}{\pi}} \mid R_1 = a\right) \\ & P\left(\text{Nenhum ponto em } C\left(\sqrt{\frac{x + \pi R_1^2}{\pi}}\right) - C(a) \mid R_1 = a\right) \\ & P\left(\text{Nenhum ponto em } C\left(\sqrt{\frac{x + \pi a^2}{\pi}}\right) - C(a)\right) \quad \text{(Condição 2)} \\ & = e^{-\lambda x}. \end{aligned}$$

O mesmo argumento pode ser repetido para obter a proposição a seguir.

Proposição 1.4. *Com $R_0 = 0, \pi R_i^2 - \pi R_{i-1}^2, i \geq 1$, são variáveis aleatórias exponenciais independentes, cada uma com uma taxa λ.*

Em outras palavras, a quantidade de área que precisa ser percorrida para encontrar um ponto de Poisson é exponencial com a taxa λ. Como, por simetria, os respectivos ângulos dos pontos de Poisson são independentes e uniformemente distribuídos sobre $(0, 2\pi)$, temos, portanto, o seguinte algoritmo para simular o processo de Poisson sobre uma região circular de raio r ao redor de **0**.

Algoritmo

Passo 1: Gere variáveis $X_1, X_2, \ldots$ exponenciais independentes com taxa λ, parando quando

$$N = \text{min}\,\{n : X_1 + \cdots + X_n > \pi r^2\};$$

Passo 2: Se $N = 1$ pare; não existem pontos em $C(r)$. Caso contrário, visto que $\pi R_i^2 = X_1 + \cdots + X_i$, para $i = 1, \ldots, N-1$, faça

$$R_i = \sqrt{\frac{X_1 + \cdots + X_i}{\pi}};$$

Passo 3: Gere números $U_1, \ldots, U_{N-1}$ uniformemente distribuídos entre 0 e 1;
Passo 4: As coordenadas polares dos $N-1$ pontos são

$$(R_i, 2\pi U_i), \;\; i = 1, \ldots, N-1;$$

Passo 5: As coordenadas retangulares são

$$(x_i = R_i \cos\{2\pi U_i\}, y_i = R_i \sin\{2\pi U_i\}).$$

No software R podemos implementar como:

```
r = 12 # (raio)
T = pi*r^2
s = 0; lambda = 0.5; areas = 0; i = 1
```

```
while(s<T){
  x = rexp(1, lambda)
  s = s+x
  areas[i] = s
  i = i+1
}
areas = areas[areas <= T]

R = sqrt(areas/pi); U = runif(length(areas))
phi = 2*pi*U; x = R*cos(phi);  y = R*sin(phi)

c = 15
plot(NULL, xlim=c(-c, c), ylim=c(-c, c), ylab = "y", xlab = "x")
abline(v=0,lty=2); abline(h=0,lty=2);
a <- seq(0, 2*pi, length = 100)
lines( r*cos(a), r*sin(a), type = 'l', lwd=2)
points(x, y, pch=16, col = 4, cex=1)
```

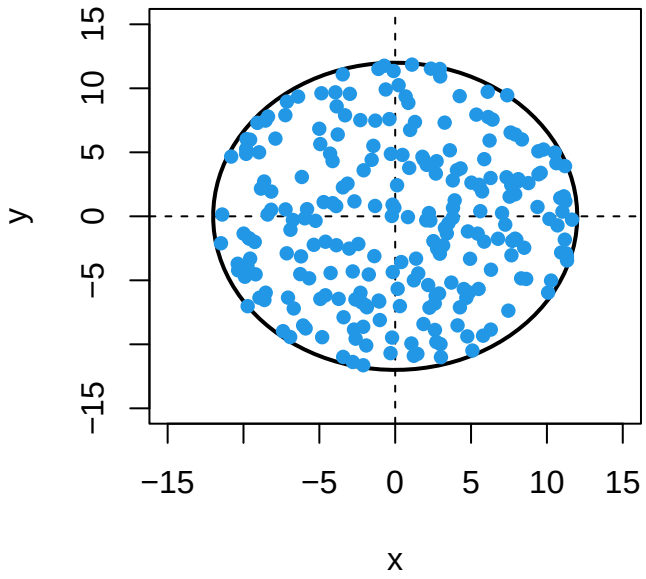

Figura 1.28: Processo de Poisson Homogêneo em uma região circular com taxa = 0.5

Essa técnica também pode ser usada para simular o processo em regiões não circulares. Por exemplo, considere uma função não-negativa $f(x)$ e suponha que estamos interessados em simular o processo de Poisson na região entre o eixo x e a função f com x indo de 0 a T.

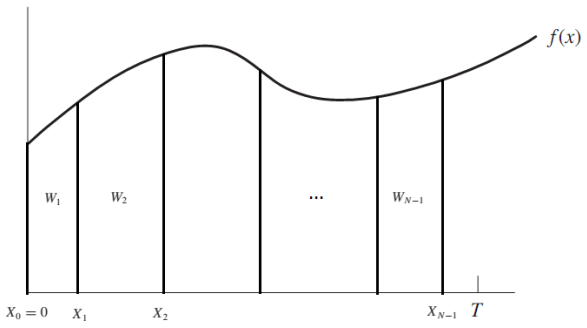

Figura 1.29: Processo de Poisson Homogêneo bidimensional em área não circular

Seja $X_1 < X_2 < \cdots < X_{N-1}$ denotando as projeções sucessivas dos pontos de processo de Poisson no eixo x. Assim,

$W_i = \int_{X_{i-1}}^{X_i} f(x)dx,\ i = 1, \ldots,$ são variáveis Exponenciais independentes com taxa λ,

em que $X_0 = 0$.

Podemos simular os pontos de Poisson gerando variáveis aleatórias $W_1, W_2, \ldots, W_n$ exponenciais independentes com taxa λ, parando quando

$$N = \min\left\{ n : W_1 + \cdots + W_n > \int_0^T f(x)dx \right\}.$$

Então, calculamos $X_1, X_2, \ldots, X_{N-1}$ a partir das seguintes equações,

$$\begin{aligned} \int_0^{X_1} f(x)dx &= W_1; \\ \int_{X1}^{X_2} f(x)dx &= W_2; \\ &\vdots \\ \int_{X_{N-2}}^{X_{N-1}} f(x)dx &= W_{N-1}. \end{aligned}$$

Para cada X_i, $Y_i \sim Uniforme(0, f(X_i))$, então segue que, após gerarmos variáveis $U_1, ..., U_{N-1}$ uniformes em (0, 1), os pontos de Poisson simulados são,

$$(X_i, U_i f(X_i)), \quad i = 1, ..., N-1.$$

1.6.4 Exercícios

Exercício 1.19. Considerando diferentes valores de λ, implemente o segundo algoritmo para gerar do Processo de Poisson homogêneo e compare a eficiência computacional dos dois algoritmos.

Exercício 1.20. (Exercício 29 do Capítulo 5 do Ross (2013)) Ônibus chegam a um evento esportivo de acordo com um processo de Poisson com taxa de 5 por hora. Cada ônibus tem a mesma probabilidade de conter $20, 21, ..., 40$ torcedores, com os números de torcedores nos diferentes ônibus sendo independentes. Escreva um algoritmo para simular a chegada de torcedores ao evento no tempo $t = 1$.

Exercício 1.21. (Exercício 31 do Capítulo 5 do Ross (2013)) Implemente um algoritmo eficiente para gerar os k primeiros tempos de um Processo de Poisson não homogêneo com função de intensidade

$$\lambda(t) = \begin{cases} \frac{t}{5}, & 0 < t < 5; \\ 1 + 5(t-5), & t \geq 5. \end{cases}$$

Exercício 1.22. Implemente o algoritmo para gerar um processo de Poisson não-homogêneo com função de intensidade dada no Exemplo 2.15. Faça com diferentes valores de a.

Exercício 1.23. Implemente um algoritmo para gerar um processo de Poisson homogêneo na área abaixo da curva $f(x) = x^2$ com $0 \leq x \leq 5$.

Capítulo 2

Métodos Monte Carlo

Neste capítulo iremos apresentar o método de integração Monte Carlo e o seu uso na Inferência Estatística. O conteúdo apresentado é baseado no Capítulo 3 do livro do Ross (2013) e nos Capitulos 5 e 6 do livro de Rizzo (2008).

2.1 Método de integração de Monte Carlo

Uma aplicação de números aleatórios é o cálculo de integrais. Seja $g(x)$ uma função e suponha que queremos calcular θ tal que

$$\theta = \int_0^1 g(x)dx.$$

Note que, podemos rescrever $\theta = E[g(U)]$, em que U tem distribuição uniforme sobre (0, 1). Se $U_1, \dots, U_k$ são variáveis aleatórias iid, segue que $g(U_1), \dots, g(U_k)$ são variáveis iid com esperança θ.

O método de integração Monte Carlo é baseado na Lei dos grandes números.

Teorema 2.1. ***(Lei Forte dos Grandes Números)*** *Sejam $X_1, X_2, \dots$ variáveis aleatórias independentes, identicamente distribuídas e integráveis, com $E[X_i] = \mu$, para $i = 1, \dots, n$. Então*

$$\frac{X_1 + \cdots + X_n}{n} \to \mu \text{ quase certamente}.$$

No contexto apresentado, pela lei dos grandes números, segue que, com probabilidade 1,

$$\sum_{i=1}^{k} \frac{g(U_i)}{k} \to E[g(U)] = \theta.$$

Então, para aproximar θ basta gerar uma grande quantidade k de números aleatórios u_i e calcular a média das quantidades $g(u_i)$.

Observações:

- Se queremos calcular $\theta = \int_a^b g(x)dx$, fazemos a substituição $y = (x-a)/(b-a)$, $dy = dx/(b-a)$, e obtemos que

  $$\theta = \int_0^1 g(a + [b-a]y)(b-a)dy = \int_0^1 h(y)dy,$$

  em que $h(y) = (b-a)g(a + [b-a]y)$;

- Também podemos considerar $\theta = \int_a^b g(x)dx = (b-a)\int_a^b g(x)\frac{1}{b-a}dx = (b-a)E[g(X)]$, com $X \sim Uniforme(a,b)$. Neste caso, podemos gerar uma amostra $x_1, ..., x_n$ da $Uniforme(a,b)$ e obter a aproximação para a integral através da multiplicação de $(b-a)$ pela média amostral de $g(x_1), ..., g(x_n)$;

- Similarmente, se queremos calcular $\theta = \int_0^\infty g(x)dx$, fazemos a substituição $y = 1/(x+1)$, $dy = -dx/(x+1)^2 = -y^2dx$, e obtemos que

  $$\theta = \int_0^1 h(y)dy,$$

  em que $h(y) = g\left(\frac{1}{y} - 1\right)/y^2$.

Exemplo 2.1. Como podemos aproximar as seguintes integrais:

a) $\int_0^1 (x^3 + \cos(x))dx$;

b) $\int_1^3 x^2 + 3x^3 dx$

Resolução:

a) Note que $\int_0^1 (x^3 + \cos(x))dx$ pode ser visto como $E[U^3 - \cos(U)]$, em que $U \sim Uniforme(0, 1)$. Assim, podemos fazer:

```
x <- seq(0, 1, length.out = 100)
par(mar = c(4,5,1,1))
plot(x, x^3 + cos(x), type = "l", ylim = c(0, 1.8),
     ylab = bquote(f(x)==x^3+cos(x)))
segments(x0 = 0, y0 = 0, x1 = 0, y1 = cos(0), col = 2, lty = 2)
segments(x0 = 1, y0 = 0, x1 = 1, y1 = 1+cos(1), col = 2, lty = 2)
abline(h=0)
```

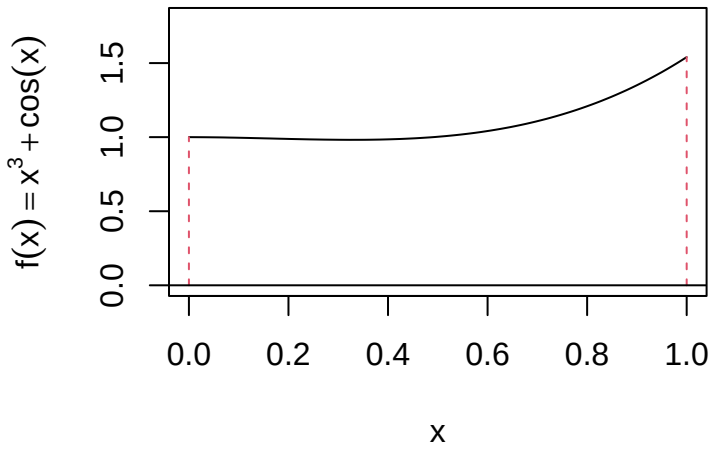

Figura 2.1: Função integrada no item a)

```
## Integração Monte Carlo
u <- runif(10000, 0, 1)
mean(u^3+cos(u))
```

```
## [1] 1.092608
```

```
## Integração numérica
integrate(f = function(x){x^3 + cos(x)}, lower = 0, upper = 1)
```

```
## 1.091471 with absolute error < 1.2e-14
```

```
## Resultado analitico
(1^4/4-0^4/4) + (sin(1)-sin(0))
```

```
## [1] 1.091471
```

b) Neste caso,

$$\int_1^3 x^2 + 3x^3 dx = \int_0^1 h(y)dy,$$

em que $h(y) = 2 \times [(1+2y)^2 + 3(1+2y)^3]$.

Isto é, $\int_1^3 x^2 + 3x^3 dx = \int_0^1 2[(1+2y)^2 + 3(1+2y)^3]dy$, que pode ser visto como $E[2(1+2U)^2 + 6(1+2U)^3]$, em que $U \sim Uniforme(0,1)$.

No R temos:

```
x <- seq(1, 3, length.out = 100)

par(mar = c(4,4,1,1))
plot(x, x^2 + 3*x^3, type = "l", ylim = c(0, 100),
     ylab = bquote(f(x)==x^2+3*x^3))
segments(x0 = 1, y0 = 0, x1 = 1, y1 = 4, col = 2, lty = 2)
segments(x0 = 3, y0 = 0, x1 = 3, y1 = 3^2 + 3*3^3, col = 2, lty = 2)
abline(h=0)
```

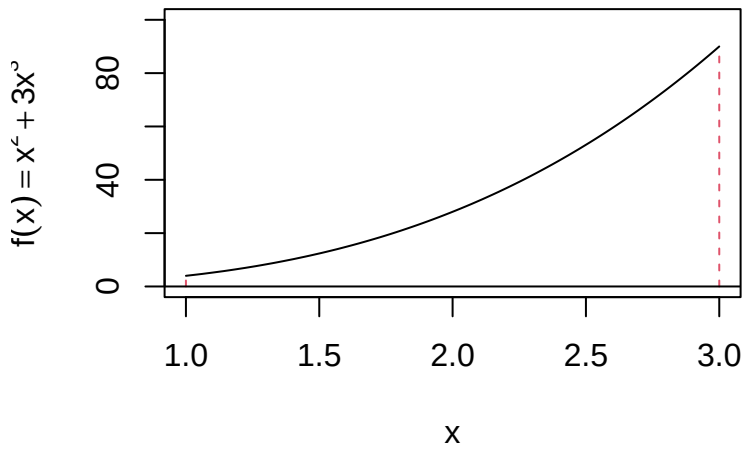

Figura 2.2: Função integrada no item b)

```
## Integração Monte Carlo
u <- runif(100000, 0, 1)
mean(2*(1+2*u)^2 + 6*(1 + 2*u)^3)
```

```
## [1] 68.7663
```

```r
## Integração numérica
integrate(f = function(x){x^2 + 3*x^3}, lower = 1, upper = 3)
```

```
## 68.66667 with absolute error < 7.6e-13
```

```r
## Resultado analitico
(3^3/3-1^3/3)+3*(3^4/4-1^4/4)
```

```
## [1] 68.66667
```

Exemplo 2.2. Como podemos calcular aproximadamente a área de um círculo com raio igual a 1 e centro na origem?

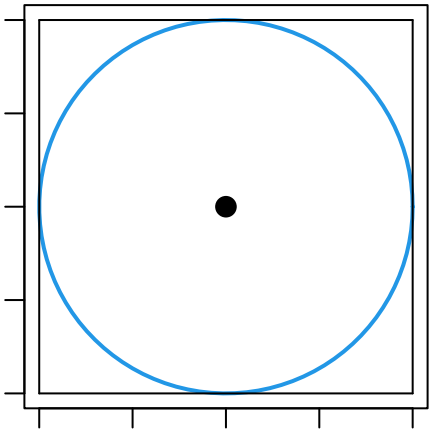

Figura 2.3: Círculo dentro do quadrado

Esta área é dada por

$$A = \int_{x^2+y^2 \leq 1} dxdy = \int_{-1}^{1}\int_{-1}^{1} I\{x^2 + y^2 \leq 1\}dxdy,$$

em que $I\{x^2 + y^2 \leq 1\}$ denota uma função indicadora tal que

$$I\{x^2 + y^2 \leq 1\} = \begin{cases} 1 & \text{se } x^2 + y^2 \leq 1 \\ 0 & \text{caso contrário.} \end{cases}$$

Note que

$$\begin{aligned} A &= 4\int_{-1}^{1}\int_{-1}^{1} I\{x^2+y^2\leq 1\}\frac{1}{4}dxdy \\ &= 4\left[\sum_{i=1}^{k} I\{x_i^2+y_i^2\leq 1\}/k\right], \end{aligned}$$

em que x_i e y_i são amostras iid de Uniforme(-1, 1), $i = 1, \ldots, k$.

No R, temos:

```
k <- 1000 # numero de replicas Monte Carlo
x <- runif(k, -1, 1); y <- runif(k, -1, 1)
A <- 4*sum( x^2 + y^2 <= 1 )/k

# Estimativa obtida
A
```

```
## [1] 3.144
```

```
# Valor verdadeiro
pi
```

```
## [1] 3.141593
```

Se Repetirmos o processo teremos o mesmo resultado sempre?

```
### Estimando a area do circulo - pi
set.seed(12456)
f1 <- function(k){
  x <- runif(k, -1, 1);   y <- runif(k, -1, 1)
  A <- 4*sum( x^2 + y^2 <= 1 )/k
}

ks = rep(1000, 300);
As = sapply(X = ks, f1)

par(mar = c(4,4,1,1))
hist(As, main = "", xlab = "Estimativas", col = "gray", ylab = "Frequência")
```

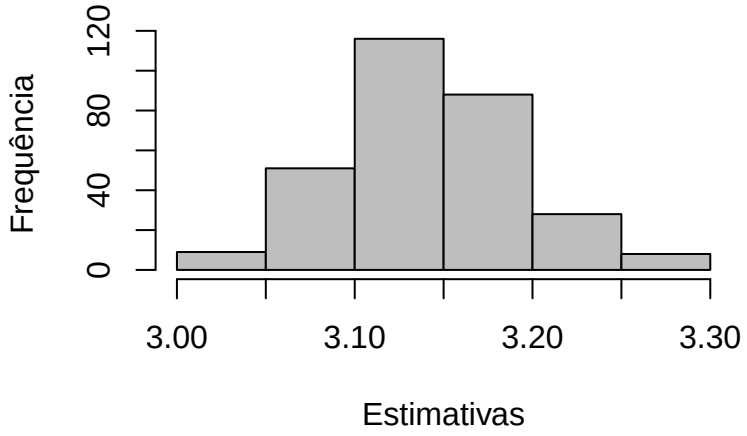

Figura 2.4: Estimativas obtidas para a área com k=1000

Qual o tamanho amostral necessário, isto é, quantas réplicas Monte Carlo necessitamos para ter uma boa aproximação?

Abaixo apresentamos o comportamento das estimativas ao variarmos o número k de réplicas de Monte Carlo.

```
### Estimando a area do circulo - pi
set.seed(12456)
ks = 10^c(3:7);  As = sapply(X = ks, f1)

par(mar = c(4, 4, 1, 1))
plot(log10(ks), As, type = "b", ylab = "Estimativas",
     xlab = "log10(k)", pch = 16, cex = 1.5)
abline(h = pi, lwd = 2, lty = 2, col = 2)
```

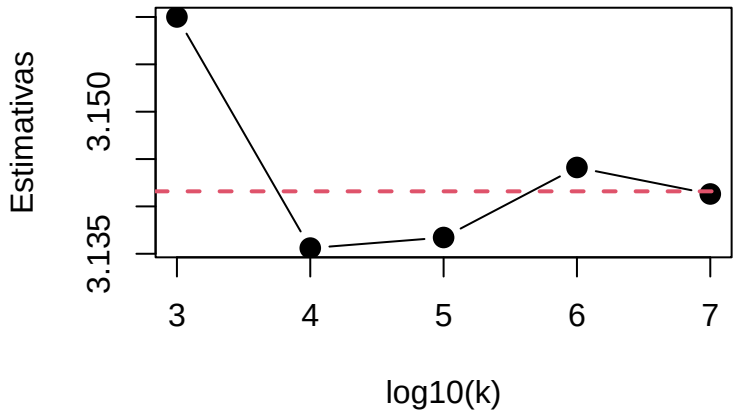

Figura 2.5: Estimando a area do circulo - pi

Como o método de integração de Monte Carlo pode ser utilizado na Inferência Estatística?

Métodos ou estudos Monte Carlo podem ser aplicados em muitas situações na área de Inferência Estatística. Entre elas:

- Estimar viés e erro quadrático médio (EQM) de um estimador;
- Calcular percentis e outras quantidades de interesse de uma distribuição amostral, por exemplo, o p-valor em um teste de hipótese;
- Estimar a probabilidade de cobertura de intervalos de confiança;
- Estimar a função poder em um procedimento de teste de hipóteses;
- Obter a função de verossimilhança marginal a partir de uma função de verossimilhança conjunta.

2.1.1 Exercícios

Exercício 2.1. (Exercícios 3 a 9 do Capítulo 3 do Ross (2013)) Utilizando o método de integração de Monte Carlo, obtenha uma aproximação para as seguintes integrais.

a. $\int_0^1 \exp\{e^x\}dx$
b. $\int_{-2}^2 \exp\{x + x^2\}dx$
c. $\int_0^\infty x(1+x^2)^{-2}dx$
d. $\int_{-\infty}^\infty e^{-x^2}dx$
e. $\int_0^1\int_0^1 \exp\{(x+y)^2\}dydx$
f. $\int_0^\infty\int_0^x \exp\{-(x+y)\}dydx$

Dica: Use a função indicadora $I(y < x) = 1$ se $y < x$ e 0 caso contrário e reescreva a integral com ambos os termos indo de 0 a ∞.

2.2 Métodos Monte Carlo para Estimação

Suponha $\mathbf{X} = (X_1, \dots, X_n)$ um vetor aleatório com fdp $f_{\mathbf{X}}$. Um estimador $\hat{\theta}$ para um parâmetro θ é uma função n variada

$$\hat{\theta} = \hat{\theta}(X_1, \dots, X_n)$$

da amostra. Note que funções do estimador $\hat{\theta}$ também são funções n variadas dos dados.

Seja $\mathbf{x} = (x_1, \dots, x_n)^T \in \mathbb{R}^n$ uma amostra observada e $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(m)}$ uma sequência de amostras aleatórias independentes geradas da distribuição de $\mathbf{X}$. Uma amostra aleatória de tamanho m da distribuição amostral de $\hat{\theta}$ pode obtida se gerarmos repetidamente

m amostras aleatórias $\mathbf{x}^{(j)}$, $j = 1, \ldots, m$, e, para cada amostra, calcularmos

$$\hat{\theta}^{(j)} = \hat{\theta}^{(j)}(\mathbf{x}^{(j)}) = \hat{\theta}^{(j)}(x_1^{(j)}, \ldots, x_n^{(j)}).$$

2.2.1 Estimação Monte Carlo e erro padrão

Para obter uma estimativa baseada em m réplicas de Monte Carlo para $\theta = \mathsf{E}[g(X_1, \ldots, X_n)]$ utilizamos os seguinte algoritmo.

Algoritmo

Passo 1: Amostrar independentemente $\mathbf{x}^{(j)} = (x_1^{(j)}, \ldots, x_n^{(j)})^T$ do vetor aleatório $\mathbf{X}^{(j)}$ com densidade $f_{\mathbf{X}}(\mathbf{x})$, para $j = 1, \ldots, m$;
Passo 2: Obter $g(\mathbf{x}^{(1)}), \ldots, g(\mathbf{x}^{(m)})$;
Passo 3: Calcular

$$\hat{\theta} = \frac{1}{m}\sum_{j=1}^{m} g(\mathbf{x}^{(j)}) = \overline{g(\mathbf{x})}.$$

Note que em situações onde o parâmetro θ é a esperança de uma função $g(\mathbf{x})$ podemos estimar a partir da média de m amostras independentes. Esse resultado é baseado no método de integração de Monte Carlo e a convergência é garantida pela Lei Forte dos Grandes Números.

Erro Monte Carlo

Como as observações $\mathbf{x}^{(j)}$ são amostradas independentemente para $j = 1, \ldots, m$, o erro padrão de $\widehat{\theta}$ é dado por

$$\begin{aligned} se(\hat{\theta}) &= \sqrt{\mathsf{Var}\left[\overline{g(\mathbf{X})}\right]} \\ &= \sqrt{\frac{1}{m^2}\sum_{j=1}^{m} \mathsf{Var}\left[g(\mathbf{X}^{(j)})\right]} \\ &= \sqrt{\frac{1}{m}\mathsf{Var}\left[g(\mathbf{X})\right]}. \end{aligned}$$

Quando a distribuição de $g(\mathbf{X})$ é desconhecida, podemos substituir F pela distribuição empírica F_n da amostra $g(\mathbf{x}^{(1)}), \ldots, g(\mathbf{x}^{(n)})$ e a estimativa "plug-in" da variância de $g(\mathbf{X})$ é

$$\widehat{\text{Var}[g(\mathbf{X})]} = \frac{1}{m}\sum_{j=1}^{m}\left(g(\mathbf{x}^{(j)}) - \overline{g(\mathbf{x})}\right)^2.$$

Note que esta é a variância populacional de uma pseudo população finita com função de distribuição acumulada (fda) F_n. A estimativa correspondente do erro padrão de $\hat{\theta}$ é

$$\begin{aligned}\hat{se}(\hat{\theta}) &= \frac{1}{\sqrt{m}}\left[\frac{1}{m}\sum_{j=1}^{m}\left(g(\mathbf{x}^{(j)}) - \overline{g(\mathbf{x})}\right)^2\right]^{1/2} \\ &= \frac{1}{m}\left[\sum_{j=1}^{m}\left(g(\mathbf{x}^{(j)}) - \overline{g(\mathbf{x})}\right)^2\right]^{1/2}.\end{aligned}$$

Se usarmos um estimador não viciado de $\text{Var}[g(\mathbf{X})]$ temos que

$$\hat{se}(\hat{\theta}) = \frac{1}{\sqrt{m}}\left[\frac{1}{m-1}\sum_{j=1}^{m}\left(g(\mathbf{x}^{(j)}) - \overline{g(\mathbf{x})}\right)^2\right]^{1/2}.$$

Obs: Em um experimento Monte Carlo, como m é grande, as duas estimativas são aproximadamente iguais.

Exemplo 2.3. Suponha que X_1 e X_2 são iid de uma distribuição Normal padrão e desejamos estimar $E[|X_1 - X_2|]$. Para isto, basta seguir o seguinte algoritmo.

Algoritmo

Passo 1: Gerar, independentemente, amostras aleatórias $\mathbf{x}^{(j)} = (x_1^{(j)}, x_2^{(j)})$ da distribuição normal, $j = 1, \ldots, m$;
Passo 2: Calcular $g(\mathbf{x}^{(j)}) = |x_1^{(j)} - x_2^{(j)}|$;
Passo 3: Calcular $\hat{\theta} = \frac{1}{m}\sum_{j=1}^{m} g(\mathbf{x}^{(j)})$.

No R temos:

```
set.seed(15568)
m <- 1000; g <- numeric(m)
for (i in 1:m) {
  x <- rnorm(2);   g[i] <- abs(x[1] - x[2])
}
est <- mean(g); est  # estimativa
```

```
## [1] 1.112816
```

```
sqrt(sum((g - est)^2)) / m  # erro padrao
```

```
## [1] 0.02687733
```

```
## Estudo Monte Carlo do estimador Monte Carlo da esperança
K <- 10000; m <- 1000;  Est <- numeric(K); g <- numeric(m)
for(k in 1:K){
  for (j in 1:m) {
    x <- rnorm(2);  g[j] <- abs(x[1] - x[2])
  }
  Est[k] <- mean(g)
}
sd(Est)
```

```
## [1] 0.02684063
```

```
par(mar = c(4, 4, 1, 1))
hist(Est, prob = T, main = "", xlab = expression( hat(theta) ), ylab = "Densidade")
```

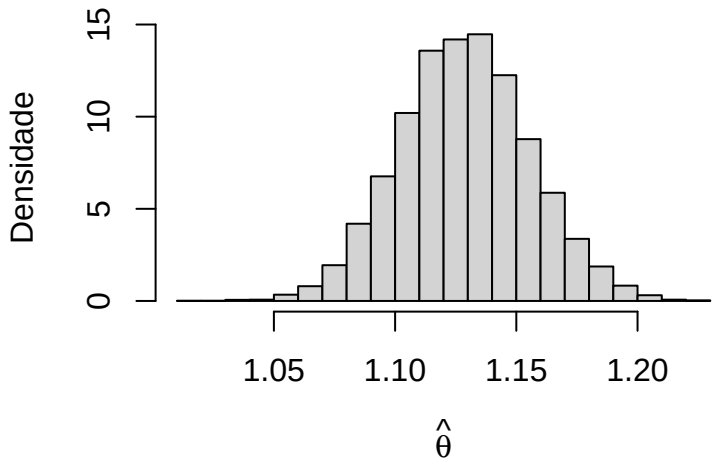

Figura 2.6: Distribuição do estimador

Um método para determinar quando parar a geração de novos dados

Para determinar quando parar a geração de novos valores podemos adotar a estratégia descrita no seguinte algoritmo.

Algoritmo

Passo 1: Escolher um valor d aceitável para o desvio-padrão de um estimador;
Passo 2: Gere pelo menos 100 réplicas;
Passo 3: Continue a gerar valores adicionais, pare quando tiver gerado k valores e $S_k/\sqrt{k} \leq d$, em que S_k é o desvio padrão amostral dos k valores gerados de $g(\mathbf{x})$;
Passo 4: O estimador de θ é dado por $\widehat{\theta} = \overline{g(\mathbf{x})}$.

2.2.2 Estimação de Viés e EQM

Os métodos Monte Carlo podem ser aplicados para estimar o viés e o Erro Quadrático Médio (EQM) de um estimador, definidos por

$$\text{Viés}(\widehat{\theta}) = \text{E}[\widehat{\theta} - \theta] \quad \text{e} \quad \text{EQM}(\widehat{\theta}) = \text{E}[(\widehat{\theta} - \theta)^2].$$

Note que temos, respectivamente, $g_1(\mathbf{X}) = \widehat{\theta}(\mathbf{X}) - \theta$ e $g_2(\mathbf{X}) = \left(\widehat{\theta}(\mathbf{X}) - \theta\right)^2$. Se m pseudo amostras aleatórias $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}$ são geradas da distribuição de $\mathbf{X}$, então estimativas Monte Carlo do viés e do EQM de $\widehat{\theta} = \widehat{\theta}(x_1, \ldots, x_n)$ são

$$\begin{aligned} \widehat{Vis} &= \frac{1}{m}\sum_{j=1}^{m}(\widehat{\theta}^{(j)} - \theta), \\ \widehat{EQM} &= \frac{1}{m}\sum_{j=1}^{m}(\widehat{\theta}^{(j)} - \theta)^2, \end{aligned}$$

em que $\widehat{\theta}^{(j)} = \widehat{\theta}(\mathbf{x}^{(j)}) = \widehat{\theta}(x_1^{(j)}, \ldots, x_n^{(j)})$.

Exemplo 2.4. (EQM da média aparada) Neste exemplo, calculamos o EQM da média aparada que é algumas vezes aplicada para estimar o centro de uma distribuição simétrica contínua.

Suponha variáveis aleatórias $X_1, \dots, X_n$ e $x_{(1)}, \dots, x_{(n)}$ a correspondente amostra ordenada. A média aparada de nível k é definida por

$$\overline{X}_{[-k]} = \frac{1}{n-2k} \sum_{i=k+1}^{n-k} X_{(i)}.$$

Vamos obter uma estimativa de Monte Carlo do $EQM(\overline{X}_{[-k]})$ da média aparada do primeiro nível supondo que a distribuição da amostra é normal padrão. Neste exemplo, o centro da distribuição é 0 e o parâmetro alvo é $\theta = \mathsf{E}[X] = \mathsf{E}[\overline{X}_{[-k]}] = 0$.

Denotamos o primeiro nível da média amostral aparada por T. Uma estimativa Monte Carlo de $EQM(T)$ baseada em m réplicas pode ser obtida como segue.

Algoritmo

Passo 1: Gere as réplicas $T^{(j)}$, $j = 1, \dots, m$, repetindo:

1. Gerar $x^{(j)} \overset{iid}{\sim} N(0,1)$;
2. Ordenar a amostra gerada para obter $x^{(j)}_{(1)} \leq \dots \leq x^{(j)}_{(n)}$;
3. Calcular $T^{(j)} = \frac{1}{n-2} \sum_{i=2}^{n-1} x^{(j)}_{(i)}$;

Passo 2: Calcule

$$\widehat{EQM} = \frac{1}{m} \sum_{j=1}^{m} (T^{(j)} - \theta)^2 = \frac{1}{m} \sum_{j=1}^{m} (T^{(j)})^2.$$

No R:

```
### Estimacao do EQM: Media aparada
n <- 20; m <- 1000
tmean <- replicate(n = m, expr = {x <- sort(rnorm(n)); sum(x[2:(n-1)]) / (n-2)})
mse <- mean(tmean^2)
mse
```

```
## [1] 0.05199946
```

```
sqrt(sum((tmean - mean(tmean))^2)) / m #se
```

```
## [1] 0.007207448
```

```r
## Considerando a mediana
n <- 20
m <- 1000
tmean <- replicate(n = m, expr = {x <- sort(rnorm(n)); median(x)})
mse <- mean(tmean^2)
mse
```

```
## [1] 0.07717845
```

```r
sqrt(sum((tmean - mean(tmean))^2)) / m #se
```

```
## [1] 0.008785124
```

Exemplo 2.5. Comparando o EQM da média aparada de nível k para a normal padrão e uma distribuição normal contaminada. A distribuição normal contaminada considerada é a mistura

$$p \times N(0, \sigma^2 = 1) + (1 - p) \times N(0, \sigma^2 = 100)$$

e parâmetro alvo é a média $\theta = 0$.

Fazemos no R:

```r
n <- 20; K <- n/2 - 1; m <- 1000; mse <- matrix(0, 6, 6)

trimmed.mse <- function(n, m, k, p) {
  tmean <- numeric(m)
  for (i in 1:m) {
    sigma <- sample(c(1, 10), size = n, replace = TRUE, prob = c(p, 1-p))
    x <- sort(rnorm(n, 0, sigma))
    tmean[i] <- sum(x[(k+1):(n-k)]) / (n-2*k)
  }
  mse.est <- mean(tmean^2)
  se.mse <- sqrt(mean((tmean-mean(tmean))^2)) / sqrt(m)
  return(c(mse.est, se.mse))
}

for (k in 0:5) {
  mse[k+1, 1:2] <- trimmed.mse(n=n, m=m, k=k, p=1.0)
```

```
  mse[k+1, 3:4] <- trimmed.mse(n=n, m=m, k=k, p=.95)
  mse[k+1, 5:6] <- trimmed.mse(n=n, m=m, k=k, p=.9)
}
colnames(mse) <- c("p=1.0", "p=1.0", "p=0.95", "p=0.95", "p=0.90", "p=0.95")
rownames(mse) <- c("k=0", "k=1", "k=2", "k=3", "k=4", "k=5")
round(mse, 4) # EQM para nível k
```

```
##      p=1.0  p=1.0 p=0.95 p=0.95 p=0.90 p=0.95
## k=0 0.0481 0.0069 0.2838 0.0168 0.5213 0.0228
## k=1 0.0514 0.0072 0.0949 0.0097 0.1748 0.0132
## k=2 0.0535 0.0073 0.0693 0.0083 0.0967 0.0098
## k=3 0.0526 0.0073 0.0638 0.0080 0.0782 0.0088
## k=4 0.0622 0.0079 0.0699 0.0084 0.0722 0.0085
## k=5 0.0599 0.0077 0.0680 0.0082 0.0757 0.0087
```

2.2.3 Estimação o nível de confiança

Algumas vezes um intervalo de confiança pode ser construído assumindo algumas suposições para ter nível de confiança γ, mas podemos estar interessados em saber qual o nível de confiança atingido quando estas suposições não são válidas.

Se (U, V) é um intervalo de confiança para um parâmetro desconhecido θ, então U e V são estatísticas com distribuições que dependem da distribuição F_X da população amostrada **X**. O nível de confiança é a probabilidade de que o intervalo (U, V) contenha o verdadeiro valor do parâmetro θ. Isto é,

$$\begin{aligned} P(U \leq \theta \leq V) &= \int I\{\theta \in (U,V)\} f_{U,V}(u,v)dudv \\ &= \mathsf{E}\left[I\{\theta \in (U,V)\}\right], \end{aligned}$$

em que $I\{\theta \in (U,V)\} = \begin{cases} 1, & \text{se } \theta \in (U,V), \\ 0, & \text{caso contrário.} \end{cases}$

Então, podemos obter o nível de confiança empírico através de

Algoritmo

Passo 1: Para cada réplica, indexada por $j = 1, \dots, m$:

1. Gere a j-ésima amostra aleatória, $x_1^{(j)}, \dots, x_n^{(j)}$;
2. Calcule o intervalo de confiança C_j para a j-ésima amostra;
3. Calcule $y^{(j)} = I\{\theta \in C_j\}$;

Passo 2: Calcule o nível de confiança empírico $\overline{y} = \frac{1}{m}\sum_{j=1}^{m} y^{(j)}$.

O estimador $\overline{Y}$ é uma proporção amostral estimando o verdadeiro nível de confiança $1-\alpha$, então Var$[\overline{Y}] = (1-\alpha)\alpha/m$ e uma estimativa para o erro padrão é dada por

$$\hat{se}(\overline{Y}) = \sqrt{(1-\overline{y})\overline{y}/m}.$$

Exemplo 2.6. Seja $x_1, \dots, x_n$ uma amostra aleatória da distribuição Normal(μ, σ^2), $n \geq 2$, e S^2 a variância amostral. Então,

$$\chi^2 = \frac{(n-1)S^2}{\sigma^2} \sim \chi^2_{(n-1)}.$$

Um intervalo de $100(1-\alpha)\%$ de confiança unilateral para σ^2 é dado por

$$IC_{100(1-\alpha)\%}(\sigma^2) = [0,\ (n-1)s^2/\chi^2_\alpha],$$

em que χ^2_α é quantil de ordem α da distribuição $\chi^2_{(n-1)}$.

Se a população amostrada é normal, então o nível de confiança é $1-\alpha$. Mas o que acontece se os dados não são normalmente distribuídos?

```
# Dados com distribuicao normal com var=4
n <- 20; alpha <- .05
UCL <- replicate(1000, expr = {
  x <- rnorm(n, mean = 0, sd = 2)
  (n-1) * var(x) / qchisq(alpha, df = n-1) # ultimo calculo é retornado
} )
mean(UCL > 4)
```

```
## [1] 0.949
```

```
## Se os dados tem distribuicao qui-quadrado com var=4
UCL <- replicate(1000, expr = {
  x <- rchisq(n, df = 2)
  (n-1) * var(x) / qchisq(alpha, df = n-1)
} )
mean(UCL > 4)
```

```
## [1] 0.768
```

E como fica para uma t-student para diferentes valores do grau de liberdade mas com variância igual a 4?

```
n <- 30; m <- 10000; alpha <- .05
gl <- seq(3, 50, by = 2); conf <- numeric( length(gl) )
for( l in 1:length(gl) ){
  UCL <- replicate(m, expr = {
    x <- rt(n, df = gl[l])*sqrt( (gl[l]-2)/gl[l] )*2
    (n-1) * var(x) / qchisq(alpha, df = n-1)
  } )
  conf[l] <- mean(UCL > 4)
}
par(mar = c(4,4,1,1))
plot(gl, conf, type = "b", pch = 16, cex = 0.75)
abline(h = 0.95, col = 2, lty = 2, lwd = 2)
```

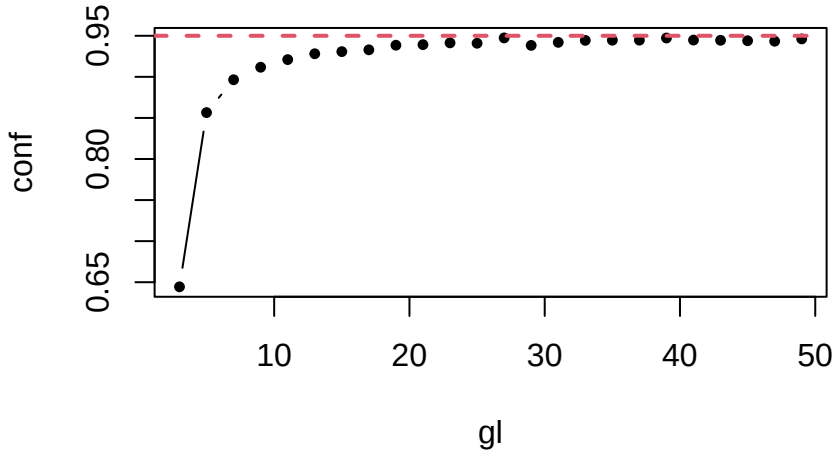

Figura 2.7: Probabilidade de cobertura para diferentes valores do grau de liberdade da distribuição t-Student

E como fica para uma t-student para diferentes tamanhos amostrais?

```
## Se os dados tem distribuicao t-student(0,sigma2,5) com var=4, mudando n
n <- seq(10, 600, by = 30)
m <- 10000; alpha <- .05; gl <- 5
conf <- numeric( length(n) )

for(l in 1:length(n)){
  UCL <- replicate(m, expr = {
    x <- rt(n[l], df = gl)*sqrt(4*(gl-2)/gl)
    (n[l]-1) * var(x) / qchisq(alpha, df = n[l]-1)
  }, simplify = T )
  conf[l] <- mean(UCL > 4)
}

par(mar = c(4,4,1,1))
plot(n, conf, type = "b", pch = 16, cex = 0.75, ylim = c(0.45, 1) )
abline(h = 0.95, col = 2, lty = 2, lwd = 2)
```

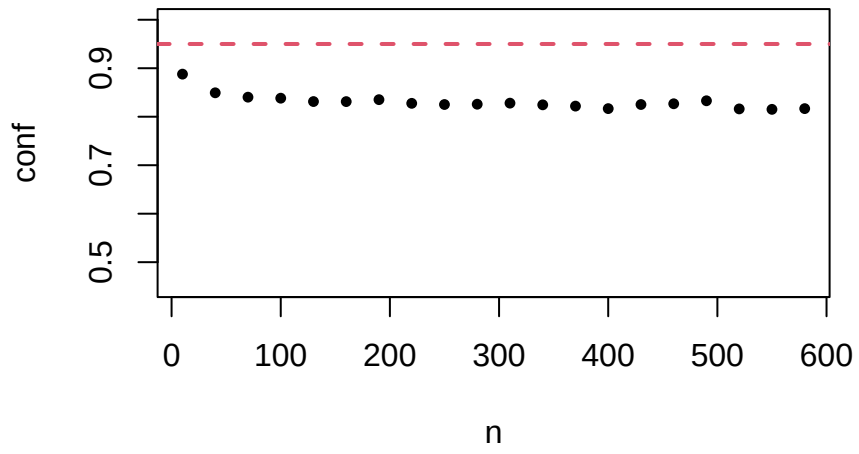

Figura 2.8: Probabilidade de cobertura para diferentes valores de tamanho amostral com distribuição t-Student

2.2.4 Exercícios

Exercício 2.2. Faça um estudo Monte Carlo para estimar o viés e o EQM dos seguintes estimadores para a variância populacional:

$$\hat{\sigma}_1^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 \quad \text{e} \quad \hat{\sigma}_2^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2.$$

Gere os dados de diferentes distribuições e considere diferentes tamanhos amostrais. Utilize $m = 1000$ réplicas de Monte Carlo. Estime o erro padrão destas estimativas.

Exercício 2.3. Uma seguradora de acidentes tem 1000 segurados, cada um dos quais apresentará um sinistro independentemente em um mês com probabilidade de 0,05. Suponha que os valores das reivindicações feitas sejam variáveis aleatórias exponenciais independentes com média de $800. Usando o método da tranformação, implemente uma função para amostrar n valores do total gasto com sinistros pela seguradora. O valor n representaria, por exemplo, n meses ou n replicações do valor de um único mês. Usando o Método de Monte Carlo, obtenha uma estimativa para o valor esperado do gasto total mensal. Estime também o erro padrão desta estimativa.

Exercício 2.4. Suponha o intervalo de 95% de confiança t-Student para a média populacional com dados não normais. A probabilidade de que o intervalo de confiança cubra a verdadeira média não é necessariamente 0,95, pois uma suposição não é atendida. Use um estudo Monte Carlo para estimar a probabilidade de cobertura (e o erro padrão desta estimativa) do intervalo t-Student para amostras de tamanhos n =20, 30 e 100 das seguintes distribuições:

a. χ^2_2
b. Outra distribuição assimétrica de sua preferência.

2.3 Métodos Monte Carlo para testes de hipóteses

Suponha que desejamos testar uma hipótese relacionada a um parâmetro θ que está em um espaço paramétrico Θ. As hipóteses de interesse são

$$H_0 : \theta \in \Theta_0 \text{ vs } H_1 : \theta \in \Theta_1,$$

em que Θ_0 e Θ_1 formam uma partição do espaço paramétrico.

O nível de significância de um teste, denotado por α, é o limite superior da probabilidade do erro tipo I. A probabilidade de rejeitar a hipótese nula depende do verdadeiro valor de θ. Para um procedimento de teste, se $\pi(\theta)$ denota a probabilidade de rejeitar H_0, então

$$\alpha = \sup_{\theta \in \Theta_0} \pi(\theta).$$

Taxa empírica do erro tipo I

O experimento é replicado um grande número de vezes sob as condições da hipótese nula e taxa empírica do erro tipo I é a proporção de valores significantes da estatística de teste entre as replicações.

Assim, temos o seguinte algoritmo:

Algoritmo

Passo 1: Para cada réplica, indexada por $j = 1, \dots, m$:

1. Gere a j-ésima amostra aleatória, $x_1^{(j)}, \dots, x_n^{(j)}$ da distribuição sob a hipótese nula, com $\theta' \in \Theta_0$;

2. Calcule a estatística de teste T_j para a amostra gerada;
   1.Faça $I_j = 1$ se H_0 é rejeitada a um nível de significância α e $I_j = 0$ caso contrário;

Passo 2: Calcule a proporção de testes significantes por $\hat{\pi}(\theta') = \frac{1}{m}\sum_{j=1}^{m} I_j$.

Uma estimativa para o erro padrão de $\hat{\pi}(\theta')$ é dada por

$$\hat{se}(\hat{\pi}(\theta')) = \sqrt{\frac{\hat{\pi}(\theta')(1-\hat{\pi}(\theta'))}{m}} \leq \frac{0,5}{\sqrt{m}}.$$

Poder de um teste

O poder de um teste é dado pela função poder $\pi : \Theta \to [0,1]$, que é a probabilidade $\pi(\theta)$ de rejeitar H_0 dado que o verdadeiro valor do parâmetro é θ. Assim, para um dado $\theta_1 \in \Theta_1$, a probabilidade do erro tipo II é $1 - \pi(\theta_1)$. Se a função poder de um teste não pode ser derivada analiticamente, o poder de um teste para $\theta_1 \in \Theta_1$ fixo pode ser estimado via métodos Monte Carlo. Note que a função poder é definida para todo $\theta \in \Theta$, mas o nível de significância α controla que $\pi(\theta) \leq \alpha$ para todo $\theta_0 \in \Theta_0$.

Obtemos o algoritmo abaixo:

Algoritmo

Passo 1: Escolha um valor $\theta_1 \in \Theta$ do parâmetro;

Passo 2: Para cada réplica, indexada por $j = 1, \ldots, m$:

1. Gere a j-ésima amostra aleatória, $x_1^{(j)}, \ldots, x_n^{(j)}$ tal que $\theta = \theta_1$;
2. Calcule a estatística de teste T_j para a amostra gerada;
3. Faça $I_j = 1$ se H_0 é rejeitada a um nível de significância α e $I_j = 0$ caso contrário;

Passo 3: Calcule a proporção de testes significantes por $\hat{\pi}(\theta_1) = \frac{1}{m}\sum_{j=1}^{m} I_j$.

Exemplo 2.7. Suponha que $X_1, \ldots, X_n$ seguem distribuição $N(\mu, \sigma^2)$. Deseja-se testar, com nível de significância $\alpha = 0,05$,

$$H_0 : \mu = 500 \text{ vs } H_1 : \mu > 500.$$

Sob a hipótese nula,

$$T = \frac{\overline{X} - 500}{S/\sqrt{n}} \sim T_{n-1},$$

em que T_{n-1} denota a distribuição t-Student com $n - 1$ graus de liberdade.

a. Use o método de Monte Carlo para calcular a probabilidade do erro tipo I quando $\sigma = 100$;
b. Use simulação para estimar o poder e plotar a curva de poder empírico.

No R podemos fazer:

```
## Estimando nível de significância
n <- 20; alpha <- .05; mu0 <- 500; sigma <- 100; m <- 10000; p <- numeric(m)
for (j in 1:m) {
  x <- rnorm(n, mu0, sigma)
  ttest <- t.test(x, alternative = "greater", mu = mu0)
  p[j] <- ttest$p.value
```

```
}
p.hat <- mean(p < alpha);  se.hat <- sqrt(p.hat * (1 - p.hat) / m)
print(c(p.hat, se.hat))
```

```
## [1] 0.04900000 0.00215868
```

```
## Erro tipo I para diferentes valores de mu para o teste mu<=500
mu0s <- seq(450, 500, by=1)
p <- numeric(m)
p.hat <- numeric(length(mu0s))
for(l in 1:length(mu0s)){
  for (j in 1:m) {
    x <- rnorm(n, mu0s[l], sigma)
    ttest <- t.test(x, alternative = "greater", mu = mu0)
    p[j] <- ttest$p.value
  }
  p.hat[l] <- mean(p < alpha)
}

par(mar = c(4,4,1,1))
plot(mu0s, p.hat, type = "b", pch = 16, cex = 0.5, ylim = c(0,0.06),
     xlab = expression(mu), ylab = "Prop. de Rejeição")
abline(h = 0.05, col = 2, lty = 2)
```

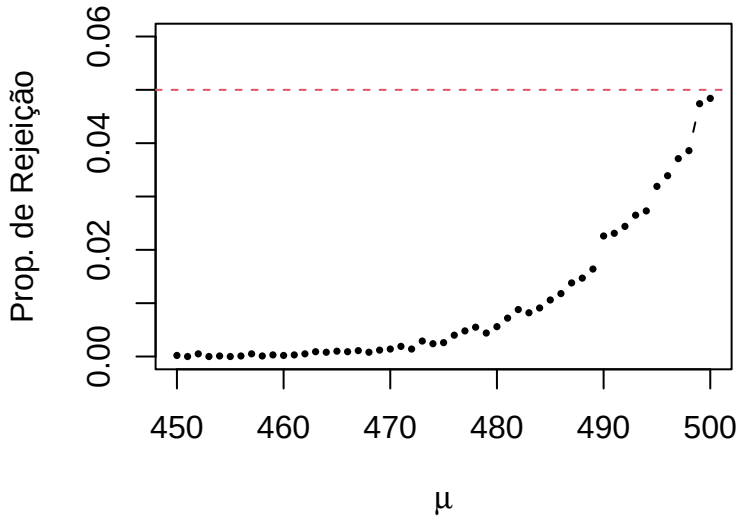

Figura 2.9: Estimativa do erro tipo I

```r
##  Funcao poder
n <- 20; m <- 1000; mu0 <- 500; sigma <- 100
mu <- c(seq(450, 650, 10))
k <- length(mu); power <- numeric(k)
for (i in 1:k) {
  mu_real <- mu[i]
  pvalues <- replicate(m, expr = {
    x <- rnorm(n, mean = mu_real, sd = sigma)
    ttest <- t.test(x, alternative = "greater", mu = mu0)
    ttest$p.value } )
  power[i] <- mean(pvalues < .05)
}

par(mar = c(4,4,1,1))
plot(mu, power, xlab = bquote(mu), ylab = "Poder", type = "b", pch = 16)
abline(v = mu0, lty = 2, col = 2)
abline(h = .05, lty = 2, col = 4)
```

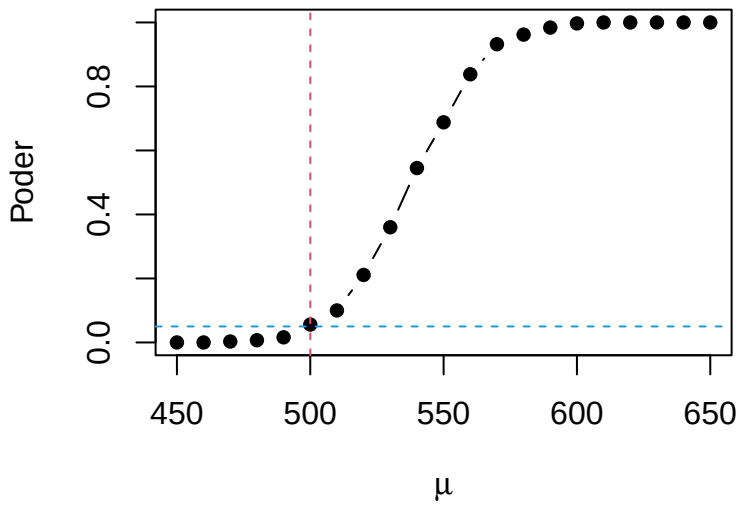

Figura 2.10: Estimativa do função poder

2.3.1 Exercícios

Exercício 2.5. Plote a curva de poder empírica para o teste t no Exemplo 3.7, mudando as hipóteses para $H_0 : \mu = 500$ vs $H_1 : \mu \neq 500$. e mantendo o nível de significância de $\alpha = 0,05$. Em um mesmo gráfico, plote as curvas de poder para os tamanhos amostrais 10, 20, 30, 40 e 50, usando cores diferentes.

Exercício 2.6. Refaça o Exemplo 3.7 simulando dados das distribuições gamma e t-student. Defina os parâmetros para obter um valor de média que faça sentido com o exemplo. Plote a curva da função poder e calcule o valor empírico da probabilidade do erro tipo I, para diferentes tamanhos amostrais.

2.4 Teste Monte Carlo

Uma simulação Monte Carlo pode ser utilizada quando a distribuição da estatística de teste não é conhecida sob H_0. Neste caso, o valor crítico ou o p-valor é determinado usando a distribuição estimada da estatística de teste. Extrai-se amostras aleatórias a partir de pseudo-populações, calcula-se o valor da estatística de teste em cada replicação e usa-se esses valores para estimar a distribuição da estatística de teste ou, mais especificamente, para uma probabilidade representando o p-valor do teste. Para ilustrar consideraremos um teste Qui-quadrado de aderência de uma distribuição ao um conjunto de dados.

Teste Qui-quadrado de aderência para dados discretos

Suponha que n variáveis aleatórias independentes $Y_1, \ldots, Y_n$ assumindo os valores $1, 2, \ldots, k$ são observadas e que há o interesse em testar a hipótese que $\{p_i,\ i = 1, \ldots, k\}$ é a função de massa de probabilidade dessas variáveis. Isto é,

$$H_0 : P(Y = i) = p_i,\ \ i = 1, \ldots, k.$$

Seja N_i o número de Y_j's iguais a i. Sob H_0, $N_i \sim Binomial(n, p_i)$ e $E[N_i] = np_i$. Assim, temos que:

- A estatística de teste é dada por $T = \sum_{i=1}^{k} \frac{(N_i - np_i)^2}{np_i}$ e a hipótese nula é rejeitada quando T é grande;
- O p-valor é dado por p-valor $= P(T \geq t_{obs}|H_0)$.

Uma aproximação razoável desta probabilidade pode ser obtida utilizando o resultado clássico de que, para grandes valores de n, T tem aproximadamente uma distribuição Qui-quadrado com $k-1$ graus de liberdade quando H_0 é verdadeira. Desta forma,

$$\text{p-valor} \approx P(\chi^2_{k-1} \geq t|H_0),$$

em que χ^2_{k-1} é uma variável aleatória Qui-quadrado com $k-1$ graus de liberdade.

Exemplo 2.8. Considere que a variável aleatória possa assumir qualquer valor entre 1, 2, 3, 4 e 5, e suponha que desejamos testar se estes valores são igualmente prováveis de ocorrer. Se uma amostra de tamanho 50 produzir os seguintes resultados de N_i:

$$12 \;\; 5 \;\; 19 \;\; 7 \;\; 7$$

então

$$\begin{aligned} T &= \frac{4+25+81+9+9}{10} = 12,8 \text{ e} \\ \text{p-valor} &\approx P(\chi^2_4 > 12,8) = 0,0122. \end{aligned}$$

Para este resultado de p-valor a hipótese nula é rejeitada.

Mas se aproximação pela Qui-quadrado não for razoável?

Podemos simular da distribuição da estatística de teste e obter um estimador mais acurado para o p-valor, com o seguinte algoritmo.

Algoritmo

Para $l = 1, \ldots, m$:

Passo 1: Geramos n variáveis aleatórias independentes $Y_1^{(l)}, \ldots, Y_n^{(l)}$ tal que $P(Y_j^{(l)} = i) = p_i, i = 1, \ldots, k, \;\; j = 1, \ldots, n$;

Passo 2: Obtemos $N_i^{(l)} = \sum_{i=1}^{n} I\{Y_j^{(l)} = i\}$;

Passo 3: Calculamos $T^{(l)} = \sum_{i=1}^{k} \frac{(N_i^{(l)} - np_i)^2}{np_i}$;

Passo 4: Pela lei dos grandes números, p-valor $\approx \frac{\sum_{l=1}^{m} I\{T^{(l)} \geq t_{obs}\}}{m}$.

No R:

```
m <- 10000; n <- 50
Ni <- c(12, 5, 19, 7, 7); pr <- rep(0.2, 5)

## Teste qui-quadrado - distribuição categorica
chisq.test(Ni)
```

```
##
##  Chi-squared test for given probabilities
##
## data:  Ni
## X-squared = 12.8, df = 4, p-value = 0.0123
```

```
## Teste Monte Carlo
chisq.test(Ni, simulate.p.value = T, B = 10000)
```

```
##
##  Chi-squared test for given probabilities with simulated p-value (based
##  on 10000 replicates)
##
## data:  Ni
## X-squared = 12.8, df = NA, p-value = 0.0131
```

```
t_obs <- chisq.test(Ni)$statistic

## Refazendo manualmente
Ts <- replicate(n = m, expr = {Ns = as.vector(rmultinom(1, size = n, prob = pr))
chisq.test(Ns)$statistic
})
sum(Ts > t_obs)/m
```

```
## [1] 0.0125
```

E se a distribuição qui-quadrado não é uma boa aproximação?

Abaixo geramos histogramas da estatística de teste em três situações e comparamos com a distribuição Qui-quadrado com 4 graus de liberdade. Note que a aproximação não é muito boa quando o tamanho amostral é pequeno e, consequentemente, existem frequências esperadas menores do que 5.

```
set.seed(1234)
Ts10 <- replicate(n = 10000,
                  expr = {Ns = as.vector(rmultinom(1, size = 10, prob = pr))
                  chisq.test(Ns)$statistic })
Ts20 <- replicate(n = 10000,
                  expr = {Ns = as.vector(rmultinom(1, size = 20, prob = pr))
                  chisq.test(Ns)$statistic })
Ts30 <- replicate(n = 10000,
                  expr = {Ns = as.vector(rmultinom(1, size = 30, prob = pr))
                  chisq.test(Ns)$statistic })


grid <- seq(0, 20, 0.01)
dens_chi4 <- dchisq(grid, 4)

par(mfrow = c(1, 3), mar = c(4, 4, 1, 1))
hist(Ts10, prob = T, breaks = 25, main = expression(n == 10),
     ylab = "Densidade", xlab = "Estatística de teste", xlim = c(0, 20))
lines(grid, dens_chi4, col = 2)
hist(Ts20, prob = T, breaks = 25, main = expression(n == 20),
     ylab = "Densidade", xlab = "Estatística de teste", xlim = c(0, 20))
lines(grid, dens_chi4, col = 2)
hist(Ts30, prob = T, breaks = 25, main = expression(n == 30),
     ylab = "Densidade", xlab = "Estatística de teste", xlim = c(0, 20))
lines(grid, dens_chi4, col = 2)
```

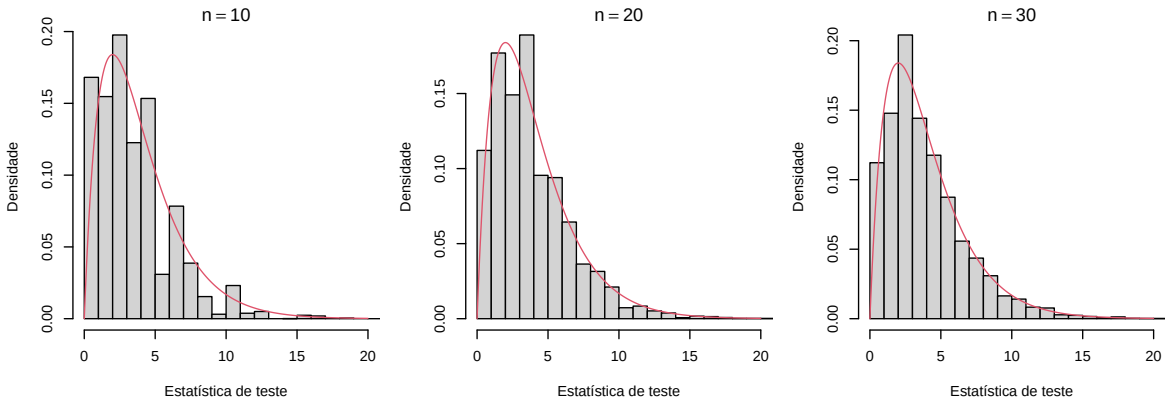

Figura 2.11: Comparação da distribuição simulada com a distribuição Qui-quadrado com 4 graus de liberdade

Exemplo 2.9. (Poisson) Teste se os dados referentes ao número de acidentes em um certo lugar durante 30 dias, apresentados na tabela de frequências abaixo, seguem a distribuição Poisson.

0	1	2	3	4	5	8
6	2	1	9	7	4	1

Para obter o valor observado da estatística de teste, definimos as k categorias a serem consideradas, estimamos $\widehat{\lambda}$, calculamos $\widehat{p}_i$ considerando o valor obtido para $\widehat{\lambda}$ e calculamos

$$T^{(l)} = \sum_{i=1}^{k} \frac{(N_i^{(l)} - n\widehat{p}_i^{(l)})^2}{n\widehat{p}_i^{(l)}}.$$

Podemos considerar o seguinte algoritmo:

Algoritmo

Para $l = 1, \dots, m$:

Passo 1: Geramos n variáveis aleatórias independentes $Y_1^{(l)}, \dots, Y_n^{(l)}$ sob a distribuição $Poisson(\widehat{\lambda})$;

Passo 2: Obtemos $N_i^{(l)} = \sum_{i=1}^{n} I\{Y_j^{(l)} = i\}$, para as categorias utilizadas;

Passo 3: Estimamos $\widehat{\lambda}^{(l)}$ e calculamos $\widehat{p}_i^{(l)}$ considerando o valor obtido para $\widehat{\lambda}^{(l)}$;

Passo 4: Calculamos $T^{(l)} = \sum_{i=1}^{k} \frac{(N_i^{(l)} - n\widehat{p}_i^{(l)})^2}{n\widehat{p}_i^{(l)}}$;

Passo 5: Calculamos p-valor $\approx \frac{\sum_{l=1}^{m} I\{T^{(l)} > t_{obs}\}}{m}$.

No R:

```
## Incluindo os dados
y <- c(rep(0, 6), rep(1, 2), rep(2, 1), rep(3, 9), rep(4, 7), rep(5, 4), 8)

par( mar = c(4, 4, 1, 1) )
barplot(table(y), xlab="Número de acidentes", ylab = "Frequência")
```

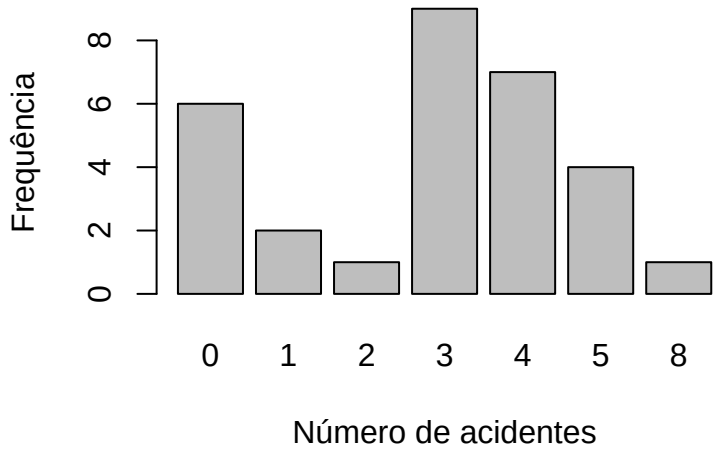

Figura 2.12: Gráfico de barras do número de acidentes nos últimos 30 dias

```
# Calculando lambda estimado para a amostra observada
(lambda_est <- mean(y))
```

```
## [1] 2.9
```

```
# Calculando probabilidades teóricas
(probs <- dpois(0:4, lambda = lambda_est))
```

```
## [1] 0.05502322 0.15956734 0.23137264 0.22366022 0.16215366
```

```
comp <- 1 - sum(probs) ## 5 ou mais acidentes

## Calculamos valor observado da estatística de teste
freqs <- c(6, 2, 1, 9, 7, 5)
t_obs <- chisq.test(x = freqs, p = c(probs, comp))$statistic

## Problema1: A funcao nao chisq.test considera que lambda foi estimado
##            e que o gl deve ser corrigido.
## Problema2: Calcula valor esperado sempre com o mesmo valor lambda estimado.
```

```r
##  Teste Monte Carlo correto (Ross)
m <- 10000
est <- numeric(m)
for(i in 1:m){
  y <- rpois(30, lambda_est)
  lambda_aux_i <- mean(y)
  probs <- dpois(0:4, lambda = lambda_aux_i)
  comp <- 1 - sum(probs)
  Ns <- as.vector(c(sum(y == 0), sum(y == 1), sum(y == 2),
                    sum(y == 3), sum(y == 4), sum(y >= 5)))
  est[i] <- chisq.test(x = Ns, p = c(probs, comp))$statistic
}
sum(est > t_obs)/m ## p-valor
```

```
## [1] 8e-04
```

2.4.1 Exercícios

Exercício 2.7. Implemente um algoritmo para fazer um teste de hipóteses Monte Carlo para testar se a média populacional é igual a um valor pré-especificado, assumindo que os dados são normalmente distribuidos com desvio padrão conhecido. Em um pequeno estudo Monte Carlo, compare o poder do teste implementado com o teste Normal (utilize a função implementada em algum pacote do R).

Dicas:

- No teste Monte carlo, utilize a estatística de teste $Z = (\bar{x} - \mu)/(\sigma/\sqrt{n})$, mas calcule o p-valor pelo procedimento Monte Carlo ao gerar amostras sob H_0 e estimar o p-valor através da proporção de amostras em que a estatística de teste apresenta valor maior em módulo do que o valor observado na amostra original.
- Será um algoritmo Monte Carlo dentro de um estudo Monte Carlo.

Capítulo 3

Bootstrap, Jackknife e testes de Permutação

Neste capítulo iremos apresentar o método de Bootstrap e o seu uso na Inferência Estatística. O conteúdo apresentado é baseado nos Capítulos 7 e 8 do livro de Rizzo (2008) e nos Capítulos de Efron and Tibshirani (1994). Mais sobre o assunto também pode ser visto no Capítulo 9 de Givens and Hoeting (2012).

3.1 Introdução

Os métodos bootstrap são uma classe métodos Monte Carlo que estimam a distribuição de uma população por reamostragem e são frequentemente utilizados quando a distribuição da população alvo não é especificada e a amostra é a única informação disponível. O bootstrap foi introduzido por Efron em 1979 e ocorreram mais desenvolvimentos nos anos seguintes. A seguir ressaltamos alguns aspectos relativos ao bootstrap:

- Efron deu o nome bootstrap porque ao usar o método parece estar puxando-se por seu próprio bootstrap.
- O termo “bootstrap” pode se referir a bootstrap não-paramétrico ou paramétrico. Métodos Monte Carlo que envolvem amostragem de uma distribuição de probabilidade completamente especificada são algumas vezes chamados de bootstrap paramétrico.

3.1.1 Ideia do bootstrap

Métodos de reamostragem tratam uma amostra observada como uma população finita e amostras aleatórias são geradas dela para estimar características populacionais e fazer inferência sobre a população amostrada. Desta forma:

- A distribuição da população finita representada pela amostra pode ser considerada como uma pseudo-população com características semelhantes à da população verdadeira.
- Ao gerar repetidamente amostras aleatórias a partir dessa pseudo-população (reamostragem), a distribuição amostral de uma estatística pode ser estimada e propriedades de um estimador, como viés ou erro padrão, podem ser estimadas.

As estimativas de Bootstrap de uma distribuição de amostragem são análogas à ideia de estimativa de densidade, pois

- Um histograma pode ser visto como uma estimativa razoável da função densidade, enquanto que a função de distribuição acumulada empírica é uma estimativa da função de distribuição acumulada teórica;
- No bootstrap geramos amostras aleatórias a partir da distribuição empírica da amostra, ou seja, a partir da função de distribuição acumulada empírica.

Suponha que $\mathbf{x} = (x_1, \dots, x_n)^T \in \mathbb{R}^n$ é uma amostra aleatória observada de uma distribuição com função de distribuição acumulada (fda) $F(x)$. Se X^* é selecionado aleatoriamente de **x**, então

$$P(X^* = x_i) = \frac{1}{n}, \;\; i = 1, \dots, n.$$

A reamostragem fornece uma amostra aleatória $X_1^*, \dots, X_n^*$ por amostrar com reposição de **x**. Assim, X^* são variáveis aleatórias iid com distribuição Uniforme no conjunto $\{x_1, \dots, x_n\}$. Note que a fda empírica $F_n(x)$ obtida com a amostra original é um estimador de $F(x)$ e também é a fda de X^*, pois reamostrar de **x** é equivalente a gerar da distribuição $F_n(x)$.

Suponha que $\theta = g(F)$ é o parâmetro de interesse e sejam:

- $\widetilde{\theta} = g(F_n)$ o valor do parâmetro quando F_n é a distribuição de probabilidade;
- $\widehat{\theta} = s(\mathbf{x})$ uma estimativa de θ obtida com a amostra observada;
- $\widehat{\theta}^* = s(\mathbf{x}^*)$ uma estimativa obtida com uma amostra bootstrap $\mathbf{x}^*$.

Observação: Frequentemente $\tilde{\theta}$ e $\hat{\theta}$ são iguais, mas podem ser diferentes. Por exemplo, se $\hat{\theta}$ for a média aparada dos dados e $\tilde{\theta}$ for a média da distribuição F_n.

Os métodos bootstrap fazem uma das duas grandes suposições a seguir:

- A fda empírica F_n é uma boa aproximação da fda F_X, então a distribuição de $\hat{\theta}^*$ é similar a distribuição de $\hat{\theta}$;
- A distribuição de $(\hat{\theta}^* - \tilde{\theta})$ é similar a distribuição de $(\hat{\theta} - \theta)$.

Exemplo 3.1. Suponha que observamos a amostra

$$\mathbf{x} = \{2, 2, 1, 1, 5, 4, 4, 3, 1, 2\}$$

e que esta amostra foi obtida da distribuição de Poisson($\lambda = 2$).

Ao reamostrar um valor da variável $\mathbf{X}^*$ a partir de **x**, obtemos ue

$$F_{X^*}(x) = F_n(x) = \begin{cases} 0, & x < 1; \\ 0,3, & 1 \leq x < 2; \\ 0,6, & 2 \leq x < 3; \\ 0,7, & 3 \leq x < 4; \\ 0,9, & 4 \leq x < 5; \\ 1, & x \geq 5. \end{cases}$$

- Observe que, se F_n não estiver perto de F, a distribuição das réplicas não estará próxima a F.
- Reamostragem de x com um grande número de réplicas produz uma boa estimativa de F_n, mas não uma boa estimativa de F, porque independentemente de quantas repetições são desenhadas, as amostras de bootstrap nunca incluirão o valor 0.

Em ambos os pressupostos citados anteriormente, a tarefa de fazer inferências sobre θ se reduz a aprender sobre a distribuição bootstrap de $\hat{\theta}^*$. Às vezes, os aspectos relevantes da distribuição de bootstrap podem ser determinados matematicamente, mas, na maioria dos problemas não-triviais, a distribuição deve ser estimada usando métodos Monte Carlo.

A estimativa bootstrap da distribuição de $\hat{\theta}$ é obtido por:

Algoritmo

Passo 1: Para cada réplica bootstrap, indexada por $b = 1, \dots, B$:

1. Gere uma amostra
$$\mathbf{x}^{*(b)} = (x_1^*, \dots, x_n^*)^T$$
por amostrar com reposição da amostra observada $\mathbf{x} = (x_1, \dots, x_n)^T \in \mathbb{R}^n$.

2. Calcule $\widehat{\theta}^{(b)}$ com a b-ésima amostra bootstrap.

Passo 2: A estimativa bootstrap de $F_{\hat{\theta}}$ é dada pela distribuição empírica de $\widehat{\theta}^{(1)}, \dots, \widehat{\theta}^{(B)}$.

3.2 Estimação do erro padrão e Viés

O método bootstrap pode ser fácilmente utilizado para estimar tanto o erro padrão quanto o viés de um estimador.

3.2.1 Estimação do erro padrão

A estimativa bootstrap do erro padrão de um estimador $\widehat{\theta}$ é o desvio padrão amostral das réplicas bootstrap $\widehat{\theta}^{(1)}, \dots, \widehat{\theta}^{(B)}$. Isto é,

$$\widehat{se}(\widehat{\theta}) = \sqrt{\frac{1}{B-1}\sum_{b=1}^{B}\left[\widehat{\theta}^{(b)} - \bar{\theta}\right]^2},$$

em que $\bar{\theta} = \frac{1}{B}\sum_{b=1}^{B}\widehat{\theta}^{(b)}$.

Observação: De acordo com Efron e Tibshirani, o número de réplicas necessárias para boas estimativas do erro padrão não é grande, $B = 50$ é suficiente usualmente e raramente temos $B > 200$.

Exemplo 3.2. O conjunto de dados da escola de direito no pacote bootstrap é de Efron e Tibshirani. O conjunto de dados contém LSAT (pontuação média na pontuação do teste de

admissão na faculdade de direito) e GPA (nota média na metade da graduação) para 15 faculdades de direito:

LSAT 576 635 558 578 666 580 555 661 651 605 653 575 545 572 594
GPA 339 330 281 303 344 307 300 343 336 313 312 274 276 288 296

Este conjunto de dados é uma amostra aleatória do universo de 82 faculdades de direito (law82 - bootstrap). O objetivo é estimar a correlação entre as pontuações LSAT e GPA e calcular a estimativa de bootstrap do erro padrão da correlação da amostra.

No R, obtemos que:

```
library(bootstrap) # Para acesso aos dados
cor(law$LSAT, law$GPA)
```

```
## [1] 0.7763745
```

```
# Bootstrap
B <- 2000 # número de réplicas
n <- nrow(law); theta.b <- numeric(B)
for (b in 1:B) {
  i <- sample(1:n, size = n, replace = TRUE) # índices da reamostragem
  LSAT <- law$LSAT[i]; GPA <- law$GPA[i] # reamostragem
  theta.b[b] <- cor(LSAT, GPA)
}
(se.R <- sd(theta.b))
```

```
## [1] 0.1280294
```

```
par(mar = c(4, 4, 1, 1))
hist(theta.b, prob = TRUE, breaks=15, ylab = "Densidade",
     xlab = "Estimativas nas réplicas", main = "")
```

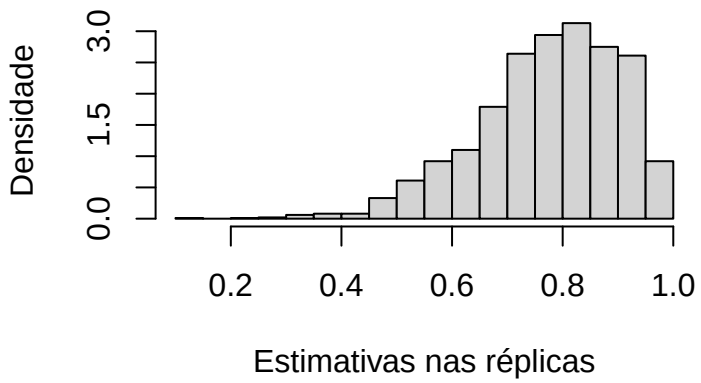

Figura 3.1: Histograma das estimativas nas réplicas bootstrap.

Podemos também utilizar o pacote boot no R.

```
library(boot)
r <- function(x, i) { # Função para calcular correlação
  cor(x[i,1], x[i,2])
}

obj <- boot(data = law, statistic = r, R = 2000)
sd(obj$t)
```

```
## [1] 0.1324654
```

3.2.2 Estimação do viés

O viés de um estimador $\hat{\theta}$ para o parâmetro θ é dado por

$$Vis(\hat{\theta}) = E_F[\hat{\theta} - \theta] = E_F[s(\mathbf{X})] - g(F),$$

A estimação bootstrap utiliza as réplicas bootstrap de $\hat{\theta}$ para estimar a distribuição amostral de $\hat{\theta}$. Uma estimativa bootstrap do víes é obtida ao substituir F por F_n e assim

$$\widehat{Vis}(\hat{\theta}) = E_{F_n}[s(\mathbf{X}^*)] - g(F_n) = \bar{\theta} - \tilde{\theta},$$

em que $\bar{\theta} = \frac{1}{B}\sum_{b=1}^{B} \hat{\theta}^{(b)}$, pois a média amostral das replicações $\{\hat{\theta}^{(b)}\}$ é um estimador não viciado para $E[\hat{\theta}^*]$.

Exemplo 3.3. Qual é o viés do estimador de correlação para os dados de LSAT e GPA?

```
## Estimando o viés - no exemplo anterior
theta.hat <- cor(law$LSAT, law$GPA)
B <- 2000 # Número de replicas bootstrap
n <- nrow(law); theta.b <- numeric(B)
for (b in 1:B) {
  i <- sample(1:n, size = n, replace = TRUE)
  LSAT <- law$LSAT[i]; GPA <- law$GPA[i]
  theta.b[b] <- cor(LSAT, GPA)
}
bias <- mean(theta.b - theta.hat)
bias
```

```
## [1] -0.007295889
```

Exemplo 3.4. Os dados de Efron e Tibshirani contêm medidas de um certo hormônio na corrente sanguínea de oito indivíduos após o uso de um adesivo médico. O parâmetro de interesse é

$$\theta = \frac{E(new) - E(old)}{E(old) - E(placebo)},$$

em que $E(new)$, $E(old)$ e $E(placebo)$ são os valores esperado da quantidade de hormônio na corrente sanguínea com, respectivamente, o novo adesivo, o adesivo antigo e placebo.

Se $|\theta| \leq 0,2$, isso indica bioequivalência dos adesivos antigo e novo. A estatística é $R = \bar{Y}/\bar{Z}$, em que $Y = new - old$ representa a diferença de quantidade de hormônio ao utilizar o adesivo novo e antigo, enquanto que $Z = old - placebo$ representa a diferença de quantidade de hormônio ao utilizar o adesivo antigo e placebo.

Desejamos calcular uma estimativa bootstrap de viés na estatística de razão de bioequivalência R.

```
data(patch, package = "bootstrap") # Dados no pacote bootstrap
patch
```

```
##   subject placebo oldpatch newpatch     z     y
## 1       1    9243    17649    16449  8406 -1200
## 2       2    9671    12013    14614  2342  2601
## 3       3   11792    19979    17274  8187 -2705
## 4       4   13357    21816    23798  8459  1982
## 5       5    9055    13850    12560  4795 -1290
## 6       6    6290     9806    10157  3516   351
## 7       7   12412    17208    16570  4796  -638
## 8       8   18806    29044    26325 10238 -2719
```

```
n <- nrow(patch); B <- 2000; theta.b <- numeric(B)
theta.hat <- mean(patch$y) / mean(patch$z)
#bootstrap
for (b in 1:B) {
  i <- sample(1:n, size = n, replace = TRUE)
  y <- patch$y[i]; z <- patch$z[i]
  theta.b[b] <- mean(y) / mean(z)
}
```

```
bias <- mean(theta.b) - theta.hat
se <- sd(theta.b)
round( data.frame(est = theta.hat, bias = bias, se = se, cv = bias/se), 4)
```

```
##       est   bias  se     cv
## 1 -0.0713 0.0073 0.1 0.0726
```

3.2.3 Exercícios

Exercício 3.1. Gere amostras de tamanho $n = 10$, 30 e 50, e estime erro padrão e viés da média amostral como estimador da média populacional nas seguintes situações:

a. $X \sim N(0, \sigma^2)$;
b. $X \sim Gama(2, 5)$.

Compare com o valor teórico.

Exercício 3.2. Considere os seguintes estimadores para variância populacional:

$$\hat{\sigma}_1^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 \text{ e } \hat{\sigma}_2^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2.$$

Gere amostras de tamanho $n = 30$ e estime erro padrão e viés com estes estimadores nas seguintes situações:

a. $X \sim N(0, \sigma^2)$;
b. $X \sim Gama(2, 5)$.

3.3 Jackknife

Jackknife é um método de reamostragem proposto por Quenouille(1949) como uma técnica para redução de viés e por Tukey para estimar o erro padrão. No Jackknife, como em um tipo de validação cruzada, são consideradas subamostras em que cada x_i é omitido.

Da mesma forma que visto anteriormente $\mathbf{x} = (x_1, \ldots, x_n)^T \in \mathbb{R}^n$ uma amostra aleatória observada de uma distribuição com fda $F(x)$ e $\hat{\theta} = s(\mathbf{x})$. Definimos $\mathbf{x}_{[-i]} = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)^T$ o subconjunto de $\mathbf{x}$ sem a i-ésima observação e $\hat{\theta}_{[-i]} = s(\mathbf{x}_{[-i]}), i = 1, \ldots, n$. Também supomos que:

- O parâmetro $\theta = g(F)$ é uma função da distribuição F;
- A estimativa "plug-in" de θ é $\hat{\theta} = g(F_n)$, em que F_n é a fda empírica de uma amostra aleatória de F;
- O estimador de $\hat{\theta}$ é suave no sentido que pequenas mudanças nos dados correspondem a pequenas mudanças em $\hat{\theta}$.

3.3.1 Jackknife para estimar viés

Se $\hat{\theta}$ é uma estatística suave, então $\hat{\theta}_{[-i]} = g(F_{n-1}(x_{[-i]}))$ e a estimativa jackknife do viés é dada por

$$\widehat{Vis}_{jack}(\hat{\theta}) = (n-1)(\bar{\theta}_{[.]} - \hat{\theta}),$$

em que $\bar{\theta}_{[.]} = \frac{1}{n}\sum_{i=1}^{n}\hat{\theta}_{[-i]}$ é a média das estimativas obtidas com as amostras com uma observação retirada. O fator $n-1$ aparece para obter estimador jackknife não viesado para o viés do estimador plug-in.

Após uma correção de viés, um estimador jackknife para θ é dado por

$$\hat{\theta}_J = n\hat{\theta} - \frac{n-1}{n}\sum_{i=1}^{n}\hat{\theta}_{[-i]}.$$

3.3.2 Jackknife para estimar erro padrão

Para estatísticas $\hat{\theta}$ suaves, uma estimativa jackknife do erro padrão é

$$\widehat{se}_{jack}(\hat{\theta}) = \sqrt{\frac{n-1}{n}\sum_{i=1}^{n}\left(\hat{\theta}_{[-i]} - \bar{\theta}_{[.]}\right)^2}.$$

O fator $\frac{n-1}{n}$ faz com que $\hat{se}_{jack}$ seja um estimador não viciado do erro padrão da média.

Exemplo 3.5. (Revisitando o Exemplo 3.4) Os dados de Efron e Tibshirani contêm medidas de um certo hormônio na corrente sanguínea de oito indivíduos após o uso de um adesivo médico. O parâmetro de interesse é

$$\theta = \frac{E(new) - E(old)}{E(old) - E(placebo)}.$$

Se $|\theta| \leq 0,2$, isso indica bioequivalência dos adesivos antigo e novo. A estatística é $\bar{Y}/\bar{Z}$.

Desejamos calcular uma estimativa jackknife de viés na estatística de razão de bioequivalência.

No R fazemos:

```
data(patch, package = "bootstrap")
n <- nrow(patch); y <- patch$y; z <- patch$z
(theta.hat <- mean(y) / mean(z))
```

```
## [1] -0.0713061
```

```
# Réplicas jackkife
theta.jack <- numeric(n)
for (i in 1:n)  theta.jack[i] <- mean(y[-i]) / mean(z[-i])

bias <- (n - 1) * (mean(theta.jack) - theta.hat)
bias # Estimativa Jackknife do vies
```

```
## [1] 0.008002488
```

```
se <- sqrt( (n-1) *  mean((theta.jack - mean(theta.jack))^2) )
se # Estimativa Jackknife do erro padrão
```

```
## [1] 0.1055278
```

Exemplo 3.6. Qual é a estimativa do erro padrão da mediana de uma amostra aleatória de 10 inteiros de 1,...,100?

```
set.seed(1234)
n <- 10
x <- sample(1:100, size = n, replace = F)

# Estimativa Jackknife do erro padrão
M <- numeric(n)
for (i in 1:n) {
  y <- x[-i]
  M[i] <- median(y)
}
Mbar <- mean(M)
(sd_jack <- sqrt((n-1)/n * sum((M - Mbar)^2)))
```

```
## [1] 9
```

```
# Estimativa bootstrap do erro padrão
Mb <- replicate(1000, expr = {
  y <- sample(x, size = n, replace = TRUE)
  median(y) })
sd(Mb)
```

```
## [1] 17.02882
```

Para o tamanho amostral igual a 10 a estimativa será precisa? Vamos ver em um estudo de simulação Monte Carlo quais são as estimativas obtidas via Bootstrap e comparar com o valor obtido via Jackknife.

```
# Estudo Monte Carlo: Estimativa bootstrap do erro padrão
sd_boot <- numeric(1000)
for(j in 1:1000){
  Mb <- replicate(1000, expr = {
    y <- sample(x, size = n, replace = TRUE)
    median(y) })
  sd_boot[j] <- sd(Mb)
}
mar = c(3, 4, 1, 1)
hist(sd_boot, prob = T, xlab = "Erro padrão" , ylab = "Densidade", main = "")
```

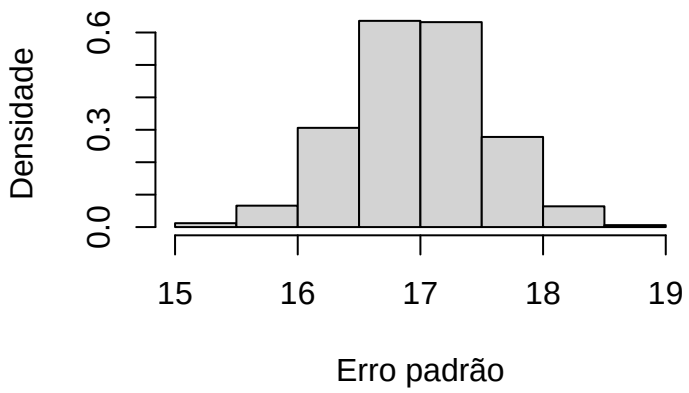

Figura 3.2: Histograma das estimativas bootstrap de erro padrão do estimador em um estudo Monte Carlo.

Alguma coisa parece estar errada para estas estimativas do bootstrap e do jackknife serem tão diferentes e o que ocorre é que a mediana não é uma estatística suave!

3.3.3 Exercícios

Exercício 3.3. Calcule a estimativa jackknife do erro padrão e do viés para o coeficiente de correlação dos dados da faculdade de direito no Exemplo 4.2. Compare estas estimativas com as estimativas bootstrap para as mesmas quantidades.

Exercício 3.4. Gere 100 amostras $X_1, X_2, \ldots, X_{20}$ de uma população $Normal(\theta, 1)$ com $\theta = 1$.

(a) Para cada amostra calcule as estimativas bootstrap e jackknife da variância para $\hat{\theta} = \bar{X}$ e calcule a média e o desvio padrão das estimativas de variância nestas 100 amostras.

(b) Repita o item (a) para o estimador $\hat{\theta} = \bar{X^2}$ e compare os resultados.

3.4 Intervalos de confiança bootstrap

Existem várias abordagens para obter intervalos de confiança aproximados para o parâmetro de interesse θ, entre eles, os intervalos de confiança obtidos pelos métodos de bootstrap normal padrão, básico, percentílico, Studentizado e percentílico acelerado com correção de viés.

3.4.1 Intervalo de confiança bootstrap normal

Este intervalo de confiança possui uma abordagem simples, mas não é necessariamente a melhor. Se $\hat{\theta}$ é uma média amostral e o tamanho amostral é grande, então o Teorema Central do Limite implica que a estatística

$$Z = \frac{\hat{\theta} - E[\hat{\theta}]}{se(\hat{\theta})}$$

é aproximadamente normal padrão. Logo, se $\hat{\theta}$ é um estimador não viesado para θ, um intervalo de confiança $100(1-\alpha)\%$ para θ é o intervalo

$$\left[\hat{\theta} \pm z_{\alpha/2} se(\hat{\theta})\right],$$

em que $z_{\alpha/2} = \Phi^{-1}(1-\alpha/2)$. O intervalo de confiança bootstrap normal é obtido estimando $se(\hat{\theta})$ através do bootstrap.

Este intervalo é de simples construção, mas faz as seguintes suposições:

i. A distribuição de $\widehat{\theta}$ é normal ou $\widehat{\theta}$ é a média amostral e o tamanho amostral é grande;
ii. $\widehat{\theta}$ é um estimador não viciado de θ. Note que o viés poderia ser estimado e usado para centrar a distribuição de Z, mas o estimador é uma variável aleatória e a variável transformada não tem distribuição Normal;
iii. O termo $se(\widehat{\theta})$ é tratado como conhecido, no entanto ele é estimado.

No R podemos implementar o intervalo de confiança bootstrap normal como abaixo:

```
IC_boot_normal = function(amostra, B = 200, estimador, conf = 0.95,...){
  # estimador é uma função a ser passada como parâmetro
  est_pontual <- estimador(amostra)

  ## Replicação
  reamostras <- replicate(n = B,
                          sample(x = amostra, size = length(amostra),
                                 replace = TRUE), simplify = T )
  est_reamostras <- apply(X = reamostras, FUN = estimador, MARGIN = 2 )
  # Limites do intervalo
  li <- est_pontual - qnorm((1+conf)/2) * sd(est_reamostras)
  ls <- est_pontual + qnorm((1+conf)/2) * sd(est_reamostras)
  return(c(li, ls))
}

## Teste
dados <- rnorm(100, 10, 3)
IC_boot_normal(amostra = dados, B = 200, estimador = mean, conf = 0.95)
```

```
## [1]  9.144792 10.318896
```

```
t.test(dados)$conf.int
```

```
## [1]  9.12397 10.33972
## attr(,"conf.level")
## [1] 0.95
```

3.4.2 Intervalo de confiança bootstrap percentílico

Suponha que $\hat{\theta}^{(1)}, \ldots, \hat{\theta}^{(B)}$ são as réplicas bootstrap da estatística $\hat{\theta}$. O intervalo de confiança $100(1-\alpha)\%$ é dado por

$$\left[\hat{\theta}_{\alpha/2},\ \hat{\theta}_{1-\alpha/2}\right],$$

em que $\hat{\theta}_{\alpha/2}$ é o percentil empírico calculado com a amostra $\hat{\theta}^{(1)}, \ldots, \hat{\theta}^{(B)}$. Note que esta abordagem utiliza a distribuição empírica das réplicas bootstrap como a distribuição de referência.

No R podemos implementar o intervalo de confiança bootstrap percentílico com a seguinte função.

```
IC_boot_perc = function(amostra, B = 200, estimador, conf = 0.95,...){
  # estimador é uma função a ser passada como parâmetro
  ## Replicação
  reamostras <- replicate(n = B,
                          sample(x = amostra, size = length(amostra),
                                 replace = TRUE), simplify = T )
  est_reamostras <- apply(X = reamostras, FUN = estimador, MARGIN = 2 )
  # Limites do intervalo
  li <- quantile(est_reamostras, (1-conf)/2)
  ls <- quantile(est_reamostras, (1+conf)/2)
  return(c(li, ls))
}

## Teste
dados <- rnorm(100, 10, 3)
IC_boot_perc(amostra = dados, B = 200, estimador = mean, conf = 0.95)
```

```
##      2.5%     97.5%
##  9.656466 10.769627
```

```
t.test(dados)$conf.int
```

```
## [1]  9.673693 10.828988
## attr(,"conf.level")
## [1] 0.95
```

3.4.3 Intervalo de confiança bootstrap básico

Suponha que $\hat{\theta}$ é um estimador de θ e $a_{\alpha/2}$ é tal que $P(\hat{\theta} - \theta > a_{\alpha/2}) = \alpha$ e, assim, $P(\hat{\theta} - a_{\alpha/2} > \theta) = \alpha/2$. Desta forma, um intervalo de confiança $100(1-\alpha)\%$ pode ser

obtido como

$$\left[\hat{\theta} - a_{1-\alpha/2},\ \ \hat{\theta} - a_{\alpha/2}\right].$$

Além disso, $P(\hat{\theta} > a_{\alpha/2} + \theta) = \alpha/2$ e $P(\hat{\theta} > \hat{\theta}_{\alpha/2}) = \alpha/2$. Através de bootstrap, $\hat{\theta}_{\alpha/2}$ é estimado com o percentíl empírico obtido de $\hat{\theta}^{(1)}, \ldots, \hat{\theta}^{(B)}$. Assim,

$$a_{\alpha/2} + \theta = \hat{\theta}_{\alpha/2}$$

e, como uma estimativa de θ é obtida com $\hat{\theta}$, o percentil $a_{\alpha/2}$ de ordem $1 - \alpha/2$ de $(\hat{\theta} - \theta)$ pode ser estimado através de $\hat{a}_{\alpha/2} = \hat{\theta}_{\alpha/2} - \hat{\theta}/2$. O limite superior do intervalo aproximado é dado por

$$\hat{\theta} - \hat{a}_{\alpha/2} = \hat{\theta} - (\hat{\theta}_{\alpha/2} - \hat{\theta}) = 2\hat{\theta} - \hat{\theta}_{\alpha/2}$$

e, similarmente, o limite inferior do intervalo é dado por $2\hat{\theta} - \hat{\theta}_{1-\alpha/2}$. Então, o intervalo de confiança $100(1-\alpha)\%$ é dado por

$$\left[2\hat{\theta} - \hat{\theta}_{1-\alpha/2},\ \ 2\hat{\theta} - \hat{\theta}_{\alpha/2}\right].$$

Note que este intervalo transforma a distribuição das réplicas do estimador por subtrair o valor observado da estatística.

No R, implementamos o intervalo de confiança bootstrap básico como a seguir:

```
IC_boot_bas = function(amostra, B = 200, estimador, conf = 0.95,...){
  # estimador é uma função a ser passada como parâmetro
  est_pontual <- estimador(amostra)

  ## Replicação
  reamostras <- replicate(n = B,
                          sample(x = amostra, size = length(amostra),
                                 replace = TRUE), simplify = T )
  est_reamostras <- apply(X = reamostras, FUN = estimador, MARGIN = 2 )
  # Limites do intervalo
  li <- 2*est_pontual - quantile(est_reamostras, (1+conf)/2)
  ls <- 2*est_pontual - quantile(est_reamostras, (1-conf)/2)
  return(c(li, ls))
}

## Comparando intervalo com o t-student
dados <- rnorm(100, 10, 3)
mat <- rbind(IC_boot_bas(amostra = dados, B = 200, estimador = mean, conf = 0.95),
```

```
t.test(dados)$conf.int)
rownames(mat) <- c("Boot-Basico", "t-student")
round(mat, 3)
```

```
##             97.5%   2.5%
## Boot-Basico 9.136 10.227
## t-student   9.120 10.241
```

3.4.4 Intervalo de confiança bootstrap t

O bootstrap t (ou *studentizado*) não usa distribuição t-Student como referência, mas usa a distribuição amostral de uma estatística (*studentized*) gerada por reamostragem. Neste método, o intervalo de confiança é dado por

$$\left[\hat{\theta} - t^*_{1-\alpha/2}\hat{se}(\hat{\theta}),\ \ \hat{\theta} - t^*_{\alpha/2}\hat{se}(\hat{\theta})\right],$$

em que $\hat{se}(\hat{\theta})$, $t^*_{1-\alpha/2}$ e $t^*_{\alpha/2}$ são calculados como descrito no algoritmo a seguir.

Algoritmo

Passo 1: Calcule a estatística observada $\hat{\theta}$;

Passo 2: Para cada réplica, indexada por $b = 1, \dots, B$:

1. Gere uma amostra
$$\mathbf{x}^{*(b)} = (x^*_1, \dots, x^*_n)^T$$
por amostrar com reposição da amostra observada $\mathbf{x} = (x_1, \dots, x_n)^T \in \mathbb{R}^n$.
2. Calcule $\hat{\theta}^{(b)}$ com a b-ésima amostra bootstrap;
3. Calcule ou estime o erro padrão $\hat{se}(\hat{\theta}^{(b)})$. (Uma estimativa bootstrap pode ser obtida por reamostrar da amostra atual $\mathbf{x}^{(b)}$);
4. Calcule $t^{(b)} = \frac{\hat{\theta}^{(b)} - \hat{\theta}}{\hat{se}(\hat{\theta}^{(b)})}$;

Passo 3: Encontre os quantis $t^*_{1-\alpha/2}$ e $t^*_{\alpha/2}$ da amostra ordenada de $t^{(b)}$;

Passo 4: Calcule $\hat{se}(\hat{\theta})$ dado pelo desvio padrão das réplicas $\hat{\theta}^{(b)}$;

Passo 5: Calcule os limites de confiança dados por

$$\left[\hat{\theta} - t^*_{1-\alpha/2}\hat{se}(\hat{\theta}),\ \ \hat{\theta} - t^*_{\alpha/2}\hat{se}(\hat{\theta})\right].$$

Uma desvantagem desta abordagem é fazer um bootstrap para cada réplica b com o intuito de estimar $\widehat{se}(\widehat{\theta}^{(b)})$, ou seja, são utilizados B bootstraps dentro de um bootstrap!

3.4.5 Intervalo bootstrap percentílico ajustado (BCa)

O intervalo de confiança bootstrap percentílico melhorado é chamado BCa em que temos "viés corrigido" e "ajustado para aceleração". Intervalos BCa são uma versão modificada de intervalos percentílicos que têm melhores propriedades teóricas e melhor desempenho.

Para um intervalo de confiança de $100(1-\alpha)\%$, os quantis habituais $\alpha/2$ e $1-\alpha/2$ são ajustados por dois fatores: uma correção para viés e uma correção para assimetria. A correção de viés é denotada z_0 e o ajuste de assimetria ou "aceleração" é dado por a.

Um intervalo bootstrap BCa de confiança de $100(1-\alpha)\%$ é calculado por

$$\begin{aligned}\alpha_1 &= \Phi\left(\hat{z}_0 + \frac{\hat{z}_0 + z_{\alpha/2}}{1-\hat{a}(\hat{z}_0+z_{\alpha/2})}\right),\\ \alpha_2 &= \Phi\left(\hat{z}_0 + \frac{\hat{z}_0 + z_{1-\alpha/2}}{1-\hat{a}(\hat{z}_0+z_{1-\alpha/2})}\right),\end{aligned}$$

em que $z_\alpha = \Phi^{-1}(\alpha)$,

$$\hat{z}_0 = \Phi^{-1}\left(\frac{1}{B}\sum_{b=1}^{B} I\{\hat{\theta}^{(b)} < \hat{\theta}\}\right) \text{ e } \hat{a} = \frac{\sum_{i=1}^{n}(\bar{\theta}_{[.]} - \hat{\theta}_{[-i]})^3}{6(\sum_{i=1}^{n}(\bar{\theta}_{[.]} - \hat{\theta}_{[-i]})^2)^{3/2}}.$$

Os limites sãos quantis empíricos das réplicas bootstrap e o intervalo BCa é $\left[\hat{\theta}_{\alpha_1},\ \hat{\theta}_{\alpha_2}\right]$.

Exemplo 3.7. (Revistando o Exemplo 3.4) Os dados de Efron e Tibshirani contêm medidas de um certo hormônio na corrente sanguínea de oito indivíduos após o uso de um adesivo médico. O parâmetro de interesse é

$$\theta = \frac{E(new) - E(old)}{E(old) - E(placebo)}.$$

Se $|\theta| \leq 0,2$, isso indica bioequivalência dos adesivos antigo e novo. A estatística é $\bar{Y}/\bar{Z}$.

Desejamos calcular uma estimativa intervalar da razão de bioequivalência.

```
library(boot) # para utilizar funções boot e boot.ci
data(patch, package = "bootstrap")
theta.boot_helper <- function(dat, ind) {
  # Função para calcular a estatística
  y <- dat[ind, 1];   z <- dat[ind, 2]
  mean(y) / mean(z)
}

theta.boot <- function(dat, ind) {
  # Função para calcular a estatística
  y <- dat[ind, 1];   z <- dat[ind, 2]
  b <- boot(dat[ind, ], theta.boot_helper, 2000) # para s.e. do Student
  return(c(mean(y)/mean(z), var(b$t)))
}

## Organizando banco de dados
y <- patch$y
z <- patch$z
dat <- cbind(y, z)
## Objeto bootstrap com estimativas de viés e erro padrão
boot.obj <- boot(dat, statistic = theta.boot, R = 2000, stype = "i")
# Intervalos de confiança bootstrap
boot.ci(boot.obj, type = c("basic", "norm", "perc", "stud", "bca"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 2000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot.obj, type = c("basic", "norm", "perc",
##     "stud", "bca"))
##
## Intervals :
## Level      Normal              Basic             Studentized
## 95%   (-0.2687,  0.1199 )   (-0.2965,  0.0840 )   (-0.2536,  0.4092 )
##
## Level     Percentile            BCa
## 95%   (-0.2266,  0.1539 )   (-0.2131,  0.1921 )
```

```
## Calculations and Intervals on Original Scale
```

Exemplo 3.8. Comparação dos intervalos de confiança para a correlação nos dados da faculdade de direito.

```
library(boot)
data(law, package = "bootstrap")
statistic.boot <- function(x, i){
  cor(x[i,1], x[i,2])
}

## Objeto bootstrap com estimativas de viés e erro padrão
boot.obj <- boot(dat, statistic = statistic.boot, R = 2000, stype = "i")
# Intervalos de confiança bootstrap
boot.ci(boot.obj, type = c("basic", "norm", "perc", "bca"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 2000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot.obj, type = c("basic", "norm", "perc",
##     "bca"))
##
## Intervals :
## Level      Normal              Basic
## 95%   (-1.2406,  0.0574 )   (-1.3928, -0.1869 )
##
## Level     Percentile            BCa
## 95%   (-0.9495,  0.2563 )   (-0.9361,  0.4356 )
## Calculations and Intervals on Original Scale
```

3.4.6 Exercícios

Exercício 3.5. Implemente funções para calcular os intervalos de confiança bootstrap studentizado e BCa. Compare o resultado fornecido com o intervalo construído com os intervalos fornecidos pelas funções do pacote “boot”.

Exercício 3.6. Faça um estudo de Monte Carlo para comparar a probabilidade de cobertura dos intervalos de confiança bootstrap para a média populacional gerando amostras das distribuições Normal e Lognormal. Condidere três métodos bootstrap de sua escolha para

a obtenção dos intervalos de confiança. Faça considerando um tamanho amostral pequeno e compare com o intervalo de confiança teórico que utiliza a distribuição t-Student

3.5 Estruturas mais gerais de dados

No mundo real, um mecanismo de probabilidade desconhecido P fornece um conjunto de dados **x** observado. Em aplicações específicas, precisamos definir a regra de construção dos dados com mais cuidado. O conjunto de dados **x** pode não ser mais um único vetor. Ele tem uma forma dependente da estrutura de dados, por exemplo, $\mathbf{x} = (\mathbf{z}, \mathbf{y})$ no problema de duas amostras.

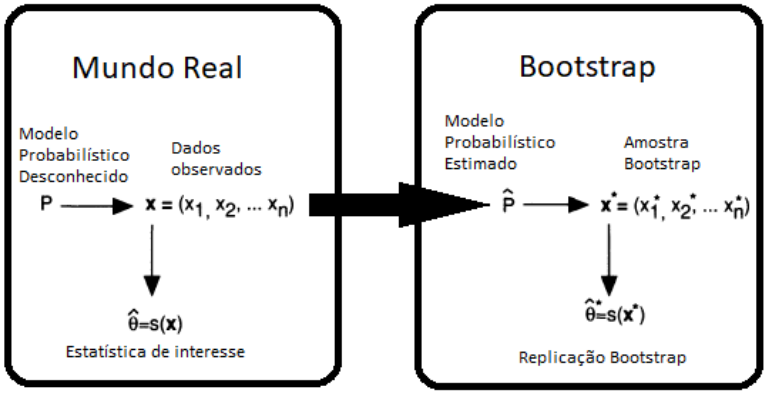

Figura 3.3: Ilustração do esquema de um Bootstrap.

Dois problemas práticos surgem:

1. Precisamos estimar todo o mecanismo de probabilidade P a partir dos dados observados **x**. É fácil de fazer para a maioria das estruturas de dados familiares. Nenhuma prescrição geral é possível, mas soluções ad hoc bastante naturais estão disponíveis.
2. Precisamos simular os dados de bootstrap de P de acordo com a estrutura de dados relevante. Este passo é conceitualmente direto, mas pode requerer algum cuidado na programação se a eficiência computacional for necessária.

3.5.1 Modelos de Regressão

O conjunto de dados **x** para um modelo de regressão linear consiste de n pontos $\mathbf{x}_1, \ldots, \mathbf{x}_n$, em que $\mathbf{x}_i = (\mathbf{c}_i, y_i)$, tal que $\mathbf{c}_i = (c_{i1}, \ldots, c_{ip})$ é um vetor de covariáveis, enquanto que y_i é a variável resposta. A suposição chave do modelo linear é que

$$\mu_i = E[Y_i|\mathbf{c}_i] = \mathbf{c}_i\beta = \sum_{j=1}^{p} c_{ij}\beta_j.$$

O vetor de parâmetros $\beta = (\beta_1, ..., \beta_p)^T$ é desconhecido e objetivo usual a análise de regressão é fazer inferência sobre β a partir dos dados observados.

A estrutura de probabilidade do modelo linear é usualmente expressa como

$$y_i = \mathbf{c}_i\beta + \epsilon_i, \text{ para } i = 1, 2, ..., n.$$

Assumimos que os temos de erro ϵ_i são uma amostra aleatória de uma distribuição desconhecida F com esperança 0, isto é,

$$F \to (\epsilon_1, ..., \epsilon_n) \quad (E[\epsilon_i] = 0).$$

Note que,

$$E[Y_i|\mathbf{c}_i] = E[\mathbf{c}_i\beta + \epsilon_i|\mathbf{c}_i] = \mathbf{c}_i\beta,$$

em que usamos o fato de que $E[\epsilon_i|\mathbf{c}_i] = E[\epsilon_i] = 0$, dado que ϵ_i é selecionado independentemente de $\mathbf{c}_i$.

Um vetor de valores **b** para β leva ao erro quadrático residual por $RSE(\mathbf{b}) = \sum_{i=1}^{n}(y_i - \mathbf{c}_i\mathbf{b})^2$ e a estimativa de mínimos quadrados de β é o valor que minimiza $RSE(\mathbf{b})$. A estimativa de mínimos quadrados é dada pela solução de $\mathbf{C}^T\mathbf{C}\hat{\beta} = \mathbf{C}^T\mathbf{y}$, que é dada por

$$\hat{\beta} = (\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\mathbf{y},$$

em que **C** é tal que a sua i-ésima linha contém $\mathbf{c}_i$ e **y** é o vetor $(y_1, ..., y_n)^T$.

3.5.1.1 Estimativa do erro padrão dos coeficentes de regressão

O erro padrão de $\hat{\beta}_j$ é dado por

$$se(\hat{\beta}_j) = \sigma_F\sqrt{G^{jj}},$$

em que G^{jj} é o j-ésimo elemento da diagonal da matriz inversa $\mathbf{G}^{-1}$, tal que $\mathbf{G} = \mathbf{C}^T\mathbf{C}$ e $\sigma_F^2 = Var_F(\epsilon)$. Na prática, σ_F^2 é estimado por

$$\hat{\sigma}_F^2 = \sum_{i=1}^{n}(y_i - \mathbf{c}_i\hat{\beta})^2/n = RSE(\hat{\beta})/n,$$

ou pela versão com viés corrigido dada por

$$\tilde{\sigma}_F^2 = \sum_{i=1}^{n}(y_i - \mathbf{c}_i\hat{\beta})^2/(n-p) = RSE(\hat{\beta})/(n-p).$$

Desta forma, os correspondentes erros padrão estimados para os componentes de $\widehat{\beta}$ são

$$\hat{se}(\hat{\beta}_j) = \hat{\sigma}_F\sqrt{G^{jj}} \text{ ou } \tilde{se}(\hat{\beta}_j) = \tilde{\sigma}_F\sqrt{G^{jj}}.$$

3.5.1.2 Aplicação do bootstrap

Nenhum dos cálculos até agora requer o bootstrap, no entanto uma análise de bootstrap para o modelo de regressão linear pode ser útil para assegurar que o bootstrap está dando respostas razoáveis. Podemos aplicar o bootstrap a modelos de regressão mais gerais que não têm solução matemática, por exemplo, onde a função de regressão é não linear nos parâmetros β e usamos métodos de ajuste diferentes de mínimos quadrados.

Bootstrap dos resíduos

O modelo de probabilidade $P \rightarrow$ **x** para regressão linear tem duas componentes,

$$P = (\beta, F),$$

em que F é a distribuição de probabilidade dos termos de erro.

Se β é conhecido, sabemos que $\epsilon_i = y_i - \mathbf{c}_i\beta$ para $i = 1, \ldots, n$. Então podemos calcular uma aproximação para os erros

$$\hat{\epsilon_i} = y_i - \mathbf{c}_i\hat{\beta}, \text{ para } i = 1, \ldots, n.$$

A estimativa para F é a distribuição empírica de $\hat{\epsilon}_i$ dada por

$$\hat{F} : \text{probabilidade } 1/n \text{ de sair } \hat{\epsilon}_i,\ i = 1, \ldots, n.$$

Com $\hat{P} = (\hat{\beta}, \hat{F})$, sabemos como gerar os conjuntos de dados bootstrap para o modelo de regressão linear: $\hat{P} \rightarrow$ **x***. Para gerar **x***, primeiro selecionamos uma amostra bootstrap dos erros aleatórios,

$$\hat{F} \rightarrow (\epsilon_i^*, \ldots, \epsilon_n^*) = \epsilon^*.$$

Então, as respostas bootstrap y_i^* são geradas de acordo com

$$y_i^* = \mathbf{c}_i\hat{\beta} + \epsilon_i^*, \text{ para } i = 1, \ldots, n.$$

A estimativa de mínimos quadrados bootstrap é dada por

$$\hat{\beta}^* = (\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T\mathbf{y}^*.$$

Neste caso, não precisamos de simulações de Monte Carlo para descobrir erros padrão de bootstrap,

$$\begin{aligned} Var(\hat{\beta}^*) &= (\mathbf{C}^T\mathbf{C})^{-1}\mathbf{C}^T Var(\mathbf{y}^*)\mathbf{C}(\mathbf{C}^T\mathbf{C})^{-1} \\ &= \hat{\sigma}_F^2(\mathbf{C}^T\mathbf{C})^{-1}, \end{aligned}$$

dado que $Var(\mathbf{y}^*) = \hat{\sigma}_F^2\mathbf{I}$ em que $\mathbf{I}$ é a matriz identidade.

Note que, o termo $\hat{\sigma}_F^2$ é estimado com os dados originai. Supondo que $\hat{\sigma}_F^2$ não seja conhecido, como é o caso em modelos mais complicados, com as réplicas bootstrap podemos estimar $\hat{\sigma}_F^2$. Se fizermos $B = \infty$, obtemos $\hat{se}_\infty(\hat{\beta}_j)$ tal que

$$\hat{se}_\infty(\hat{\beta}_j) = se_{\hat{F}}(\hat{\beta}_j^*) = \hat{\sigma}_F\sqrt{G^{jj}}.$$

Em outras palavras, a estimativa bootstrap do erro padrão para β_j é igual a estimativa usual!

Bootstrap dos pares

Enquanto que o método apresentado anteriormente produz conjuntos de dados da forma $\mathbf{x}^* = \{(\mathbf{c}_1, \mathbf{c}_1\hat{\beta} + \hat{\epsilon}_{i_1}), ..., (\mathbf{c}_n, \mathbf{c}_n\hat{\beta} + \hat{\epsilon}_{i_n})\}$, existe um segundo método denominado de método dos pares. Este método considera $\mathbf{x}_i = (\mathbf{c}_i, y_i)$ de modo que um conjunto de dados de bootstrap $\mathbf{x}^*$ é da forma

$$\mathbf{x}^* = \{(\mathbf{c}_{i_1}, y_{i_1}), ..., (\mathbf{c}_{i_n}, y_{i_n})\}$$

para $i_1, ..., i_n$ sendo uma amostra aleatória dos inteiros de 1 a n.

Bootstrap dos pares vs bootstrap dos resíduos

A resposta de qual método de bootstrap é melhor depende de até que ponto acreditamos no modelo de regressão linear. Esse modelo assume que os erros tem a mesma distribuição F para qualquer valor de $\mathbf{c}_i$. Esta suposição pode falhar mesmo se a esperança $\mu_i = \mathbf{c}_i\beta$ esteja correta. Desta forma, temos os seguintes pontos:

- O bootstrap dos pares é menos sensível as suposições do modelo. A estimativa de erro padrão obtida por bootstrap dos pares dá respostas razoáveis, mesmo que as suposições do modelo de regressão estejam completamente erradas;
- A única suposição por trás deste bootstrap é que os pares originais $\mathbf{x}_i = (\mathbf{c}_i, y_i)$ foram amostrados aleatoriamente a partir de alguma distribuição F, onde F é uma distribuição em vetores $(\mathbf{c}, y)$ com dimensão $(p + 1)$;

- Mesmo que as suposições do modelo de regressão estejam corretas, não é um desastre. Pode-se mostrar que as estimativas obtidas por esse método se aproximam daquelas dadas pelo bootstrap dos resíduos quando o número de pares n se torna grande;
- O argumento inverso também pode ser feito. O modelo de regressão não precisa se manter perfeito para que os resíduos possam dar resultados razoáveis. Além disso, as diferenças nas distribuições de erro podem ser incorporadas no modelo, levando a uma versão mais apropriada dos resíduos de bootstrap.

Exemplo 3.9. (**Estudo com dados simulados**)
Para ilustrar o uso de bootstrap para estimar erro padrão dos estimadores dos coeficientes de regressão, é apresentado um estudo com dados simulados em que uma amostra de tamanho $n = 50$ foi gerada a partir de um modelo de regressão linear. O erros padrão dos coeficientes de regressão são estimados através da função lm, do bootstrap dos resíduos e do bootstrap dos pares.

No R temos:

```
## Gerando uma amostra
n <- 50
betas <- c(0.5, 2)
sigma2 <- 1
covariavel <- data.frame(x = rnorm(n))
X <- model.matrix(~x+1, covariavel)
preditor_linear <- X %*% betas
y <- rnorm(n, preditor_linear, sqrt(sigma2))
dados <- data.frame(y = y, covariavel)

# Ajustando um modelo linear com a função lm
ajuste <- lm(formula = y ~ 1+x, data = dados)
summary(ajuste)$coefficients
```

```
##              Estimate Std. Error  t value     Pr(>|t|)
## (Intercept) 0.4925194  0.1303806  3.77755 4.370695e-04
## x           1.9764371  0.1334700 14.80810 1.595004e-19
```

```
# Bootstrap dos resíduos
B = 200
betas = matrix(0, 200, 2)
```

```
residuos <- residuals(ajuste)
betas_est <- coefficients(ajuste)

for(i in 1:B){
  indices <- sample(1:n, n, replace = T)
  y_boot <- X %*% betas_est + residuos[indices]
  subamostra <- data.frame(y = y_boot, covariavel)
  betas[i,] <- coefficients(lm(formula = y ~ 1+x, data = subamostra))
}

apply(betas, 2, sd)
```

```
## [1] 0.1337412 0.1221817
```

```
# Bootstrap dos pares
B = 200
betas = matrix(0, 200, 2)

for(i in 1:B){
  indices <- sample(1:n, n, replace = T)
  amostra_boot <- dados[indices, ]
  betas[i,] <- coefficients(lm(formula = y ~ 1+x, data = amostra_boot))
}

apply(betas, 2, sd)
```

```
## [1] 0.1270624 0.1113338
```

3.5.2 Bootstrap paramétrico

A amostragem de bootstrap pode ser realizada de forma paramétrica, isto é, a diferença do bootstrap paramétrico para o não paramétrico é que as amostras são extraídas de uma estimativa paramétrica da distribuição da população ao invés da estimativa não paramétrica $\hat{F}$.

No bootstrap parâmetrico, extraímos B amostras de tamanho n tal que

$$\mathbf{x}_1^{*(b)}, \dots, \mathbf{x}_n^{*(b)} \overset{iid}{\sim} F(\mathbf{x}, \hat{\theta})$$

e calculamos a estimativa $\hat{\theta}$ em cada uma das B amostras. Por exemplo, a variância amostral desses B valores é uma estimativa da variância de $\hat{\theta}$.

Quando o modelo é conhecido ou acredita-se ser uma boa representação da realidade, o bootstrap paramétrico pode ser uma ferramenta poderosa, pois:

- permite inferência em situações de outra forma intratáveis;
- produz intervalos de confiança muito mais precisos do que aqueles produzidos pela teoria assintótica padrão.

É tentador usar um modelo conveniente, todavia, se o modelo não se encaixa bem no mecanismo que gera os dados, o bootstrap paramétrico pode levar a uma inferência errônea. No entanto, há ocasiões que poucas outras ferramentas inferenciais parecem viáveis.

Exemplo 3.10. (**Estudo de simulação Monte Carlo**)
Considerando dados amostrados da distribuição Poisson com parâmetro $\lambda = 10$, é apresentado um estudo de simulação, com 1000 réplicas, para estimar o erro padrão do estimador $\hat{\lambda} = \bar{X}$ utilizando tanto o bootstrap não-paramétrico quanto o bootstrap paramétrico. Nesta situação, sabe-se que $Var(\bar{X}) = Var(X)/n = \lambda/n$. Qual das versões de bootstrap apresenta melhor resultado?

No R temos:

```
## Estudo Monte Carlo
# Geração dos dados - X ~ Poisson(lambda) - E[X] = VAR[X] = lambda
# media amostral é um estimador de lambda e VAR(X_barra) = lambda/n

M = 1000; B = 200; n = 30; lambda = 10
se_npar <- se_par <- numeric(M)

for(i in 1:M){
  amostra <- rpois(n, lambda)
  lambda_est <- mean(amostra)
  est_boot_npar <- est_boot_par <- numeric(B)

  for(j in 1:B){
    # Bootstrap não paramétrico
    est_boot_npar[j] <- mean(sample(amostra, n, replace = T))

    # Bootstrap paramétrico
    est_boot_par[j] <- mean(rpois(n, lambda_est))
  }
  se_npar[i] <- sd(est_boot_npar);   se_par[i] <- sd(est_boot_par)
```

```
}

par(mfrow = c(1, 3), mar = c(4, 4, 2, 1))
hist(se_npar, prob = T, ylab = "Densidade", xlab = "Erro padrão",
     main = "Bootstrap \n Não-paramétrico")
abline(v = sqrt(lambda/n), col = 2, lty = 2, lwd = 2)
hist(se_par, prob = T, ylab = "Densidade", xlab = "Erro padrão",
     main = "Bootstrap \n paramétrico")
abline(v = sqrt(lambda/n), col = 2, lty = 2, lwd = 2)
boxplot(se_npar, se_par, names = c("Não-param.", "Param."),
        xlab = "Bootstrap", ylab = "Erro padrão")
abline(h = sqrt(lambda/n), col = 2, lty = 2, lwd = 2)
```

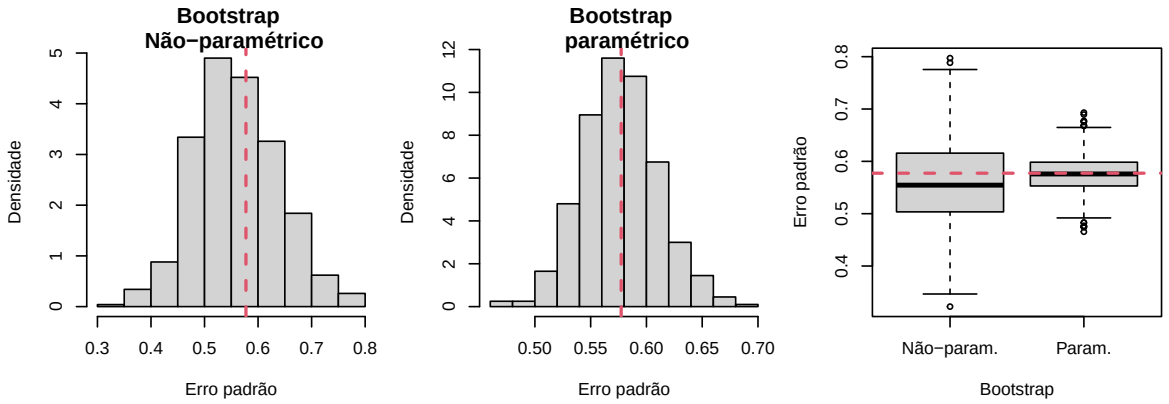

Figura 3.4: Histogramas e boxplot das estimativas de erro padrão do estimador nos bootstraps não-paramétrico e paramétrico.

3.5.3 Exercícios

Exercício 3.7. Consulte o conjunto de dados do ar condicionado, conjunto de dados "aircondit" fornecido no pacote boot do R. As 12 observações são os tempos em horas entre as falhas do equipamento de ar condicionado:

$$3, 5, 7, 18, 43, 85, 91, 98, 100, 130, 230, 487.$$

Suponha que os tempos entre as falhas seguem um modelo exponencial Exp(λ). Use bootstrap para estimar o viés e o erro padrão da estimativa fornecida pelo estimador de máxima verossimilhança da taxa de falha λ, isto é, considere a parametrização em que $E[X] = 1/\lambda$.

Exercício 3.8. Utilizando os dados do exercício anterior e utilizando bootstrap paramétrico, calcule os intervalos de 95% de confiança para o tempo esperado entre falhas $1/\lambda$ pelos métodos normal, percentílico e básico. Compare os intervalos obtidos.

3.6 Teste de hipóteses com o Bootstrap

Os testes de bootstrap são amplamente aplicáveis, embora menos precisos do que outras métodologias. Eles retornam resultados semelhantes aos testes de permutação (apresentados adiante) quando ambos estão disponíveis.

3.6.1 O problema de duas amostras

Observamos duas amostras aleatórias independentes **z** e **y** de possivelmente duas diferentes distribuições de probabilidade F e G,

$$\begin{aligned} F &\rightarrow \mathbf{z} = (z_1, \ldots, z_n) \text{ independente de} \\ G &\rightarrow \mathbf{y} = (y_1, \ldots, y_m) \end{aligned}$$

e desejamos testar a hipótese nula

$$H_0 : F = G.$$

Um teste de hipóteses pode ser baseado em uma estatística de teste $t(\mathbf{x})$, em que **x** é a amostra combinada de **z** e **y**. Podemos considerar, por exemplo, $t(\mathbf{x}) = \bar{z} - \bar{y}$ e decidir sobre as hipóteses a partir de

$$\text{p-valor} = P_{H_0}(|t(\mathbf{X}^*)| \geq |t(\mathbf{x})|).$$

A quantidade $t(\mathbf{x})$ é o valor observado da estatística de teste e a variável aleatória $\mathbf{X}^*$ tem uma distribuição especificada pela hipótese nula H_0, denotada por F_0.

A distribuição F_0 pode ser estimada pela distribuição empírica de **x**, denotada por $\hat{F}_0$, que assume probabilidade

$$1/(n+m)$$

para cada valor de **x**. Sob H_0, $\hat{F}_0$ fornece uma estimativa não-paramétrica da população comum que gerou **z** e **y**.

Testes mais precisos podem ser obtidos através do uso de uma estatística estudentizada, assim. podemos utilizar

$$t(\mathbf{x}) = \frac{\bar{z} - \bar{y}}{\bar{\sigma}\sqrt{1/n + 1/m}},$$

em que

$$\bar{\sigma} = \left[\frac{\sum_{i=1}^{n}(z_i - \bar{z})^2 + \sum_{j=1}^{m}(y_j - \bar{y})^2}{n+m-2} \right]^{1/2}.$$

O teste de hipóteses pode ser feito implementando o seguinte algoritmo:

Algoritmo

Passo 1: Gere B amostras de tamanho $n+m$ com reposição de **x**. Para cada uma das amostras, denote as primeiras n observações por $\mathbf{z}^*$ e as m observações restantes por $\mathbf{y}*$;
Passo 2: Avalie $t(\cdot)$ em cada reamostra, isto é, $t(\mathbf{x}^{*b})$;
Passo 3: Calcule

$$\widehat{\text{p-valor}}_{boot} = \sum_{b=1}^{B} I\{|t(\mathbf{x}^{*b})| \geq |t_{obs}|\}/B,$$

em que $t_{obs} = t(\mathbf{x})$ é o valor observado da estatística de teste.

O algoritmo acima testa a hipótese nula de que as duas populações são idênticas ($F = G$). E se quiséssemos testar apenas se suas médias eram iguais?

Se não estivermos dispostos a assumir que as variâncias nas duas populações são iguais, poderíamos basear o teste em

$$t(\mathbf{x}) = \frac{\bar{z} - \bar{y}}{\sqrt{\bar{\sigma}_1^2/n + \bar{\sigma}_2^2/m}},$$

em que

$$\bar{\sigma}_1 = \sum_{i=1}^{n}(z_i - \bar{z})^2/(n-1) \text{ e } \bar{\sigma}_2 = \sum_{j=1}^{m}(y_j - \bar{y})^2/(m-1).$$

A suposição de variância igual é atraente para o teste t porque simplifica a forma da distribuição da estatística de teste. Mas ao considerar um teste de hipótese de bootstrap para

comparar as duas médias, não há razão convincente para assumir variações iguais e, portanto, não fazemos essa suposição.

Para prosseguir, precisamos de estimativas de F e G que utilizem apenas a suposição de uma média comum. Seja $\bar{x}$ a média da amostra combinada, podemos:

- Transladar ambas as amostras de modo que tenham a média $\bar{x}$;
- Reamostrar de cada população separadamente.

Desta forma, o algoritmo é dado por:

Algoritmo

Passo 1: Faça $\hat{F}$ colocando igual probabilidade nos pontos

$$\tilde{z}_i = z_i - \bar{z} + \bar{x},$$

para $i = 1, \ldots, n$ e $\hat{G}$ colocando igual probabilidade nos pontos

$$\tilde{y}_i = y_i - \bar{y} + \bar{x},$$

para $i = 1, \ldots, m$, em que $\bar{z}$ e $\bar{y}$ são as médias dos grupos e $\bar{x}$ é a média da amostra combinada;

Passo 2: Gere B conjuntos de dados $(\mathbf{z}^*, \mathbf{y}^*)$ em que $\mathbf{z}^*$ é amostrado com reposição de $\tilde{z}_1, \ldots, \tilde{z}_n$ e $\mathbf{y}^*$ é amostrado com reposição de $\tilde{y}_1, \ldots, \tilde{y}_m$.

- Para cada banco de dados calcule

$$t(\mathbf{x}) = \frac{\bar{z}^* - \bar{y}^*}{\sqrt{\bar{\sigma}_1^{*2}/n + \bar{\sigma}_2^{*2}/m}},$$

Passo 3: Calcule

$$\widehat{\text{p-valor}}_{boot} = \sum_{b=1}^{B} I\{|t(\mathbf{x}^{*b})| \geq |t_{obs}|\}/B,$$

em que $t_{obs} = t(\mathbf{x})$ é o valor observado da estatística de teste.

Exemplo 3.11. Apresentamos um estudo de simulação para avaliar o desempenho do teste de hipóteses bootstrap para as hipóteses $H_0 : \mu_Z = \mu_Y$ versus $H_1 : \mu_Z \neq \mu_Y$.

```
n <- 10;  mu1 <- 5;  sigma1 <- 2
z <- rnorm(n, mu1, sigma1)
m <- 15;  mu2 <- 7;  sigma2 <- 4
y <- rnorm(m, mu2, sigma2)

## Valor observado da estatística de teste
est <- t.test(z, y, var.equal = F)$statistic

## Bootstrap
B <- 1000; est_ast <- numeric(B)
for(b in 1:B){
  amost_comb_ast <- sample(c(z, y), replace = T)
  z_ast <- amost_comb_ast[1:n]
  y_ast <- amost_comb_ast[-(1:n)]
  est_ast[b] <- t.test(z_ast, y_ast, var.equal = F)$statistic
}
p_valor <- ( sum(est_ast >= abs(est)) + sum(est_ast <= -abs(est)) )/B
p_valor
```

```
## [1] 0.07
```

```
par( mar = c(4, 4, 1, 1) )
hist(est_ast, prob = T, main = " ", ylab = "Densidade", xlab = "Estatística de teste")
abline(v = est, lty = 2, col = 2, lwd = 2)
```

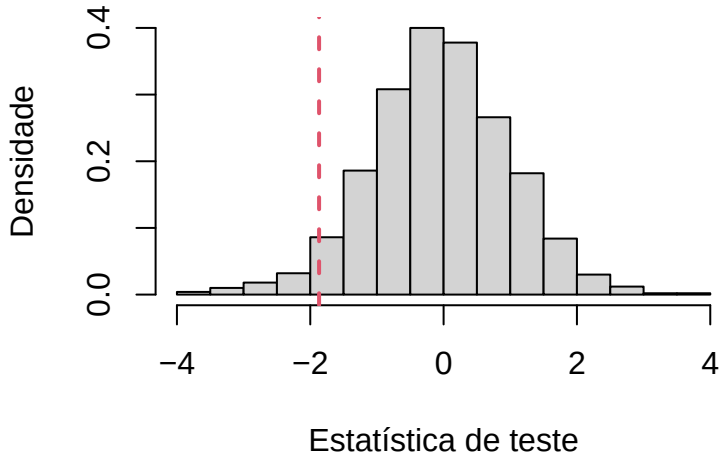

Figura 3.5: Histograma das réplicas da estatística de teste.

```r
teste_boot1 = function(z , y, B){
  teste = t.test(z, y, var.equal = F)
  est = teste$statistic
  est_ast = numeric(B)

  for(b in 1:B){
    amost_comb_ast = sample(c(z, y), replace = T)
    z_ast <- amost_comb_ast[1:n]
    y_ast <- amost_comb_ast[-(1:n)]
    est_ast[b] = t.test(z_ast, y_ast, var.equal=F)$statistic
  }
  p_valor = ( sum(est_ast >= abs(est)) + sum(est_ast <= -abs(est)) )/B
  return(p_valor)
}

## Estudo MC
n <- 10;  mu1 <- 7;  sigma1 <- 4
m <- 15;  mu2 <- 7;  sigma2 <- 4

M = 1000
pvalores = numeric(M)
for(i in 1:M){
  z <- rnorm(n, mu1, sigma1)
  y <- rnorm(m, mu2, sigma2)
  pvalores[i] <- teste_boot1(z, y, 1000)
}

# Probabilidade do erro tipo I
mean(pvalores <= 0.05)
```

```
## [1] 0.048
```

3.6.2 O problema com uma amostra

Suponha que observamos uma amostra **z** com distribuição de probabilidade F, isto é

$$F \ \rightarrow \ \mathbf{z} = (z_1, ..., z_n)$$

e desejamos testar a hipótese nula

$$H_0 : \mu_Z = \mu_0.$$

Um bootstrap pode ser utilizado considerando a estatística de teste

$$t(\mathbf{z}) = \frac{\bar{z} - \mu_0}{\hat{\sigma}/\sqrt{n}}.$$

Necessitamos de uma distribuição $\hat{F}$ que estima da distribuição F sob H_0 para calcular p-valor ou encontar a região crítica. Observe primeiro que a distribuição empírica $\hat{F}_n$ não é apropriada estimar para F porque não obedece a H_0, isto é, a média de F não é igual ao valor nulo de μ_0.

Uma maneira simples é transladar a distribuição empírica $\hat{F}_n$ para que tenha a média desejada. Em outras palavras, usamos como nossa distribuição nula estimada a distribuição empírica nos valores

$$\tilde{z}_i = z_i - \bar{z} + \mu_0, \ \text{ para } \ i = 1, ..., n.$$

Então, amostramos $\tilde{z}_1^*, ..., \tilde{z}_n^*$ com reposição de $\tilde{z}_1, ..., \tilde{z}_n$ e para cada amostra bootstrap calculamos a estatística

$$t(\mathbf{z}^*) = \frac{\bar{\tilde{z}}^* - \mu_0}{\hat{\tilde{\sigma}}^*/\sqrt{n}}.$$

Disto, obtemos

$$\widehat{\text{p-valor}}_{boot} = \sum_{b=1}^{B} I\{|t(\mathbf{z}^{*b})| \geq |t_{obs}|\}/B.$$

Existe uma maneira diferente, mas equivalente, de fazer um bootstrap no problema de uma amostra. Amostramos com reposição dos dados originais (não transladados) $z_1, ..., z_n$ e calculamos a estatística

$$t(\mathbf{z}^*) = \frac{\bar{z}^* - \bar{z}}{\hat{\sigma}^*/\sqrt{n}},$$

em que $\hat{\sigma}^*$ é o desvio padrão da reamostra. Esta estatística é igual a anterior pois $\bar{\tilde{z}}^* - \mu_0 = (\bar{z}^* - \bar{z} + \mu_0) - \mu_0 = \bar{z}^* - \bar{z}$ e os desvios padrão também são iguais.

Exemplo 3.12. Considere abaixo o código no R para fazer um teste bootstrap para H_0 : $\mu = \mu_0$ versus $H_1 : \mu \neq \mu_0$.

```
teste_boot_1mean = function(z, mu0, B){
  est = t.test(z, mu = mu0)$statistic
  est_ast = numeric(B)

  for(b in 1:B){
    z_til <- z - mean(z) + mu0
    z_ast = sample(z_til, replace = T)
    est_ast[b] = t.test(z_ast, mu = mu0)$statistic
  }
  p_valor = ( sum( est_ast >= abs(est) ) + sum( est_ast <= -abs(est) ) )/B
  return(p_valor)
}

# Testando e comparando com o teste t
n <- 20; mu1 = 6; sigma1 = 2
z <- rnorm(n, mu1, sigma1)

teste_boot_1mean(z, 6, B = 1000)
```

```
## [1] 0.972
```

```
t.test(z, mu = 6)$p.value
```

```
## [1] 0.976262
```

3.6.3 Exercícios

Exercício 3.9. Faça uma adaptação da função apresentada no Exemplo 4.11 de tal forma que exista um argumento para definir o tipo teste (bilateral, unilateral à esquerda ou unilateral à direita). Use um estudo Monte Carlo para estimar a probabilidade do erro tipo I em um teste unilateral à esquerda, considerando diferentes tamanhos amostrais e gerando dados da distribuição Gama. Utilize o mesmo tamanho amostral para as duas amostras das populações comparadas.

Exercício 3.10. Realize um estudo Monte Carlo para estimar a função poder do teste de hipóteses bootstrap para média de uma população e compare com a função poder estimada para o teste t em uma situação em que uma hipótese do teste t é violada.

3.7 Testes de permutação

Os testes de permutação são baseados em reamostragem, mas as amostras são geradas sem reposição. Eles podem ser aplicados para realizar testes não paramétricos de igualdade de distribuições, correlação, entre outros.

3.7.1 Igualdade de distribuições

Suponha duas amostras aleatórias independentes **z** e **y** de possivelmente duas diferentes distribuições de probabilidade F e G,

$$\begin{aligned} F &\rightarrow \mathbf{z} = (z_1, \dots, z_n) \text{ independente de} \\ G &\rightarrow \mathbf{y} = (y_1, \dots, y_m) \end{aligned}$$

Seja

- $\mathbf{x} = (z_1, \dots, z_n, y_1, \dots, y_n)$ a amostra agrupada, que é indexada por

$$V = \{1, \dots, n, n+1, \dots, n+m\} = \{1, \dots, N\};$$

- $\mathbf{X}^* = (\mathbf{Z}^*, \mathbf{Y}^*)$ representando uma partição da amostra agrupada **X**, em que $\mathbf{Z}^*$ tem n elementos e $\mathbf{Y}^*$ tem $m = N - n$ elementos.

Então, $\mathbf{X}^*$ é uma permutação v dos inteiros V, em que $z_i^* = z_{v(i)}$.

O número de possíveis partições é igual a $K = \binom{N}{n}$ e sob $H_0 : F = G$, uma amostra aleatória $\mathbf{Z}^*$ tem probabilidade

$$\frac{1}{\binom{N}{n}} = \frac{n!m!}{N!}$$

para quaisquer valores possíveis. Isto é, sob H_0 todas as permutações são igualmente prováveis.

Os grupos podem ser comparados de várias maneiras. Por exemplo, com médias amostrais, medianas ou médias aparadas. Mais geralmente, pode-se perguntar se as distribuições das duas variáveis diferem e comparar os grupos por qualquer estatística que mede a distância

entre duas amostras. Se $\widehat{\theta}(\mathbf{Z}, \mathbf{Y}) = \widehat{\theta}(\mathbf{X}, V)$ é uma estatística, então a distribuição de permutação de $\widehat{\theta}^*$ é a distribuição de replicações

$$\{\widehat{\theta}^*\} = \left\{\widehat{\theta}(\mathbf{X}, v_j(V)),\ j = 1, \ldots, K\right\}.$$

Assim,

$$\text{p-valor} = P(|\widehat{\theta}^*| \geq |\widehat{\theta}|) = K^{-1} \sum_{j=1}^{K} I\{|\widehat{\theta}^{(j)}| \geq |\widehat{\theta}|\},$$

onde $\widehat{\theta}$ é o valor calculado com a amostra observada.

Observações:

- O p-valor pode ser calculado de maneira similar para um teste unilateral a esquerda ou unilateral a direita.
- Na prática, a menos que o tamanho amostral seja muito pequeno, avaliar a estatística de teste para todas as permutações é computacionalmente intensivo.
- Um teste de permutação aproximado é implementado ao amostrar aleatoriamente um grande número de amostras sem reposição.

O algoritmo para teste de permutação aproximado é dado por:

Algoritmo

Passo 1: Calcule o valor observado $\widehat{\theta}(\mathbf{Z}, \mathbf{Y}) = \widehat{\theta}(\mathbf{X}, v)$ para a estatística de teste.
Passo 2: Para cada réplica, indexada por $b = 1, \ldots, B$:

1. Gere uma permutação aleatória $v_b = v(V)$;
2. Calcule a estatística $\widehat{\theta}^{(b)} = \widehat{\theta}(\mathbf{X}, v_b)$;

Passo 3: Se grandes valores de $\widehat{\theta}$ (em módulo) dão suporte a hipótese alternativa, calcule

$$\widehat{\text{p-valor}} = \frac{1 + \sum_{j=1}^{B} I\{|\widehat{\theta}^{(b)}| \geq |\widehat{\theta}|\}}{B + 1};$$

Passo 4: Rejeite H_0 se $\widehat{\text{p-valor}} < \alpha$.

Exemplo 3.13. São registrados pesos em gramas, para seis grupos de pintinhos recém-nascidos alimentados com suplementos diferentes. Existem seis tipos de suplementos alimentares. Sugere-se que os grupos soja e linhaça podem ser semelhantes. A distribuição de pesos para esses dois grupos é comparada.

No R temos:

```
attach(chickwts)
x <- sort(as.vector(weight[feed == "soybean"]))
y <- sort(as.vector(weight[feed == "linseed"]))
detach(chickwts)
x
```

```
##  [1] 158 171 193 199 230 243 248 248 250 267 271 316 327 329
```

```
y
```

```
##  [1] 141 148 169 181 203 213 229 244 257 260 271 309
```

```
# Teste de permutação com a estatística t para duas amostras
B <- 999 # Número de réplicas
z <- c(x, y) # amostra combinada
V <- 1:26
reps <- numeric(B) # guardar replicas
t0 <- t.test(x, y)$statistic # valor observado da estatística t

for (i in 1:B) {
  v <- sample(V, size = 14, replace = FALSE)
  x1 <- z[v]
  y1 <- z[-v] # complementar de x1
  reps[i] <- t.test(x1, y1)$statistic
}
p_valor <- mean(abs(c(t0, reps)) >= abs(t0))
p_valor
```

```
## [1] 0.182
```

```
par(mar = c(4, 4, 1, 1))
hist(reps, main = "", freq = FALSE, xlab = "T ", breaks = "scott", ylab = "Densidade")
abline(v = t0, lty = 2, col = 2, lwd = 2)
```

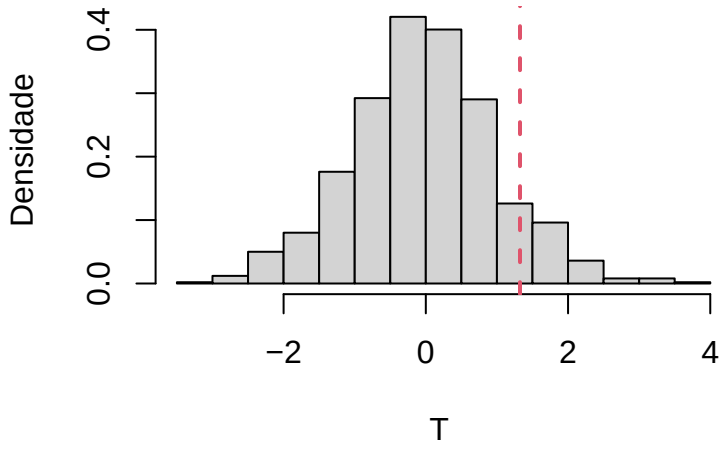

Figura 3.6: Histograma das réplicas de permutação da estatística de teste.

Também podemos utilizar a estatística de teste do teste de Kolmogorov-Smirnov. No R temos:

```
B <- 999
z <- c(x, y)
V <- 1:26
D <- numeric(B)
options(warn = -1)
D0 <- ks.test(x, y, exact = FALSE)$statistic
for (i in 1:B) {
  v <- sample(V, size = 14, replace = FALSE)
  x1 <- z[v]
  y1 <- z[-v]
  D[i] <- ks.test(x1, y1, exact = FALSE)$statistic
}

p_valor <- mean(c(D0, D) >= D0)
options(warn = 0)
p_valor
```

```
## [1] 0.478
```

```
par(mar = c(4, 4, 1, 1))
hist(D, main = "", freq = FALSE, xlab = "D", breaks = "scott", ylab = "Densidade")
```

```
abline(v = D0, lty = 2, col = 2, lwd = 2)
```

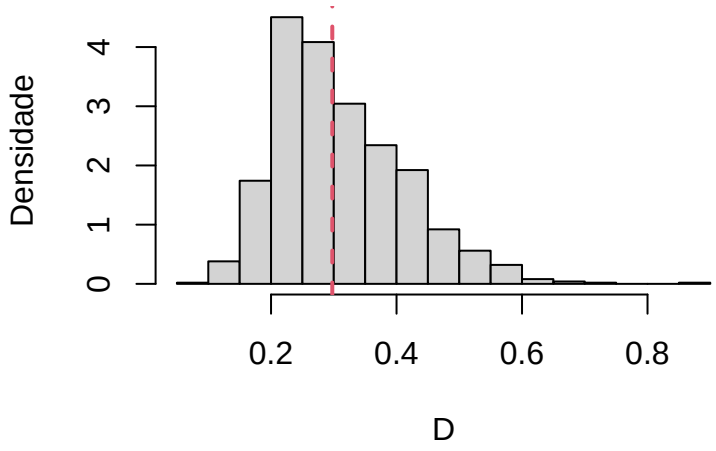

Figura 3.7: Histograma das réplicas de permutação da estatística de teste de Kolmogov-Smirnov.

3.7.2 Teste de correlação

Uma teste de correlação de Z e Y dado por

$$H_0 : \rho = 0 \text{ versus } H_1 : \rho \neq 0,$$

em que $\rho = cor(Z, Y)$. Diferentes definições de ρ medem diferentes tipos de associação.

Lembre que independência entre duas variáveis aleatórias implica que elas são não correlacionadas, mas o contrário não é verdadeiro. Sendo assim, se a hipótese nula não for rejeitada, não podemos afirmar que as variáveis sejam independentes, mas se rejeitamos podemos afirmar que elas são dependentes.

Seja $v = (v_1, \dots, v_n)$ o vetor de permutação que contém os inteiros $\{1, \dots, n\}$ em alguma ordem. Note que este vetor estará associado a ordenação de $y = (y_1, \dots, y_n)$ e existem $n!$ possíveis vetores v. Além disso, se H_0 é verdadeira, reordenar a amostra de y não afetará a correlação entre as variáveis Z e Y.

Sob H_0, como o vetor v tem probabilidade $1/n!$ de assumir cada um dos $n!$ possíveis

resultados, para testar $H_0 : \rho = 0$, obtemos

$$\text{p-valor} = \frac{\sum_{j=1}^{n!} I\{|\hat{\rho}^{(j)}| \geq |\hat{\rho}|\}}{n!}.$$

Quando $n!$ é muito grande utilizamos a aproximação Monte Carlo para estimar este p-valor.

No R, podemos implementar um teste de permutação para correlação com a função abaixo, em que estatística de correlação utilizada é um argumento da função.

```
## Teste de correlação
permcor_test <- function(z, y, method = "pearson", B = 10000,
                    alternative = c("two.sided", "less", "greater") ){
  n = length(z)
  if(n != length(y)) stop("Tamanhos de z e y devem ser iguais")
  theta.hat = cor(z, y, method = method)
  mat_permut = replicate( B, sample.int(n) )
  theta.mc = apply(mat_permut, 2, function(g) cor(z, y[g], method = method))
  if(alternative[1] == "less"){
    p_valor = sum(theta.mc <= theta.hat) / B
  } else if(alternative[1] == "greater"){
    p_valor = sum(theta.mc >= theta.hat) / B
  } else{
    p_valor = sum(abs(theta.mc) >= abs(theta.hat)) / B
  }
  list(theta.hat = theta.hat, theta.mc = theta.mc, p_valor = p_valor)
}
```

A seguir apresentamos um estudo com dados simulados utilizando a estatística de correlação de Pearson.

```
## Geração de dados
set.seed(1)
n = 50; rho = -0.2
z = rnorm(n);  y = rnorm(n)
Amat = matrix(c(1, rho, rho, 1), 2, 2)
Aeig = eigen(Amat, symmetric = TRUE)
evec = Aeig$vec
evalsqrt = diag(Aeig$val^0.5)
```

```
Asqrt = evec %*% evalsqrt %*% t(evec)
x = cbind(z, y) %*% Asqrt
z = x[, 1]; y = x[, 2]

## Pearson
ptest = permcor_test(z, y)
ptest$p_valor
```

```
## [1] 0.1011
```

```
par(mar = c(4, 4, 1, 1))
hist(ptest$theta.mc, ylab = "Frequência", main = "", xlab = expression(rho))
abline(v = ptest$theta.hat, col = "red", lty = 2, lwd = 2)
```

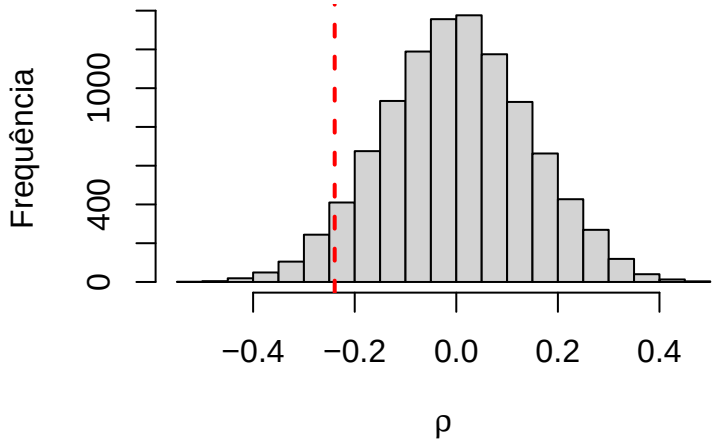

Figura 3.8: Histograma das réplicas de permutação da estatística de correlação de Pearson.

Agora é apresentado um estudo com dados simulados utilizando a estatística de correlação de Spearman e de Kendall.

```
## Spearman
ptest_sp = permcor_test(z, y, method = "spearman")
ptest_sp$p_valor
```

```
## [1] 0.0352
```

```
## Kendall
ptest_ke = permcor_test(z, y, method = "kendall")
```

```
ptest_ke$p_valor
```

```
## [1] 0.026
```

```
# Histogramas
par(mfrow = c(1, 2), mar = c(4, 4, 1, 1))
hist(ptest_sp$theta.mc, ylab = "Frequência", main = "", xlab = expression(rho))
abline(v = ptest_sp$theta.hat, col = "red", lty = 2, lwd = 2)
hist(ptest_ke$theta.mc, ylab = "Frequência", main = "", xlab = expression(rho))
abline(v = ptest_ke$theta.hat, col = "red", lty = 2, lwd = 2)
```

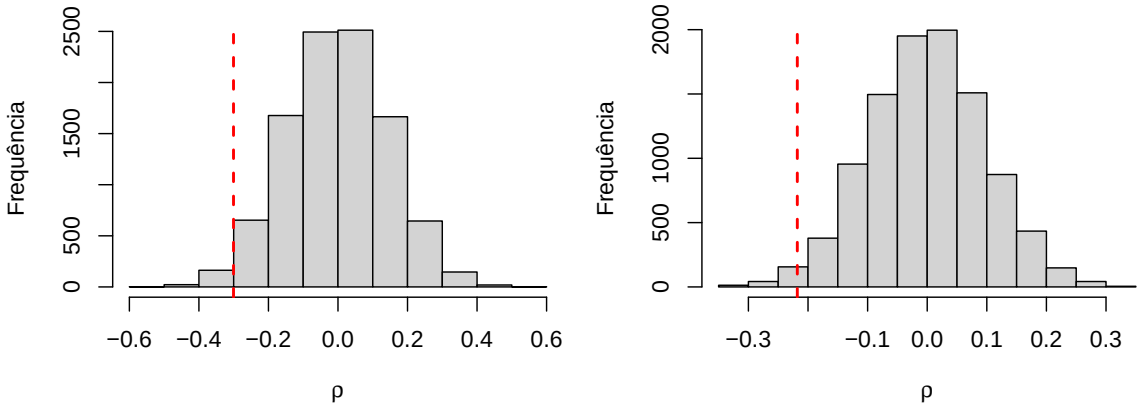

Figura 3.9: Histogramas das réplicas de permutação da estatística de correlação de Spearman e de Kendall.

3.7.3 Exercícios

Exercício 3.11. Implemente um teste de Cramer-von Mises para duas amostras para igualdade de distribuições como um teste de permutação. Aplique o teste para os dados do Exemplo 4.13.

Exercício 3.12. Faça um estudo Monte Carlo para calcular o valor empírico do nível de significância do teste de permutação descrito na Seção 4.7.1. Utilize a estatística do teste t para a média populacional.

Capítulo 4

Algoritmos Newton-Raphson, Escore de Fisher e EM

Neste capítulo iremos abordar conceitos relativos a métodos computacionais de máximização úteis para obtermos estimadores de máxima verossimilhança. O conteúdo apresentado é baseado nos Capítulos 2 e 4 do livro de Givens and Hoeting (2012), no Capitulo 11 do livro de Rizzo (2008) e alguns artigos citados ao longo do capítulo.

4.1 Introdução

Estimação de máxima verossimilhança é um tema central em inferência estatística. No entanto, em muitas situações, as funções a serem maximizadas não possuem soluções análiticas e torna-se necessário o uso de métodos númericos. Este é o caso de muitos modelos estatísticos realísticos que levam a funções de verossimilhança que não podem ser otimizadas analiticamente.

Em outras situações estatísticos também se deparam com problemas de otimização, como é o caso do uso de modelos em que o processo de estimação envolve:

- Minimizar o risco em um problema de decisão Bayesiano;
- Minimizar mínimos quadrados não lineares;
- Achar intervalos de mais alta densidade.

Todas essas tarefas são versões do mesmo problema genérico: otimizar uma função real g com respeito ao seu argumento, um vetor **x** de dimensão p. Vamos considerar funções g que são suaves e diferenciáveis com respeito a x.

Observação: Não existe distinção significativa entre maximização e minimização, dado que maximização de uma função é equivalente a minimizar seu negativo.

Exemplificando, suponha que desejamos desejamos maximizar

$$g(x) = \frac{\log x}{1+x}$$

com respeito a x. Nesta situação,

- Dado que a solução analítica não pode ser encontrada, recorremos a métodos iterativos que contam com sucessivas aproximações da solução;
- Estes métodos são baseados em $g'(x) = \frac{1+1/x-\log x}{(1+x)^2}$ ou alguma outra lógica.

Máxima verossimilhança

Na estimação de máxima verossimilhança, g é a função de logverossimilhança l e x é o correspondente vetor de parâmetros θ. Se $\widehat{\theta}$ é um estimador MLE, ele maximiza a log verossimilhança. Então, $\widehat{\theta}$ é a solução de uma função escore

$$l'(\theta) = \mathbf{0},$$

em que $l'(\theta) = \left(\frac{dl(\theta)}{d\theta_1}, \ldots, \frac{dl(\theta)}{d\theta_n}\right)$ e **0** é um vetor coluna de zeros.

Veja que,

- Otimização está intimamente ligada com resolver equações não lineares.
- Podemos reinterpretar como métodos para achar raízes ao invés de maximização.

4.2 Alguns métodos numéricos de maximização

Os métodos de maximização apresentados nesta seção são baseados na aproximação da função a ser maximizada a partir da série de Taylor. Iniciamos apresentado este conceito e os métodos de Newton-Raphson, Escore de Fisher e da Secante no caso univariado. Posteriormente são discutidas adaptações para o caso multivariado.

4.2.1 Série de Taylor

Suponha que f tem $n+1$ derivadas finitas em (a,b) e n derivadas contínuas em $[a,b]$. Então para algum $x_0 \in [a,b]$ distinto de x, a expansão em série de Taylor de f sobre x_0 é dada por

$$f(x) = \sum_{i=0}^{n} \frac{1}{i!} f^{(i)}(x_0)(x-x_0)^i + R_n,$$

em que $f^{(i)}(x_0)$ é a i-ésima derivada de f avaliada em x_0 e

$$R_n = \frac{1}{(n+1)!} f^{(n+1)}(\varepsilon)(x - x_0)^{n+1}$$

para algum ponto ε no intervalo entre x e x_0. Quando $|x - x_0| \to 0$, temos que $R_n = \mathbb{O}(|x - x_0|^{n+1})$.

Seja z_0 um ponto no intervalo ou um ponto limite $(-\infty$ ou $\infty)$. Seja $g(z) \neq 0$ para todo $z \neq z_0$ na vizinhança de z_0. Dizemos que $f(z) = \mathbb{O}(g(z))$ se existe uma constante M tal que

$$|f(z)| \leq M|g(z)|.$$

Por exemplo,

$$\frac{n+1}{3n^2} = \mathbb{O}(n^{-1})$$

e entende-se que estamos considerando $n \to \infty$.

4.2.2 Método de Newton-Raphson

Esta é uma abordagem extremamente rápida para encontrar raízes. Suponha que g' é continuamente diferenciável e $g''(x^*) \neq 0$. Na iteração t, a abordagem aproxima $g'(x^*)$ pela expansão linear por série de Taylor:

$$0 = g'(x^*) \approx g'(x^{(t)}) + (x^* - x^{(t)})g''(x^{(t)}).$$

Dado que g' é aproximada pela linha tangente em $x^{(t)}$, parece sensato aproximar a raiz de g' pela raiz da linha tangente. Assim, obtemos

$$x^* = x^{(t)} - \frac{g'(x^{(t)})}{g''(x^{(t)})} = x^{(t)} + h^{(t)}.$$

A equação acima descreve uma aproximação de x^* que depende do atual valor $x^{(t)}$ e um refinamento $h^{(t)}$.

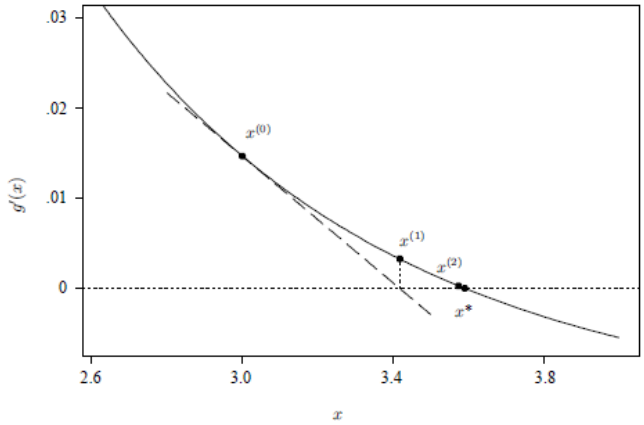

Figura 4.1: Ilustração do método de Newton-Raphson.

Iterando, a estratégia produz uma equação de atualização para o método de Newton:

$$x^{(t+1)} = x^{(t)} - \frac{g'(x^{(t)})}{g''(x^{(t)})} = x^{(t)} + h^{(t)}$$

em que $h^{(t)} = -\frac{g'(x^{(t)})}{g''(x^{(t)})}$.

A mesma atualização é obtida por resolver analiticamente o máximo da aproximação quadrática da serie de Taylor para $g(x^*)$, isto é,

$$g(x^{(t)}) + (x^* - x^{(t)})g'(x^{(t)}) + (x^* - x^{(t)})^2 g''(x^{(t)})/2.$$

A convergência do método depende da forma de g e também do valor do valor inicial $x^{(0)}$. A Figura abaixo ilustra um exemplo em que o método diverge com o seu valor inicial.

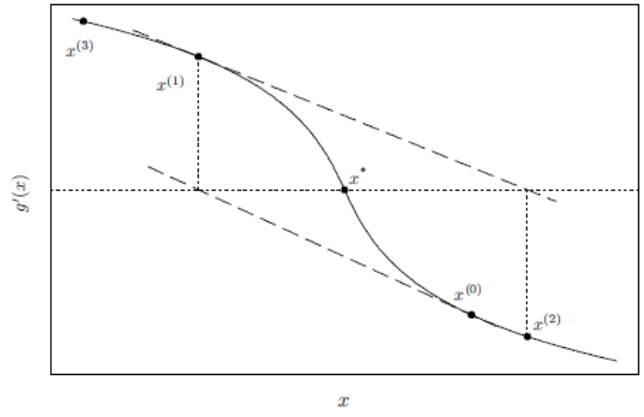

Figura 4.2: Ilustração de um situação em que não existe convergência pelo método de Newton-Raphson.

No software R, podemos implementar o método de Newton-Raphson com o seguinte código:

```
## Metodo de Newton: https://rpubs.com/aaronsc32/newton-raphson-method
# f é a função da qual desejamos encontrar a raiz
# A derivada da função que desejamos maximizar

newton.raphson <- function(f, a, b, tol = 1e-5, n = 1000, x0 = -99999) {
  require(numDeriv) # Pacote para derivadas

  if(x0 == -99999){ x0 <- a } # Definindo o valor inicial como o limite inferior
  k <- n

  # Verificando se os limites são aproximações do resultado zero
  fa <- f(a)
  if (fa == 0.0) { return(a) }
  fb <- f(b)
  if (fb == 0.0) { return(b) }

  for (i in 1:n) {
    dx <- genD(func = f, x = x0)$D[1] # Derivada de primeira ordem f'(x0)
    x1 <- x0 - (f(x0) / dx) # Calculo do próximo valor x1
    k[i] <- x1 # Guardando x1
    # Critério de parada
    if (abs(x1 - x0) < tol) {
      raiz_aprox <- tail(k, n = 1)
      res <- list('raiz' = raiz_aprox, 'iteracoes' = k)
      return(res)
    }
    x0 <- x1
  }
  print('Não houve convergência após muitas iterações')
}
```

Exemplo 4.1. Suponha que desejamos encontrar a raiz das seguintes funções:

a. $f(x) = x^2 - 10$.
b. $f(x) = x^3 - 2x - 5$.
c. $f(x) = exp(2x) - x - 6$

```r
## item a)
f <- function(x) { x^2 - 10 }
(resultado = newton.raphson(f, .5, 5, x0 = 5))
```

```
## $raiz
## [1] 3.162278
##
## $iteracoes
## [1] 3.500000 3.178571 3.162319 3.162278 3.162278
```

```r
par(mar = c(4, 4, 0, 0), mfrow = c(2, 1))
curve(f, col = 'blue', lwd = 2, from = 1, to = 4, n = 100, ylab = 'f(x)')
abline(a = 0, b = 0, lty = 5); abline(v = resultado$raiz, lty = 2, col = 2)
points(resultado$iteracoes, f(resultado$iteracoes), pch=16)
plot.ts(resultado$iteracoes, type = "b", pch = 15,
        ylab = "Estimativas", xlab = "Iterações")
abline(h = resultado$raiz,lty=2)
```

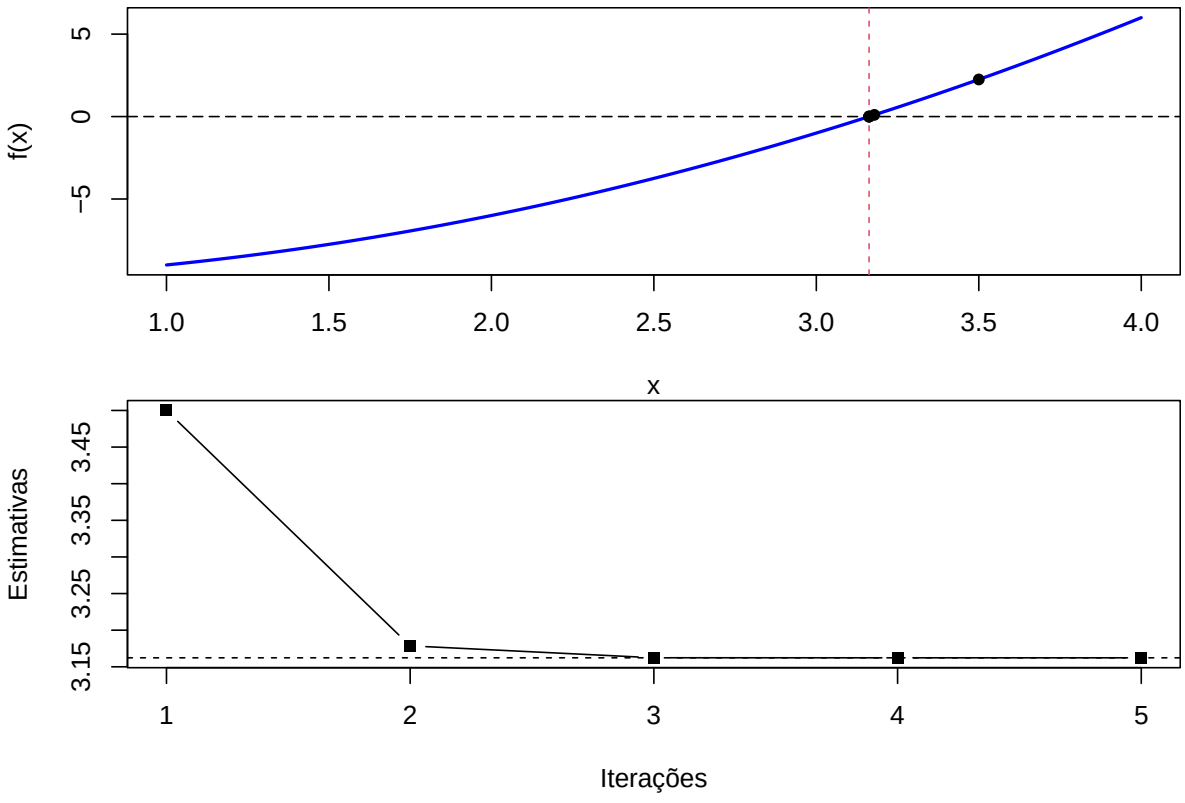

Figura 4.3: Método de Newton-Raphson no item a).

```
## item b
f <- function(x) { x^3 - 2* x - 5 }
(resultado <- newton.raphson(f, .5, 5))
```

```
## $raiz
## [1] 2.094551
##
## $iteracoes
##  [1] -4.2000000 -2.8117832 -1.8169232 -0.8851742 10.3048198  6.9290765
##  [7]  4.7196317  3.3206268  2.5170845  2.1693859  2.0975244  2.0945564
## [13]  2.0945515
```

```
par(mar = c(5, 5, 0, 0), mfrow=c(2, 1))
curve(f, from = -5, to = 5, col = 'blue', lwd = 2, lty = 2, ylab = 'f(x)')
abline(h=0,lty=2); abline(v = resultado$raiz, lty = 2, col = 2)
points(resultado$iteracoes, f(resultado$iteracoes), pch = 16)
plot.ts(resultado$iteracoes, type = "b", pch = 15,
        ylab = "Estimativas", xlab = "Iterações")
abline(h = resultado$raiz, lty = 2)
```

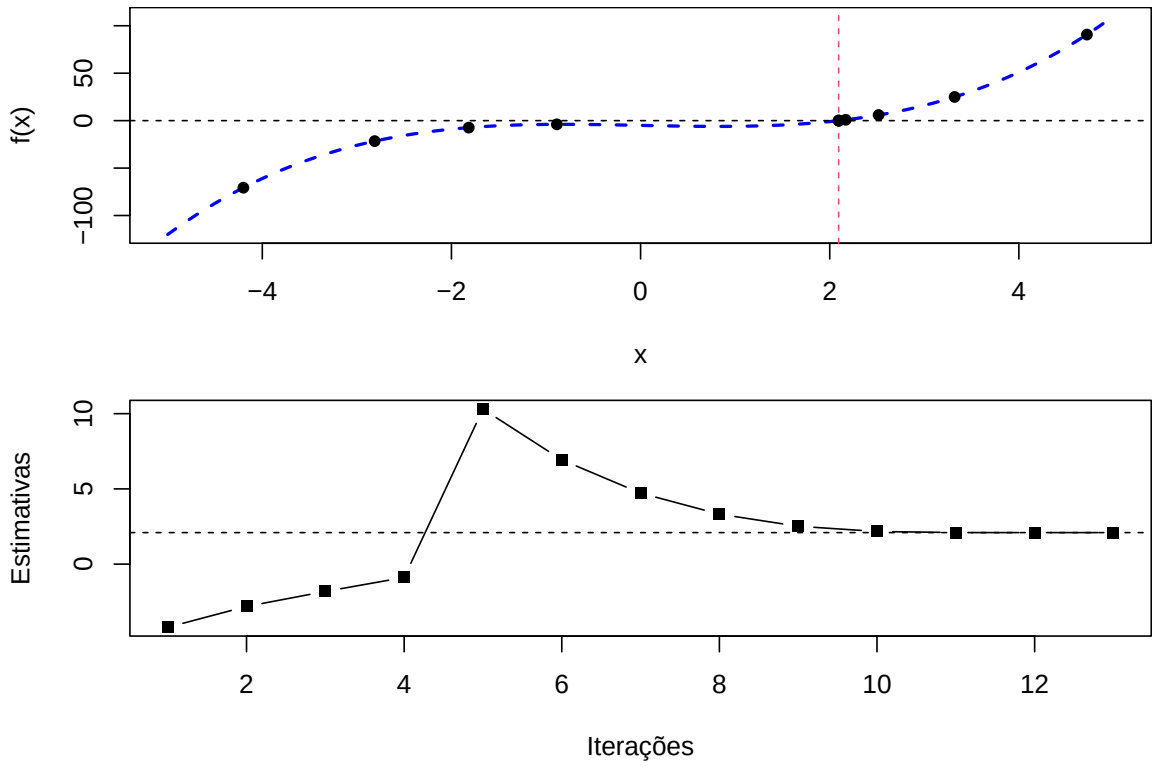

Figura 4.4: Método de Newton-Raphson no item b).

```r
## item c)
f <- function(x) { exp(2 * x) - x - 6 }
(resultado <- newton.raphson(f, .5, 2))
```

```
## $raiz
## [1] 0.97087
##
## $iteracoes
## [1] 1.3523980 1.0894844 0.9846706 0.9710729 0.9708701 0.9708700
```

```r
par(mar = c(5, 5, 0, 0), mfrow = c(2, 1))
curve(f, col = 'blue', lty = 2, lwd = 2, from = -7, to = 1.5, ylab = 'f(x)')
abline(h = 0, lty = 2); abline(v = resultado$raiz, lty = 2, col = 2)
points(resultado$iteracoes, f(resultado$iteracoes), pch = 16)
plot.ts(resultado$iteracoes, type = "b", pch = 15,
        ylab = "Estimativas", xlab = "Iterações")
abline(h = resultado$raiz, lty = 2)
```

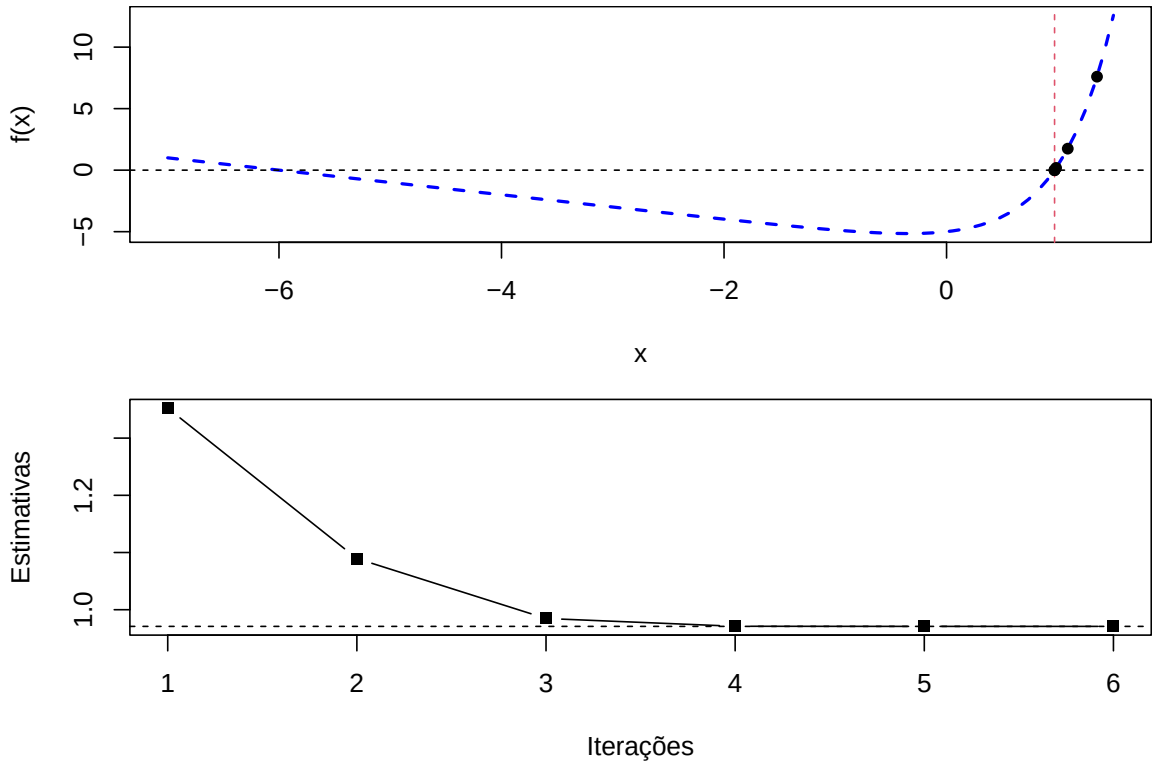

Figura 4.5: Método de Newton-Raphson no item c).

Estimador de máxima verossimilhança

Quando se trata de um problema de EMV, temos que

$$\theta^{(t+1)} = \theta^{(t)} - \frac{l'(\theta^{(t)})}{l''(\theta^{(t)})}.$$

4.2.2.1 Convergência

Se g''' é contínua e x^* é uma raiz de g', então existe uma vizinhança de x^* para o qual o método de Newton converge para x^* quando iniciado de algum $x^{(0)}$ nesta vizinhança.

De fato, quando g' duas vezes continuamente diferenciável, é convexa e tem uma raiz, então o método de Newton converge para a raiz a partir de qualquer ponto de partida.

Ao começar de algum lugar em um intervalo $[a, b]$, outro conjunto de condições que se pode verificar é o seguinte. Se

- $g''(x) \neq 0$ em $[a, b]$,
- $g'''(x)$ não muda de sinal em $[a, b]$,
- $g'(a)g'(b) < 0$, e
- $|g'(a)/g''(a)| < b - a$ e $|g'(b)/g''(b)| < b - a$,

então o método de Newton convergirá de qualquer $x^{(0)}$ no intervalo.

Ordem de Convergência

Um método tem ordem de convergência de β se $\lim_{t\to\infty} \epsilon^{(t)} = 0$ e

$$\lim_{t\to\infty} \frac{|\epsilon^{(t+1)}|}{|\epsilon^{(t)}|^\beta} = c,$$

para $\epsilon^{(t)} = x^{(t)} - x^*$ e alguma constante $c \neq 0$ e $\beta > 0$.

O método de Newton tem ordem de convergência quadrática ($\beta = 2$).

4.2.3 Escore de Fisher

A informação de Fisher esperada $I(\theta)$ pode ser aproximada pela informação de Fisher observada $-l''(\theta)$. Então, quando a otimização corresponde a um problema de EMV, é razoável substituir $-l''(\theta)$ por $I(\theta)$. A equação de atualização é dada por

$$\theta^{(t+1)} = \theta^{(t)} + \frac{l'(\theta^{(t)})}{I(\theta^{(t)})},$$

em que $I(\theta^{(t)})$ é a informação de Fisher esperada avaliada em $\theta^{(t)}$.

Observações:

- O escore de Fisher e o método de Newton-Raphson tem as mesmas propriedades assintóticas, mas para alguns problemas um deles pode ser computacionalmente ou analiticamente mais fácil do que o outro;
- O escore de Fisher trabalha melhor no início por fazer melhorias rápidas, enquanto que o método de Newton trabalha melhor para refinamento próximo do fim.

4.2.4 Método da Secante

Se no método de Newton-Raphson o cálculo da segunda derivada $g''(x$ é muito complicado, ele pode ser substituído por sua aproximação

$$\frac{g'(x^{(t)}) - g'(x^{(t-1)})}{x^{(t)} - x^{(t-1)}}.$$

O resultado é o método da secante que tem equação de atualização

$$x^{(t+1)} = x^{(t)} - g'(x^{(t)})\frac{x^{(t)} - x^{(t-1)}}{g'(x^{(t)}) - g'(x^{(t-1)})},$$

para $t \geq 1$. Esta abordagem requer dois pontos iniciais $x^{(0)}$ e $x^{(1)}$.

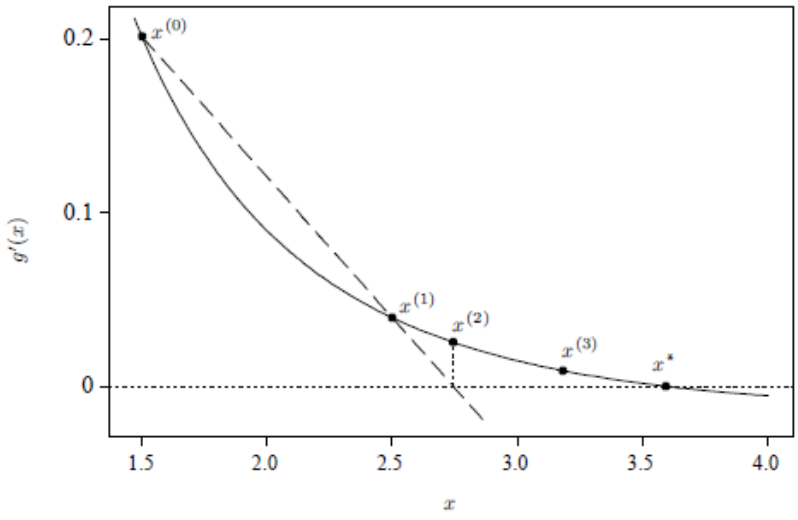

Figura 4.6: Ilustração do método da secante.

O método da secante tem uma ordem de convergência de $\beta \approx 1,62$.

No software R podemos implementar o método de Newton-Raphson com o seguinte código:

```
## Método de secante
## Fonte: https://rpubs.com/aaronsc32/secant-method-r

metodo_secante <- function(f, x0, x1, tol = 1e-9, n = 500) {
  k <- x1 # Guardando x1
  for (i in 1:n) {
    # Calculo do novo valor de x
    x2 <- x1 - f(x1)/( (f(x1) - f(x0))/(x1 - x0) )
    k[i+1] <- x2
    # Criterio de convergencia
    if (abs(x2 - x1) < tol) {
      raiz_aprox <- tail(k, n=1)
      res <- list('raiz' = raiz_aprox, 'iteracoes' = k)
      return(res)
    }
    # Se a raiz não foi obtida, atualize os valores e continue.
    x0 <- x1
    x1 <- x2
  }
}
```

Exemplo 4.2. Suponha que desejamos encontrar a raiz das seguintes funções:

a. $f(x) = \cos x$.
b. $f(x) = x^3 - 2x - 5$.
c. $f(x) = exp(2x) - x - 6$

```
## item a)
f <- function(x) { cos(x) }
x0 <- 0; x1 <- 3
a <- cos(x0); b <- cos(x1)

par(mar = c(5, 5, 0, 0), mfrow = c(1, 3))
curve(expr = f, col = 'blue', lwd = 2, lty = 2, xlim = c(-0.5, 3.5))
```

```
points(x0, a, cex = 1.5, pch = 16);   points(x1, b, cex = 1.5, pch = 16)
segments(x0, a, x1, b, col = 'red', lwd = 2); abline(h = 0)

x1 <- 3; x2 <- 1.507543
a <- cos(x1); b <- cos(x2)

curve(expr = f, col = 'blue', lwd = 2, lty = 2, xlim = c(-0.5, 3.5))
points(x1, a, cex = 1.5, pch = 16);   points(x2, b, cex = 1.5, pch = 16)
segments(x1, a, x2, b, col = 'red', lwd = 2); abline(h = 0)

x2 <- 1.507543; x3 <- 1.597117
a <- cos(x2); b <- cos(x3)

curve(expr = f, col = 'blue', lwd = 2, lty = 2, xlim = c(-0.5, 3.5))
points(x2, a, cex = 1.5, pch = 16);  points(x3, b, cex = 1.5, pch = 16)
segments(x2, a, x3, b, col = 'red', lwd = 2); abline(h = 0)
```

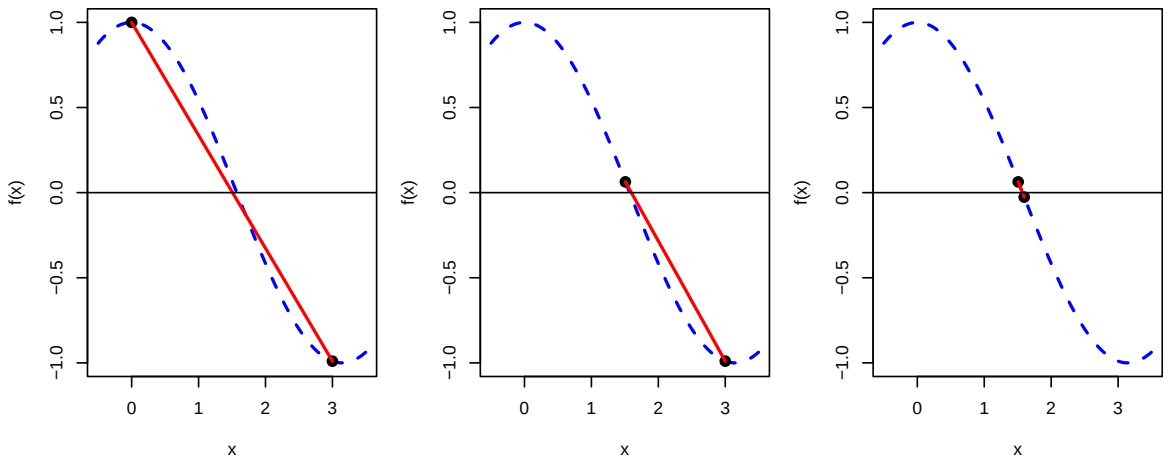

Figura 4.7: Método da secante no item a).

```
# Rodando o método da secante
metodo_secante(f, 1, 3)
```

```
## $raiz
## [1] 1.570796
##
## $iteracoes
## [1] 3.000000 1.706141 1.501965 1.570900 1.570796 1.570796 1.570796
```

```r
## item b)
f <- function(x) { x^3 - 2* x - 5 }

par(mar = c(5, 5, 0, 0), mfrow = c(2, 1))
curve(f, from = -5, to = 5, col = 'blue', lwd = 2, lty = 2, ylab = 'f(x)')
abline(h = 0, lty = 2)

resultado = metodo_secante(f, .5, 1.5)
abline(v = resultado$raiz, lty = 2, col = 2)
points(resultado$iteracoes, f(resultado$iteracoes), pch = 16)

plot.ts(resultado$iteracoes, type = "b", pch = 15,
        ylab = "Estimativas", xlab = "Iterações")
abline(h = resultado$raiz, lty = 2)
```

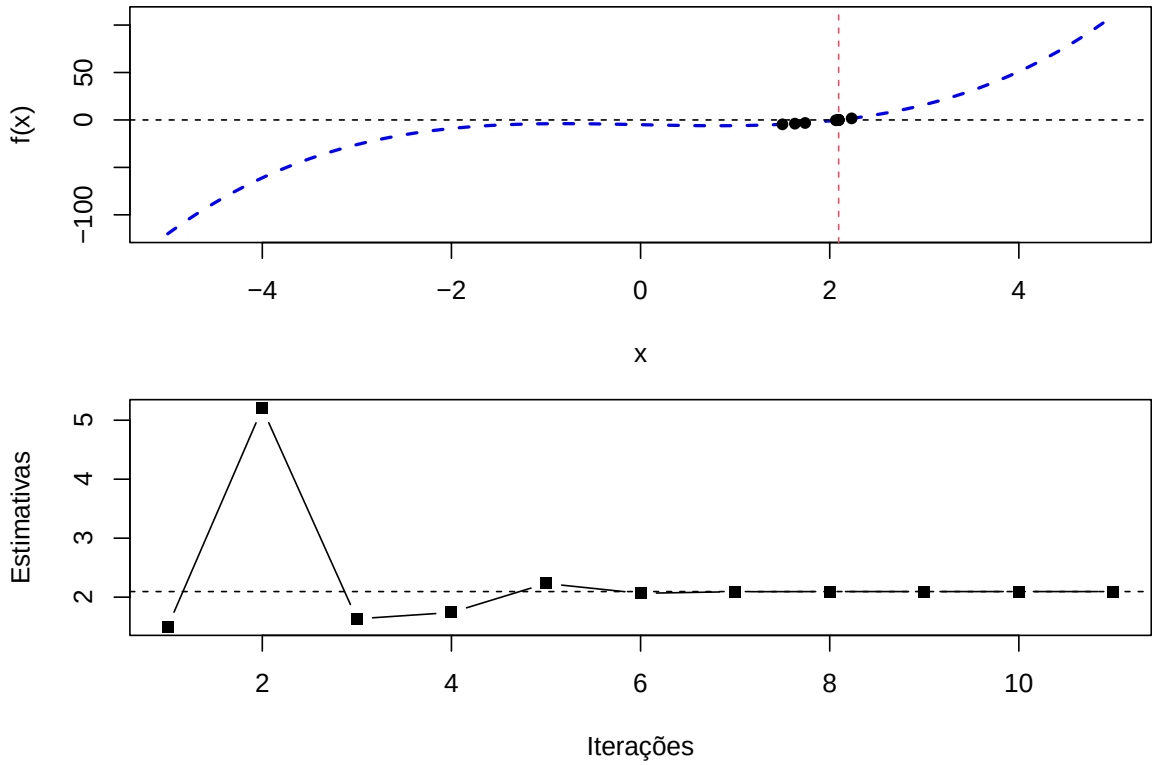

Figura 4.8: Método da secante no item b).

```r
## item c)
f <- function(x) { exp(2 * x) - x - 6 }
```

```
par(mar = c(5, 5, 0, 0), mfrow = c(2, 1))
curve(f, col = 'blue', lty = 2, lwd = 2, from = -5, to = 1.5, ylab = 'f(x)')
abline(h = 0, lty = 2)

resultado = metodo_secante(f, 0.5, 1.5)
abline(v = resultado$raiz, lty = 2, col = 2)
points(resultado$iteracoes, f(resultado$iteracoes), pch = 16)

plot.ts(resultado$iteracoes, type = "b", pch = 15,
        ylab = "Estimativas", xlab = "Iterações")
abline(h = resultado$raiz, lty = 2)
```

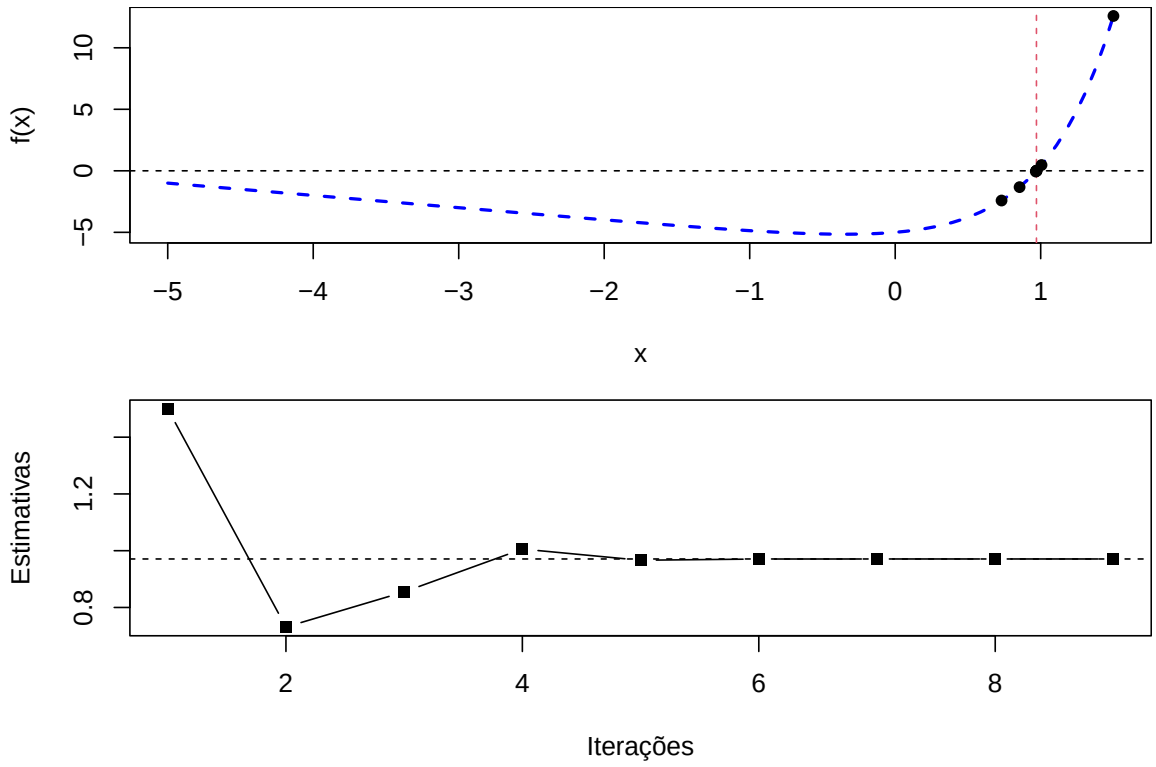

Figura 4.9: Método da secante no item c).

4.2.5 Regra de parada

Na prática, para o procedimento não rodar indefinidamente, necessitamos de uma regra de parada baseada em algum critério de convergência. A cada iteração, a regra de parada deve ser checada e quando atendida, o valor atual $x^{(t+1)}$ é a solução.

Existem duas razões para parar o algoritmo:

- Se o procedimento aparenta ter atingido satisfatória convergência;
- Se parece improvável que isso aconteça em breve.

Ideias por trás dos critérios:

- Uma regra baseada diretamente em $g'(x^{(t+1)})$ não é confiável, pois grande mudanças de $x^{(t)}$ para $x^{(t+1)}$ podem ocorrer quando $g'(x^{(t+1)})$ é muito pequena;
- Pequenas mudanças de $x^{(t)}$ para $x^{(t+1)}$ é mais frequentemente associado com $g'(x^{(t+1)})$ perto de zero.

Então, avaliamos a convergência por monitorar $|x^{(t+1)} - x^{(t)}|$ e usar $g'(x^{(t+1)})$ como uma verificação de segurança. Podemos considerar os seguintes critérios:

- O critério de convergência absoluta define a parada quando

$$|x^{(t+1)} - x^{(t)}| < \epsilon,$$

  em que ϵ é uma constante que indica a imprecisão tolerada.

- O critério de convergência relativa define a parada quando

$$\frac{|x^{(t+1)} - x^{(t)}|}{|x^{(t)}|} < \epsilon \text{ ou } \frac{|x^{(t+1)} - x^{(t)}|}{|x^{(t)}| + \epsilon} < \epsilon,$$

  em que a segunda fórmula deve ser considerara para casos em que $x^{(t)}$ está perto do zero

É importante incluir regras de parada que sinalizam uma falha na convergência. Neste caso podemos citar:

- Uma regra simples é parar depois de N iterações sem convergência. Também podemos parar se uma ou mais medidas de convergência falhar em decrescer ao longo de várias iterações.
- Também é sensato parar se o procedimento parece estar convergindo para um ponto em que $g(x)$ é inferior a outro valor que você já encontrou.

Valor inicial

Outro aspecto a ser pensado é o valor inicial definido no procedimento de maximização. Podemos dizer que:

- Em geral, um valor inicial ruim pode levar à divergência, a um ciclo, à descoberta de um máximo local enganoso ou a um mínimo local ou a outros problemas;
- O resultado depende de g, do valor inicial e o algoritmo de otimização utilizado;
- Em geral, ajuda a começar bem perto do máximo global, contanto que g não seja virtualmente plano na vizinhança que contém $x^{(0)}$ e x;
- Os métodos para gerar valores iniciais razoáveis incluem gráficos, estimativas preliminares (por exemplo, estimativas do método de momentos) e tentativa e erro;
- Usar uma coleção de execuções a partir de múltiplos valores iniciais pode ser uma maneira eficaz de ganhar confiança em seu resultado e evitar ser enganado por máximos locais ou bloqueado por falhas de convergência.

4.2.6 Problemas multivariados

Em um problema de otimização multivariada, procuramos o ponto de máximo de uma função g de um um vetor $\mathbf{x} = (x_1, \ldots, x_p)^T$ de dimensão $p \times 1$. Na iteração t, denotamos as estimativas do ponto de máximo por $\mathbf{x}^{(t)} = (x_1^{(t)}, \ldots, x_p^{(t)})^T$. Muitos dos princípios gerais discutidos no caso univariado também são aplicados no caso multivariado.

4.2.6.1 Método de Newton e Escore de Fisher

Aproximamos $g(\mathbf{x}^*)$ por sua expansão quadrática em série de Taylor:

$$g(\mathbf{x}^*) \approx g(\mathbf{x}^{(t)}) + (\mathbf{x}^* - \mathbf{x}^{(t)})g'(\mathbf{x}^{(t)}) + \frac{(\mathbf{x}^* - \mathbf{x}^{(t)})^2}{2} g''(\mathbf{x}^{(t)}).$$

Maximizando esta função com respeito a $\mathbf{x}^*$ obtemos

$$g'(\mathbf{x}^{(t)}) + (\mathbf{x}^* - \mathbf{x}^{(t)})g''(\mathbf{x}^{(t)}) = 0.$$

Desta forma,

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - g'' \left(\mathbf{x}^{(t)}\right)^{-1} g' \left(\mathbf{x}^{(t)}\right).$$

Como no caso univariado, em problemas de EMV, temos

$$\theta^{(t+1)} = \theta^{(t)} - l''(\theta^{(t)})^{-1} l'(\theta^{(t)}).$$

e podemos substituir a informação de Fisher observada de $\theta^{(t)}$ por $I(\theta^{(t)})$, obtendo

$$\theta^{(t+1)} = \theta^{(t)} + I(\theta^{(t)})^{-1} l'(\theta^{(t)}).$$

O método de Newton é assintoticamente equivalente ao escore de Fisher.

Regras de parada

Os critérios de parada são similares, mas é necessário algumas adaptações.

Seja $D(\mathbf{u}, \mathbf{v})$ uma medida de distância entre vetores. Duas escolhas possíveis são:

$$D(\mathbf{u}, \mathbf{v}) = \sum_{i=1}^{p} |u_i - v_i| \text{ ou } D(\mathbf{u}, \mathbf{v}) = \sum_{i=1}^{p} \sqrt{(u_i - v_i)^2}.$$

Critérios de convergência absoluta ou relativa são dados por

$$D(\mathbf{x}^{(t+1)}, \mathbf{x}^{(t)}) < \epsilon \quad \text{ou} \quad \frac{D(\mathbf{x}^{(t+1)}, \mathbf{x}^{(t)})}{D(\mathbf{x}^{(t)}, \mathbf{0})} < \epsilon \quad \text{ou} \quad \frac{D(\mathbf{x}^{(t+1)}, \mathbf{x}^{(t)})}{D(\mathbf{x}^{(t)}, \mathbf{0}) + \epsilon} < \epsilon.$$

No R, o método de Newton-Raphson multivariado pode ser implementado pela função abaixo.

```
# Método de Newton multivariado
newton <- function(f, x0, tol = 1e-9, n.max = 100,...) {
  # f é uma função que dado x retorna uma lista com:
  # {f(x), gradiente de f(x), Hessiana de f(x)}, para alguma f
  x <- x0
  f.x <- f(x, dados)
  n <- 0
  while ( ( max(abs(f.x[[2]]) ) > tol  ) & (n < n.max) ) {
    x <- x - solve(f.x[[3]], f.x[[2]])
    f.x <- f(x, dados)
    n <- n + 1
  }
  if (n == n.max) {
    cat('Convergência falhou!\n')
  } else {
    return(x)
  }
}
```

Exemplo 4.3. Como ilustração no método de Newton-Raphson no contexto multivariado, iremos considerar o modelo de regressão linear em que y_i são independentes tal que

$$y_i \sim Bernoulli(p_i), \text{ para } i = 1, \ldots, n,$$

em que $p_i = \frac{\exp\{\beta_0+\beta_1 x_i\}}{1+\exp\{\beta_0+\beta_1 x_i\}}$.

A função de verossimilhança é dada por

$$\begin{aligned} L(\theta \mid \mathbf{y}) &= \prod_{i=1}^{n} \left[\frac{\exp\{\beta_0+\beta_1 x_i\}}{1+\exp\{\beta_0+\beta_1 x_i\}}\right]^{y_i} \left[\frac{1}{1+\exp\{\beta_0+\beta_1 x_i\}}\right]^{1-y_i} \\ &= \prod_{i=1}^{n} \left[\exp\{\beta_0+\beta_1 x_i\}\right]^{y_i} \left[1+\exp\{\beta_0+\beta_1 x_i\}\right]^{-1} \end{aligned}$$

em que $\theta = (\beta_0, \beta_1)$.

Consequentemente, a função de log-verossimilhança é dada por

$$l(\theta \mid \mathbf{y}) = \sum_{i=1}^{n} y_i (\beta_0+\beta_1 x_i) - \sum_{i=1}^{n} \log\{1+\exp\{\beta_0+\beta_1 x_i\}\}.$$

Assim, temos que

$$l'(\theta \mid \mathbf{y}) = \left(\sum_{i=1}^{n}(y_i-p_i);\ \sum_{i=1}^{n} x_i(y_i-p_i)\right)$$

e

$$l''(\theta \mid \mathbf{y}) = \begin{pmatrix} -\sum_{i=1}^{n} p_i(1-p_i) & -\sum_{i=1}^{n} p_i(1-p_i)x_i \\ -\sum_{i=1}^{n} p_i(1-p_i)x_i & -\sum_{i=1}^{n} p_i(1-p_i)x_i^2 \end{pmatrix},$$

em que $p_i = \frac{\exp\{\beta_0+\beta_1 x_i\}}{1+\exp\{\beta_0+\beta_1 x_i\}}$.

Assim, no R temos o código abaixo.

```
logvero <- function(betas, dados) {
  beta0 <- betas[1];   beta1 <- betas[2];
  y <- dados[,1];   x <- dados[,2]
  pred_linear <- beta0 + beta1*x
  lvero <- t(y)%*%pred_linear - sum(log(1+exp(pred_linear)))
  return(lvero)
}

## Gerando uma amostra
require(boot)
n <- 500
betas <- c(0.5, 2)
covariavel <- data.frame(x = rnorm(n))
X <- model.matrix(~x+1, covariavel)
preditor_linear <- X %*% betas
prob <- inv.logit(preditor_linear)
y <- rbinom(n, 1, prob)
dados <- data.frame(y = y, covariavel)
```

```
## Plotando o gráfico da função de verossimilhança
b0 <- seq(-2, 4, .1)
b1 <- seq(1, 4, .1)
yb0b1 <- data.frame(matrix(0, length(b0)*length(b1), 3))
names(yb0b1) <- c('b0', 'b1', 'lvero')
n <- 0
for (i in 1:length(b0)) {
  for (j in 1:length(b1)) {
    n <- n + 1
    yb0b1[n,] <- c(b0[i], b1[j], logvero(c(b0[i], b1[j]), dados))
  }
}

library(lattice)
require(RColorBrewer)
colr <- brewer.pal(11, "Spectral")
colr <- colorRampPalette(colr, space = "rgb")
wireframe(lvero ~ b0*b1, data = yb0b1, scales = list(arrows = FALSE),
                zlab = 'l', drape = T, col.regions = colr(100))
```

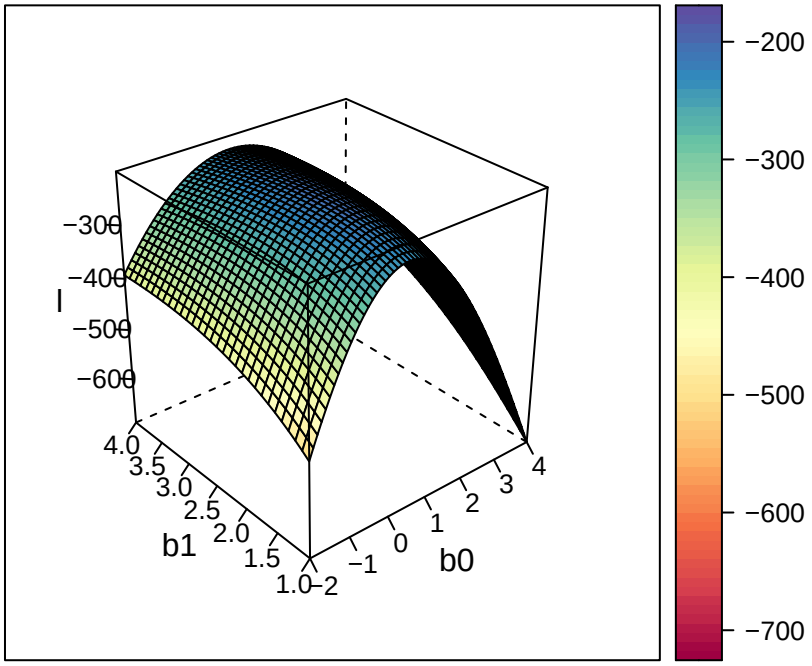

Figura 4.10: Função de logverossimilhança do modelo de regressão logística.

Para utilizar o algoritmo de Newtown-Raphson precisamos de uma função que calcule a função, o gradiente e a matriz Hessiana para cada ponto.

```
## Funcao com a funcao para maximizar, a primeira derivada e a matriz hessiana
lvero <- function(betas, dados) {
  beta0 <- betas[1];   beta1 <- betas[2]
  y_obs <- dados[,1];   x_obs <- dados[,2]
  pred_linear <- beta0 + beta1*x_obs
  p <- inv.logit(pred_linear)
  f <- t(y_obs)%*%pred_linear - sum(log(1+exp(pred_linear)))
  f1 <- sum(y_obs-p)
  f2 <- t(x_obs)%*%(y_obs-p)
  f11 <- -sum(p*(1-p))
  f12 <- -t(x_obs)%*%(p*(1-p))
  f22 <- -t(x_obs^2)%*%(p*(1-p))
  return(list(f, c(f1, f2), matrix(c(f11, f12, f12, f22), 2, 2)))
}
```

```
## partindo de diferentes valores iniciais
for (b0 in seq(-1, 1.6, .9)) {
  for (b1 in seq(0.4, 3, .9)) {
    cat(c(b0, b1), '-->', newton(lvero, c(b0, b1)), '\n')
  }
}
```

```
## -1 0.4 --> 0.5398808 2.292881
## -1 1.3 --> 0.5398808 2.292881
## -1 2.2 --> 0.5398808 2.292881
## -0.1 0.4 --> 0.5398808 2.292881
## -0.1 1.3 --> 0.5398808 2.292881
## -0.1 2.2 --> 0.5398808 2.292881
## 0.8 0.4 --> 0.5398808 2.292881
## 0.8 1.3 --> 0.5398808 2.292881
## 0.8 2.2 --> 0.5398808 2.292881
```

4.2.7 Exercícios

Exercício 4.1. Adapte a função newton.raphson no script4 para também receber como argumento a função f' teórica. Compare o tempo computacional para a execução da função original e da adaptação em algum exemplo de sua preferência.

Exercício 4.2. Os seguintes dados são uma amostra iid de uma distribuição Cauchy(θ, 1): 1.77, −0.23, 2.76, 3.80, 3.47, 56.75, −1.34, 4.24, −2.44, 3.29, 3.71, −2.40, 4.53, −0.07, −1.05, −13.87, −2.53, −1.75, 0.27, 43.21.

a) Desenhe o gráfico da função de log-verossimilhança. Ache o EMV para θ usando o método de Newton-Raphson. Tente todos os seguintes pontos iniciais: −11, −1, 0, 1.5, 4, 4.7, 7, 8, e 38. Discuta os resultados. A média amostral é um bom ponto inicial?

b) Com os valores iniciais $(\theta^{(0)}, \theta^{(1)}) = (-2, -1)$, aplique o método da secante para estimar θ. O que acontece quando $(\theta^{(0)}, \theta^{(1)}) = (-3, 3)$ e para outras escolhas iniciais?

Exercício 4.3. Implemente o algoritmo Escore de Fisher para estimar θ no modelo $X \sim Beta(\theta, 2)$ a partir de uma amostra de tamanho n. Ilustre o funcionamento deste método com dados simulados do modelo teórico.

4.3 Algoritmo EM

O algoritmo EM (*expectation-maximization*) é uma estratégia de maximização baseada na ideia de dados faltantes e considera a distribuição do que está faltando dado o que foi observado. Ressaltamos os seguintes aspectos:

- O artigo seminal de Dempster et al. (1977) apresenta os fundamentos estatísticos desta estratégia e a eficácia em uma variedade de problemas estatísticos.
- A popularidade do algoritmo EM origina-se de que pode ser simples de implementar e com que confiabilidade consegue encontrar o máximo global através de passos estáveis e crescentes;
- No contexto Bayesiano, o algoritmo EM pode ser utilizado para encontrar a moda da distribuição *a posteriori*.

Suponha que temos dados observados gerados a partir de variáveis aleatórias **X**, juntamente com dados faltantes (ou não observados) de variáveis aleatórias **Z**. Condicional nos dados observados **x**, queremos maximizar uma função de verossimilhança $L(\theta|\mathbf{x})$. Definimos por dados completos os dados gerados de $\mathbf{Y} = (\mathbf{X}, \mathbf{Z})$. No entanto, pode ser mais fácil trabalhar com as densidades de $\mathbf{Y}|\theta$ e $\mathbf{Z}|(\mathbf{x}, \theta)$.

Observação: Os dados faltantes podem não ser verdadeiros, sendo apenas uma manobra conceitual que simplifica o problema. Neste caso, **Z** é frequentemente referenciado como *variáveis latentes*.

4.3.1 Dados faltantes, Marginalização e Notação

Se **Z** é considerado latente ou ausente, ele pode ser visto como tendo sido removido do **Y** completo por meio da aplicação de uma função $\mathbf{X} = M(\mathbf{Y})$.

Sejam $f_{\mathbf{X}}(\mathbf{x}|\theta)$ e $f_{\mathbf{Y}}(\mathbf{y}|\theta)$ denotam as densidades dos dados observados e os dados completos, respectivamente. A suposição de dados latentes ou ausentes equivale ao modelo de uma marginalização no qual observamos que o **X** possui densidade

$$f_{\mathbf{X}}(\mathbf{x}|\theta) = \int_{\{\mathbf{y}:M(\mathbf{y})=\mathbf{x}\}} f_{\mathbf{Y}}(\mathbf{y}|\theta)dy.$$

Note que a densidade condicional dos dados faltantes dado os dados observados é dada por

$$f_{\mathbf{Z}|\mathbf{X}}(\mathbf{z}|\mathbf{x}, \theta) = f_{\mathbf{Y}}(\mathbf{y}|\theta)/f_{\mathbf{X}}(\mathbf{x}|\theta).$$

4.3.2 O algoritmo

O algoritmo EM visa, iterativamente, maximizar $L(\theta|\mathbf{x})$ com respeito a θ. Seja $\theta^{(t)}$ a estimativa de máximo na iteração t, para $t = 0, 1, ...$ Defina $Q(\theta|\theta^{(t)})$ a esperança da função de log-verossimilhança para dados completos, condicional aos dados observados $\mathbf{X} = \mathbf{x}$. Isto é,

$$\begin{aligned} Q(\theta|\theta^{(t)}) &= E[\log L(\theta|\mathbf{Y})|\mathbf{x}, \theta^{(t)}] \\ &= \int \log f_{\mathbf{Y}}(\mathbf{y}|\theta) f_{\mathbf{Z}|\mathbf{X}}(\mathbf{z}|\mathbf{x}, \theta^{(t)}) d\mathbf{z}. \end{aligned}$$

O algoritmo é inicializado com $\theta^{(0)}$ e então altera entre passos de esperança e maximização. O algoritmo pode ser descrito como:

Algoritmo

- Iniciar $\theta^{(0)}$ e fazer $t = 0$;
- **Passo E:** Calcular $Q(\theta|\theta^{(t)})$;
- **Passo M:** Obter $\theta^{(t+1)}$ ao maximizar $Q(\theta|\theta^{(t)})$ com respeito a θ e fazer $t = t + 1$;
- Repetir passos E e M até obter convergência para a sequência de estimativas $\theta^{(t)}$, segundo algum critério de parada.

Podemos considerar os critérios citados anteriormente nos métodos de maximização, usualmente com

$$(\theta^{(t+1)} - \theta^{(t)})^T (\theta^{(t+1)} - \theta^{(t)})$$

ou

$$|Q(\theta^{(t+1)}|\theta^{(t)}) - Q(\theta^{(t)}|\theta^{(t)})|.$$

Outra regra muito usada é parar se

$$\max_r \left(\frac{|\theta_r^{(t+1)} - \theta_r^{(t)}|}{|\theta_r^{(t)}| + \delta_1} \right) < \delta_2,$$

em que δ_1 e δ_2 são constantes pré-determinadas (por exemplo, $\delta_1 = 0,001$ e $\delta_2 = 0,0001$) e $\max_r(A_r)$ representa o maior valor A_r com o subscrito r tal que $r = 1, ..., p$.

Exemplo 4.4. (Modelo Probito)
Suponha uma amostra aleatória simples de X_i, $i = 1, \dots, n$, tal que

$$X_i \sim Bernoulli(p_i),$$

em que $p_i = \Phi(\beta_0 + w_i\beta_1)$ e w_i é a covariável observada o indivíduo i. Note que para estimar tal modelo para um conjunto de dados é necessário maximizar numericamente uma expressão que irá envolver a inversa da distribuição acumulada da normal padrão.

Podemos utilizar o algoritmo EM reescrevendo tal que

$$X_i = \begin{cases} 1 & \text{se } Z_i \geq 0, \\ 0 & \text{caso contrário,} \end{cases}$$

tal que $Z_i = \beta_0 + \beta_1 w_i + \epsilon_i$ e $\epsilon_i \sim N(0, 1)$. Note que, condicional em Z_i, a variável X_i tem distribuição degenerada em 0 ou em 1, isto é, assume um desses valores com probabilidade igual a 1. Desta forma, podemos escrever

$$f_{X_i|Z_i}(x_i|z_i) = I\{z_i \geq 0\}I\{x_i = 1\} + I\{z_i < 0\}I\{x_i = 0\}.$$

Além disso, $f_{Z_i}(z_i|\theta) = \phi(z_i|\mu_{z_i}, 1)$, em que $\theta = (\beta_0, \beta_1)$, $\mu_{z_i} = \beta_0 + \beta_1 w_i$ e $\phi(\cdot|a, b)$ é função de densidade da distribuição normal com média a e variância b. Logo,

$$\begin{aligned} f_{Y_i}(y_i|\theta) &= f_{X_i|Z_i}(x_i|z_i) f_{Z_i}(z_i|\theta) \\ &= \left[I\{z_i \geq 0\}I\{x_i = 1\} + I\{z_i < 0\}I\{x_i = 0\}\right] \phi(z_i|\mu_{z_i}, 1). \end{aligned}$$

Necessitamos obter a distribuição de $Z_i \mid X_i$ para calcular a função $Q(\theta|\theta^{(t)})$. Temos que

$$f_{Z_i|X_i}(z_i|x_i, \theta) = \frac{f_{X_i|Z_i}(x_i|z_i) f_{Z_i}(z_i|\theta)}{\int f_{X_i|Z_i}(x_i|z_i) f_{Z_i}(z_i|\theta) dz_i} = \begin{cases} \frac{\phi(z_i|\mu_{z_i},1)}{1-\Phi(0|\mu_{z_i},1)} I\{z_i \geq 0\} & \text{se } x_i = 1; \\ \frac{\phi(z_i|\mu_{z_i},1)}{\Phi(0|\mu_{z_i},1)} I\{z_i < 0\} & \text{se } x_i = 0. \end{cases}$$

Note que $Z_i|X_i = x_i$ tem distribuição normal truncada com parâmetro de locação μ_{z_i} e parâmetro de escala igual a 1, colocando massa de probabilidade em valores de Z_i maiores o iguais a 0 se $x_i = 1$ e menores do que 0 se $x_i = 0$.

Assim, visto que

$$\begin{aligned}
\log f_{\mathbf{Y}}(\mathbf{y}|\theta) &= \log\left\{\prod_{i=1}^{n} f_{Y_i}(y_i|\theta)\right\} \\
&= \sum_{i=1}^{n}\left[\log f_{X_i|Z_i}(x_i|z_i) + \log f_{Z_i}(z_i|\theta)\right] \\
&= \sum_{i=1}^{n}\log\{I\{z_i \geq 0\}I\{x_i = 1\} + I\{z_i < 0\}I\{x_i = 0\}\} \\
&\quad + \sum_{i=1}^{n}\left[-\frac{\log 2\pi}{2} - \frac{(z_i - \beta_0 - \beta_1 w_i)^2}{2}\right],
\end{aligned}$$

segue que

$$\begin{aligned}
Q(\theta|\theta^{(t)}) &= E[\log f_{\mathbf{Y}}(\mathbf{y}|\theta) \mid \mathbf{x}, \theta^{(t)}] \\
&= -\frac{1}{2}\sum_{i=1}^{n}\hat{Z}_i^{2(t)} + \sum_{i=1}^{n}\hat{Z}_i^{(t)}(\beta_0 + \beta_1 w_i) - \frac{1}{2}\sum_{i=1}^{n}(\beta_0 + \beta_1 w_i)^2 + C, \\
&= -\frac{1}{2}E[\mathbf{Z}^T\mathbf{Z}|\mathbf{x}, \theta^{(t)}] + \beta^T\mathbf{W}^T E[\mathbf{Z}|\mathbf{x}, \theta^{(t)}] - \frac{1}{2}(\mathbf{W}\beta)^T(\mathbf{W}\beta) + C,
\end{aligned}$$

em que $\hat{Z}_i^{(t)} = E[Z_i|x_i, \theta^{(t)}]$, $\hat{Z}_i^{2(t)} = E[Z_i^2|x_i, \theta^{(t)}]$, C é uma constante com respeito aos parâmetros, $\beta^T = (\beta_0, \beta_1)$, $\mathbf{Z}^T = (Z_1, \ldots, Z_n)$ e $\mathbf{W} = (\mathbf{1}_n, (w_1, \ldots, w_n)^T)$.

O próximo passo é maximizar a função $Q(\theta|\theta^{(t)})$ com respeito a θ. Para isso encontramos a seguinte derivada,

$$\frac{Q(\theta|\theta^{(t)})}{d\beta} = \mathbf{W}^T E[\mathbf{Z}|\mathbf{x}, \theta^{(t)}] - \mathbf{W}^T\mathbf{W}\beta.$$

Disto, obtemos

$$\beta^{(t+1)} = (\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T E[\mathbf{Z}|\mathbf{x}, \theta^{(t)}],$$

tal que

$$\hat{Z}_i^{(t)} = E[Z_i|\mathbf{x}, \theta^{(t)}] = \begin{cases} W_i^T\beta^{(t)} - \frac{\phi(W_i^T\beta^{(t)})}{\Phi(-W_i^T\beta^{(t)})} & \text{se } x_i = 0; \\ W_i^T\beta^{(t)} + \frac{\phi(W_i^T\beta^{(t)})}{1-\Phi(-W_i^T\beta^{(t)})} & \text{se } x_i = 1, \end{cases}$$

com $W_i^T = (1, w_i)$. O resultado sobre a esperança condicional pode obtido utilizando resultados conhecidos sobre a distribuição normal truncada.

Com isso, neste exemplo o algoritmo EM é dado a seguir.

Algoritmo

- Iniciar $\beta^{(0)} = (\beta_0^{(0)}, \beta_1^{(0)})$ e faça $t = 0$;
- **Passo E:** Calcular $E[\mathbf{Z}|\mathbf{x}, \theta^{(t)}]$;
- **Passo M:** Calcular $\beta^{(t+1)}$ e fazer $t = t + 1$;
- Calcular um critério de parada. Se tolerância for atendida faça $\widehat{\beta} = \beta^{(t+1)}$. Caso contrário, incremente t e volte ao passo 2.

No R temos:

```
## Funcao para gerar dados do modelo probito
gera_dados <- function(n, w, betas){
  prob <- pnorm(betas[1] + betas[2]*w)
  x <- rbinom(n, 1, prob)
  return( data.frame(x=x, w=w) )
}

## Funcao para ajustar o modelo usando um loop desnecessario
EM_probito1 <- function(Dados, inicial, max_itera = 200, tol = 0.001){
  n <- dim(Dados)[1]
  W <- cbind(rep(1, n), Dados$w)
  WW <- solve( t(W) %*% W ) %*% t(W)
  betas <- inicial
  estimativas <- matrix(0, ncol = 2, nrow = max_itera)
  estimativas[1,] <- betas
  cont <- 1;   crit <- 1; Esp <- numeric(n)
  while(cont < max_itera && crit>=tol){
    for(i in 1:n){
      pred <- betas[1]+betas[2]*Dados$w[i]
      aux1 <- -dnorm(pred)/pnorm(-pred)
      aux2 <- dnorm(pred)/( 1-pnorm(-pred) )
      Esp[i] <- pred + aux1*(Dados$x[i] == 0) + aux2*(Dados$x[i] == 1)
    }
    betas <- WW %*% Esp
    estimativas[cont+1,] <- betas
```

```
    crit <- max(abs(betas-estimativas[cont,])/(abs(estimativas[cont,])+0.001))
    cont <- cont + 1
  }
  return(list(betas = betas, iteracoes = estimativas[1:cont,],
              nitera = cont-1, error = crit))
}

## Funcao para ajustar o modelo - vetorizando
EM_probito2 <- function(Dados, inicial, max_itera = 200, tol = 0.001){
  n <- dim(Dados)[1]
  W <- cbind(rep(1, n), Dados$w)
  WW <- solve( t(W)%*%W ) %*% t(W)
  betas <- inicial
  estimativas <- matrix(0, ncol = 2, nrow = max_itera)
  estimativas[1,] <- betas
  cont <- 1;   crit <- 1
  while(cont < max_itera && crit >= tol){
    pred <- betas[1] + betas[2]*Dados$w
    aux1 <- -dnorm(pred)/pnorm(-pred)
    aux2 <- dnorm(pred)/( 1-pnorm(-pred) )
    Esp <- pred + aux1*(Dados$x==0) + aux2*(Dados$x==1)
    betas <- WW%*%Esp
    estimativas[cont+1,] <- betas
    crit <- max(abs(betas-estimativas[cont,])/(abs(estimativas[cont,])+0.001))
    cont <- cont + 1
  }
  return(list(betas = betas, iteracoes = estimativas[1:cont,],
              nitera = cont-1, error = crit))
}
```

```
## Simulando
n <- 100
dados <- gera_dados(n, rnorm(n), c(0.5, 2))
Ajuste <- EM_probito1(dados, c(0,1))
Ajuste2 <- EM_probito2(dados, c(0,1))

par( mfrow = c(1, 2), mar = c(4, 4, 4, 1) )
plot.ts(Ajuste[[2]][,1], type = "b", pch = 16, cex = 0.5,
        xlab = "Iterações", ylab = "Estimativas", main = expression(beta[0]))
```

```
plot.ts(Ajuste[[2]][,2], type = "b", pch = 16, cex = 0.5,
        xlab = "Iterações", ylab = "Estimativas", main = expression(beta[1]))
```

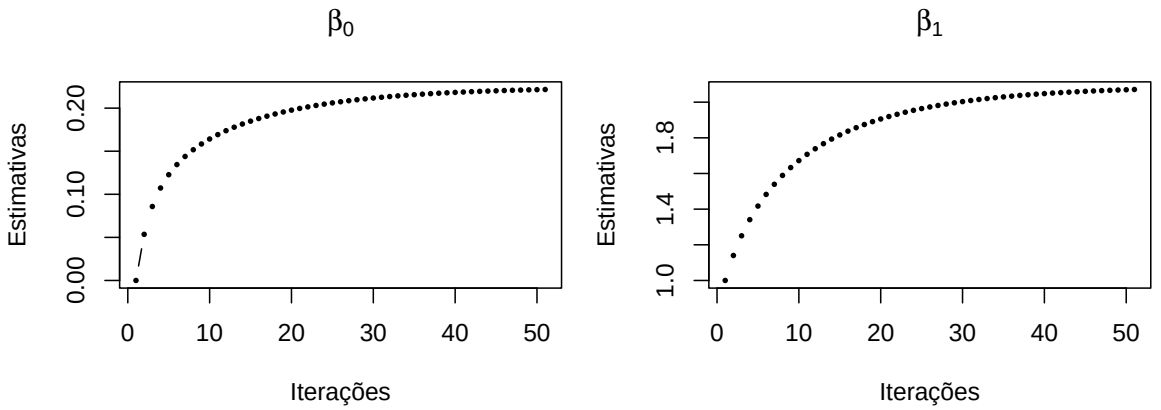

Figura 4.11: Estimativas dos parâmetros no modelo Probito com o algoritmo EM (Primeiro algoritmo).

```
## Curva com as estimativas
par( mfrow = c(1, 2), mar = c(4, 4, 4, 1) )
plot.ts(Ajuste2[[2]][,1], type = "b", pch = 16, cex = 0.5,
        xlab = "Iterações", ylab = "Estimativas", main = expression(beta[0]))
plot.ts(Ajuste2[[2]][,2], type = "b", pch = 16, cex = 0.5,
        xlab = "Iterações", ylab = "Estimativas", main = expression(beta[1]))
```

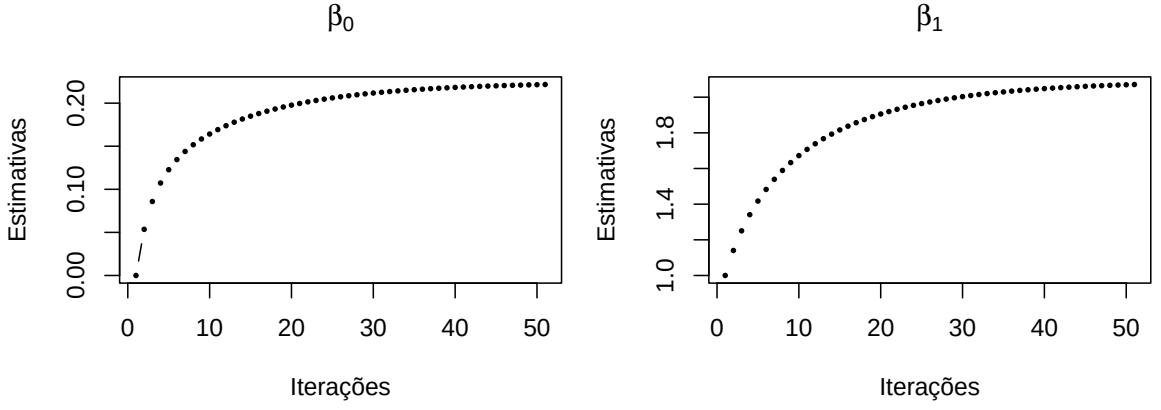

Figura 4.12: Estimativas dos parâmetros no modelo Probito com o algoritmo EM (Segundo algoritmo).

```
require(microbenchmark)
options(digits = 2)
microbenchmark(EM_probito1(dados, c(0,1)), EM_probito2(dados, c(0,1)), unit = "ms")
```

```
## Unit: milliseconds
##                         expr  min   lq mean median   uq  max neval cld
##  EM_probito1(dados, c(0, 1)) 27.8 29.5 32.7   30.9 33.7 86.5   100   b
##  EM_probito2(dados, c(0, 1))  2.4  2.4  2.6    2.5  2.6  5.7   100  a
```

4.3.3 Variantes do algoritmo EM

Existem algumas variações no algoritmo EM tanto no que diz respeito ao passo de esperança quanto sobre o passo de maximização.

4.3.3.1 Variações no passo M do algoritmo

Uma das vantagens do algoritmo EM é o fato de que frequentemente a derivação e maximização de $Q(\theta|\theta^{(t)})$ é mais simples do que os cálculos com a verossimilhança para dados incompletos. Em alguns casos, a maximização de $Q(\theta|\theta^{(t)})$ pode não ser tão simples e algumas estratégias foram propostas para facilitar o passo M do algoritmo.

4.3.3.1.1 Algoritmo EM Condicional O algoritmo ECM de Meng and Rubin (1993) propõe substituir o passo M por uma série de passos de maximizações condicionais computacionalmente mais simples. Cada maximização condicional é construída para ser um problema simples de otimização com restrições para θ para um subespaço particular e permitindo uma solução analítica ou uma solução numérica elementar.

Denotamos a coleção de passos simples CM depois do k-ésimo passo E de **ciclo CM**.

Ciclo CM

- Seja S o número total de passos CM dentro de cada ciclo;
- Para $s = 1, \dots, S$, o s-ésimo passo CM dentro do t-ésimo ciclo requer a maximização de $Q(\theta|\theta^{(t)})$ sujeito a restrições

$$g_s(\theta^{(t+(s-1)/S)}),$$

  em que $\theta^{(t+(s-1)/S)}$ é o máximo encontrado no $(s-1)$-ésimo passo CM do ciclo atual. Isto é, as restrições levam em conta valores obtidos no passo anterior;
- Quando o ciclo é terminado, fazemos $\theta^{(t+1)} = \theta^{(t+S/S)}$.

Usualmente é natural particionar θ em S sub-vetores, isto é,

$$\theta = (\theta_1, \dots, \theta_S).$$

Então o s-ésimo passo CM se torna maximizar Q com respeito a θ_s enquanto mantém os outros componentes de θ fixados. Neste caso, a função de restrições é

$$g_s(\theta) = (\theta_1, \dots, \theta_{s-1}, \theta_{s+1}, \dots, \theta_S).$$

Observações:

- Alternativamente, o s-ésimo passo CM pode ser maximizar Q com respeito a todos os outros elementos de θ enquanto θ_s é mantido fixo, ou seja, $g_s(\theta) = \theta_s$.
- Uma variação do ECM insere um passo E entre cada par de passos CM, atualizando assim Q em todas as etapas do ciclo CM.

Exemplo 4.5. (Mistura de distribuições Gama)
Suponha $X_1, \dots, X_n$ iid tal que

$$f_{X_i}(x_i|\theta) = \delta f(x_i|\alpha_1, \beta_1) + (1-\delta) f(x_i|\alpha_2, \beta_2),$$

em que $\theta = (\delta, \alpha_1, \alpha_2, \beta_1, \beta_2)$ e $f(x_i|\alpha_k, \beta_k) = x_i^{\alpha_k - 1} e^{-x_i \beta_k} \frac{\beta_k^{\alpha_k}}{\Gamma(\alpha_k)}$, para $k = 1$ e 2.

Podemos inserir variáveis latentes Z_i e reescrever o modelo

$$\begin{aligned} f_{X_i}(x_i|\theta, z_i) &= f(x_i|\alpha_1, \beta_1)^{z_i} f(x_i|\alpha_2, \beta_2)^{1-z_i}; \\ f_{Z_i}(z_i|\delta) &= \delta^{z_i}(1-\delta)^{1-z_i}. \end{aligned}$$

Note que,

$$\begin{aligned} Q(\theta|\theta^{(t)}) &= \sum_{i=1}^{n} E[Z_i|x_i, \theta^{(t)}] \left\{\log \delta + \log f(x_i|\alpha_1, \beta_1)\right\} \\ &+ \sum_{i=1}^{n} (1 - E[Z_i|x_i, \theta^{(t)}]) \left\{\log (1-\delta) + \log f(x_i|\alpha_2, \beta_2)\right\} \end{aligned}$$

e

$$Z_i|x_i, \theta^{(t)} \sim Bernoulli\left(p_i\right), \text{ com } p_i = \frac{\delta^{(t)} f(x_i|\alpha_1^{(t)}, \beta_1^{(t)})}{\delta^{(t)} f(x_i|\alpha_1^{(t)}, \beta_1^{(t)}) + (1-\delta^{(t)}) f(x_i|\alpha_2^{(t)}, \beta_2^{(t)})}.$$

Assim,

$$\begin{aligned}
\delta^{(t+1)} &= \frac{\sum_{i=1}^{n} E[Z_i|x_i,\theta^{(t)}]}{n};\\
\beta_1^{(t+1)} &= \frac{\alpha_1^{(t)}\sum_{i=1}^{n} E[Z_i|x_i,\theta^{(t)}]}{\sum_{i=1}^{n} x_i E[Z_i|x_i,\theta^{(t)}]};\\
\beta_2^{(t+1)} &= \frac{\alpha_2^{(t)}(n-\sum_{i=1}^{n} E[Z_i|x_i,\theta^{(t)}])}{\sum_{i=1}^{n} x_i - \sum_{i=1}^{n} x_i E[Z_i|x_i,\theta^{(t)}]}.
\end{aligned}$$

Enquanto, $\alpha_1^{(t+1)}$ e $\alpha_2^{(t+1)}$ são obtidos numericamente tal que

$$\begin{aligned}
&\sum_{i=1}^{n} E[Z_i|x_i,\theta^{(t)}]\left\{\log x_i + \log\beta_1^{(t+1)} - \frac{\Gamma'(\alpha_1^{(t+1)})}{\Gamma(\alpha_1^{(t+1)})}\right\} = 0;\\
&\sum_{i=1}^{n}(1-E[Z_i|x_i,\theta^{(t)}])\left\{\log x_i + \log\beta_2^{(t+1)} - \frac{\Gamma'(\alpha_2^{(t+1)})}{\Gamma(\alpha_2^{(t+1)})}\right\} = 0.
\end{aligned}$$

4.3.3.1.2 Algoritmo EM gradiente Para evitar *loops* em casos em que são necessários métodos numéricos de maximização, Lange (1995) propõe substituir o passo M por um simples passo do método de Newton, aproximando o máximo sem obter a solução "exata". Desta forma, o passo M do algoritmo é substituído por

$$\begin{aligned}
\theta^{(t+1)} &= \theta^{(t)} - Q''(\theta|\theta^{(t)})^{-1}\big|_{\theta=\theta^{(t)}} Q'(\theta|\theta^{(t)})\big|_{\theta=\theta^{(t)}}\\
&= \theta^{(t)} - Q''(\theta|\theta^{(t)})^{-1}\big|_{\theta=\theta^{(t)}} l'(\theta^{(t)}|\mathbf{x}),
\end{aligned}$$

em que $l'(\theta^{(t)}|\mathbf{x})$ é a avaliação da função escore na iteração atual. A segunda expressão segue da observação de que $\theta^{(t)}$ maximiza $Q(\theta|\theta^{(t)}) - l(\theta|\mathbf{x})$ e, portanto

$$Q'(\theta|\theta^{(t)}) - l'(\theta|\mathbf{x}) = 0.$$

4.3.3.2 Variações no passo E do algoritmo

O passo E do algoritmo EM requer o cálculo da esperança condicional da log-verossimilhança dos dados completos condicional nos dados observados. Em alguns casos, quando o cálculo desta esperança pode ser muito complexo e ela pode ser aproximada via método Monte Calo, como ocorre nos algoritmos EM Monte Carlo, EM Estócastico, EM com aproximação Estocástica, entre outros. Nesta apostila, iremos apresentar apenas o algorimo EM Monte Carlo.

4.3.3.2.1 Algoritmo EM Monte Carlo Wei and Tanner (1990) propuseram que o passo E do algoritmo EM pode ser substituído pelos seguintes passos:

Passo E do Algoritmo EM Monte Carlo

- Amostrar $\mathbf{Z}_1^{(t+1)}, \ldots, \mathbf{Z}_L^{(t+1)}$ iid de $f_{\mathbf{Z}|\mathbf{X}}(\mathbf{z}|\mathbf{x}, \theta^{(t)})$ em que cada $\mathbf{Z}_l^{(t+1)}$ é um vetor de dados faltantes para completar os dados observados, então $\mathbf{Y}_l = (\mathbf{x}, \mathbf{Z}_l^{(t+1)})$ denota um conjunto de dados completos em que os dados faltante foram substituídos por $\mathbf{Z}_l^{(t+1)}$;
- Calcule a aproximação de $Q(\theta|\theta^{(t)})$ dada por

$$\hat{Q}_L(\theta|\theta^{(t)}) = \frac{1}{L}\sum_{l=1}^{L} l(\theta|\mathbf{Y}_l^{(t)}).$$

Algumas observações podem ser feitas a respeito deste método:

- Se escolhermos $L = 1$, obtemos o caso particular introduzido por Celeux and Diebolt (1985) conhecido como algoritmo EM estocástico;
- Valores maiores de L levam a estimativas mais precisas, mas o algoritmo se torna lento (Booth and Hobert, 1999);
- Wei and Tanner (1990) propôs uma estratégia para construir um algoritmo MCEM mais eficiente que consiste em assumir valores pequenos de L nos primeiros passos, quando as estimativas ainda estão longe dos parâmetros verdadeiros, e aumentando o valor de L a medida que as iterações aumentam para reduzir a variabilidade introduzida pelo método Monte Carlo.
- Procedimentos ad-hoc foram propostos para definir L em cada iteração (McCulloch, 1997; Chan and Kuk, 1997);
- Procedimentos automáticos para definir L são discutidos, por exemplo, por Booth and Hobert (1999), Levine and Casella (2001), Levine and Fan (2004), Caffo et al. (2005) no contexto de modelos MLR normais. Tais procedimentos são geralmente baseados na avaliação do erro de Monte Carlo.

O algoritmo EMMC pode não convergir da mesma forma do que o EM ordinário. Os valores de $\widehat{\theta}$ ficam variando em torno do máximo e a variabilidade depende do valor L. Uma possibilidade é usar a média das estimativas nas iterações finais do algoritmo como estimativa.

Exemplo 4.6. (Modelo Probito revisitado)
Suponha uma amostra aleatória simples de X_i, $i = 1, \dots, n$, tal que

$$X_i \sim Bernoulli(p_i),$$

em que $p_i = \Phi(\beta_0 + w_i\beta_1)$ e w_i é a covariável observada o indivíduo i. Note que para estimar tal modelo para um conjunto de dados é necessário maximizar numericamente uma expressão que irá envolver a inversa da distribuição acumulada da normal padrão.

Lembrando do Exemplo 2.4 que a distribuição de $Z_i \mid X_i$ é dada por

$$f_{Z_i|X_i}(z_i|x_i,\theta) = \frac{f_{X_i|Z_i}(x_i|z_i)f_{Z_i}(z_i|\theta)}{\int f_{X_i|Z_i}(x_i|z_i)f_{Z_i}(z_i|\theta)dz_i} = \begin{cases} \frac{\phi(z_i|\mu_{z_i},1)}{1-\Phi(0|\mu_{z_i},1)}I\{z_i \geq 0\} & \text{se } x_i = 1; \\ \frac{\phi(z_i|\mu_{z_i},1)}{\Phi(0|\mu_{z_i},1)}I\{z_i < 0\} & \text{se } x_i = 0. \end{cases}$$

Note que $Z_i|X_i = x_i$ tem distribuição normal truncada com parâmetro de locação μ_{z_i} e parâmetro de escala igual a 1, colocando massa de probabilidade em valores de Z_i maiores o iguais a 0 se $x_i = 1$ e menores do que 0 se $x_i = 0$.

Em cada iteração $t+1$ do algoritmo EMMC, geramos amostras aleatórias $z_1^{(l)}, \dots, z_n^{(l)}$, para $l = 1, \dots, L$, a partir da distribuição de $Z_i|X_i = x_i$ e obtemos

$$\begin{aligned}
\hat{Q}(\theta|\theta^{(t)}) &= \frac{1}{L}\sum_{l=1}^{L} \log f_{\mathbf{Y}}(\mathbf{y}^{(l)}|\theta) \\
&= \frac{1}{L}\sum_{l=1}^{L}\left[-\frac{1}{2}\sum_{i=1}^{n}(z_i^{(l)})^2 + \sum_{i=1}^{n} z_i^{(l)}(\beta_0+\beta_1 w_i) - \frac{1}{2}\sum_{i=1}^{n}(\beta_0+\beta_1 w_i)^2 + C\right] \\
&= -\frac{1}{2}\frac{1}{L}\sum_{l=1}^{L}\sum_{i=1}^{n}(z_i^{(l)})^2 + \frac{1}{L}\sum_{l=1}^{L}\sum_{i=1}^{n} z_i^{(l)}(\beta_0+\beta_1 w_i) - \frac{1}{2}\sum_{i=1}^{n}(\beta_0+\beta_1 w_i)^2 + C \\
&= -\frac{1}{2}\left[\frac{1}{L}\sum_{l=1}^{L}(\mathbf{z}^{(l)})^T\mathbf{z}^{(l)}\right] + \beta^T\mathbf{W}^T\left[\frac{1}{L}\sum_{l=1}^{L}\mathbf{z}^{(l)}\right] - \frac{1}{2}(\mathbf{W}\beta)^T(\mathbf{W}\beta) + C,
\end{aligned}$$

em que C é uma constante com respeito aos parâmetros, $\beta^T = (\beta_0, \beta_1)$, $\mathbf{z}^{(l)} = (z_1^{(l)}, \dots, z_n^{(l)})^T$ e $\mathbf{W} = (\mathbf{1}_n, (w_1, \dots, w_n)^T)$.

Maximizando obtemos

$$\beta^{(t+1)} = (\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T\left[\frac{1}{L}\sum_{l=1}^{L}\mathbf{z}^{(l)}\right].$$

Com isso, neste exemplo o algoritmo EM é dado a seguir.

Algoritmo

- Iniciar $\beta^{(0)} = (\beta_0^{(0)}, \beta_1^{(0)})$ e faça $t = 0$;
- **Passo E:** Calcular uma aproximação para $E[\mathbf{Z}|\mathbf{x}, \theta^{(t)}]$:
  - Gerar $z_1^{(l)}, \dots, z_n^{(l)}$, para $l = 1, \dots, L$, a partir da distribuição de $Z_i|X_i = x_i$;
  - Calcular $\frac{1}{L}\sum_{l=1}^{L} \mathbf{z}^{(l)}$;
- **Passo M:** Calcular $\beta^{(t+1)}$ e fazer $t = t + 1$;
- Calcular um critério de parada. Se tolerância for atendida faça $\widehat{\beta}$ igual a médias das últimas estimativas. Caso contrário, incremente t e volte ao passo 2.

No R temos:

```
## Geração da normal truncada
rnormtrunc <- function(n, mu, sigma, a, b){
  us <- runif(n)
  amostra <- qnorm( pnorm(a, mu, sigma) +
                      (pnorm(b, mu, sigma) - pnorm(a, mu, sigma))*us, mu, sigma )
}

## Funcao para gerar dados do modelo probito
gera_dados <- function(n, w, betas){
  prob <- pnorm(betas[1] + betas[2]*w)
  x <- rbinom(n, 1, prob)
  return( data.frame(x=x, w=w) )
}

## Funcao para ajustar o modelo usando um loop desnecessario
EMMC_probito <- function(Dados, inicial, max_itera = 1000, tol = 0.001, Ls){
  n <- dim(Dados)[1]
  W <- cbind(rep(1, n), Dados$w)
  WW <- solve( t(W)%*%W ) %*% t(W)
  betas <- inicial
  estimativas <- matrix(0, ncol = 2, nrow = max_itera)
```

```
  estimativas[1,] <- betas
  cont <- 1;   crit <- 1
  Esp <- numeric(n)
  cont_convergencia <- 0

  while(cont < max_itera && cont_convergencia < 3){
    L <- Ls[cont]
    if(crit < tol){
      cont_convergencia <- cont_convergencia + 1
    }else{
      cont_convergencia <- 0
    }

    for(i in 1:n){
      pred <- betas[1] + betas[2]*Dados$w[i]
      if(Dados$x[i] == 1){
        amostra_zi <- rnormtrunc(L, pred, 1, 0, 100000)
      }else{
        amostra_zi <- rnormtrunc(L, pred, 1, -100000, 0)
      }
      Esp[i] <- mean(amostra_zi)
    }
    betas <- WW %*% Esp
    estimativas[cont+1,] <- betas
    crit <- max(abs(betas - estimativas[cont,])/(abs(estimativas[cont,])+0.001))
    cont <- cont + 1
  }
  return(list(betas = betas, iteracoes = estimativas[1:cont,],
              nitera = cont-1, error = crit))
}
```

```
## Simulando com tamanho amostral 100
set.seed(1234)
n <- 100
dados <- gera_dados(n, rnorm(n), c(0.5, 2))
Ls <- c(rep(5, 100), rep(10, 100), rep(30, 100), rep(100, 100),
       rep(500, 100), rep(1000, 100),  rep(5000, 1000) )
Ajuste <- EMMC_probito(dados, c(0,1), Ls = Ls, tol = 0.001 )
Ajuste2 <- EMMC_probito(dados, c(0.4, 2.3), Ls = Ls, tol = 0.001 )
```

```
## Curva com as estimativas
par(mfrow = c(3, 2), mar = c(4, 4, 3, 1))

plot.ts(Ajuste[[2]][,1], type = "o", pch = 16, cex = 0.5,
        xlab = "Iterações", ylab = "Estimativas", main = expression(beta[0]))
abline(v = c(100, 200, 300, 400, 500, 600), lty = 3, col = "gray")
plot.ts(Ajuste[[2]][,2], type = "o", pch = 16, cex = 0.5,
        xlab = "Iterações", ylab = "Estimativas", main = expression(beta[1]))
abline(v = c(100, 200, 300, 400, 500, 600), lty = 3, col = "gray")

## Melhorando os valores iniciais
plot.ts(Ajuste2[[2]][,1], type = "o", pch = 16, cex = 0.5,
        xlab = "Iterações", ylab = "Estimativas", main = expression(beta[0]))
abline(v = c(100, 200, 300, 400, 500, 600), lty = 3, col = "gray")
plot.ts(Ajuste2[[2]][,2], type = "o", pch = 16, cex = 0.5,
        xlab = "Iterações", ylab = "Estimativas", main = expression(beta[1]))
abline(v = c(100, 200, 300, 400, 500, 600), lty = 3, col = "gray")

## Aumentando o tamanho amostral
set.seed(1234)
n <- 500
dados <- gera_dados(n, rnorm(n), c(0.5, 2))
Ajuste <- EMMC_probito(dados, c(0,1), Ls = Ls, tol = 0.001 )

## Curva com as estimativas
plot.ts(Ajuste[[2]][,1], type = "o", pch = 16, cex = 0.5,
        xlab = "Iterações", ylab = "Estimativas", main = expression(beta[0]))
abline(v = c(100, 200, 300, 400, 500, 600), lty = 3, col = "gray")
plot.ts(Ajuste[[2]][,2], type = "o", pch = 16, cex = 0.5,
        xlab = "Iterações", ylab = "Estimativas", main = expression(beta[1]))
abline(v = c(100, 200, 300, 400, 500, 600), lty = 3, col = "gray")
```

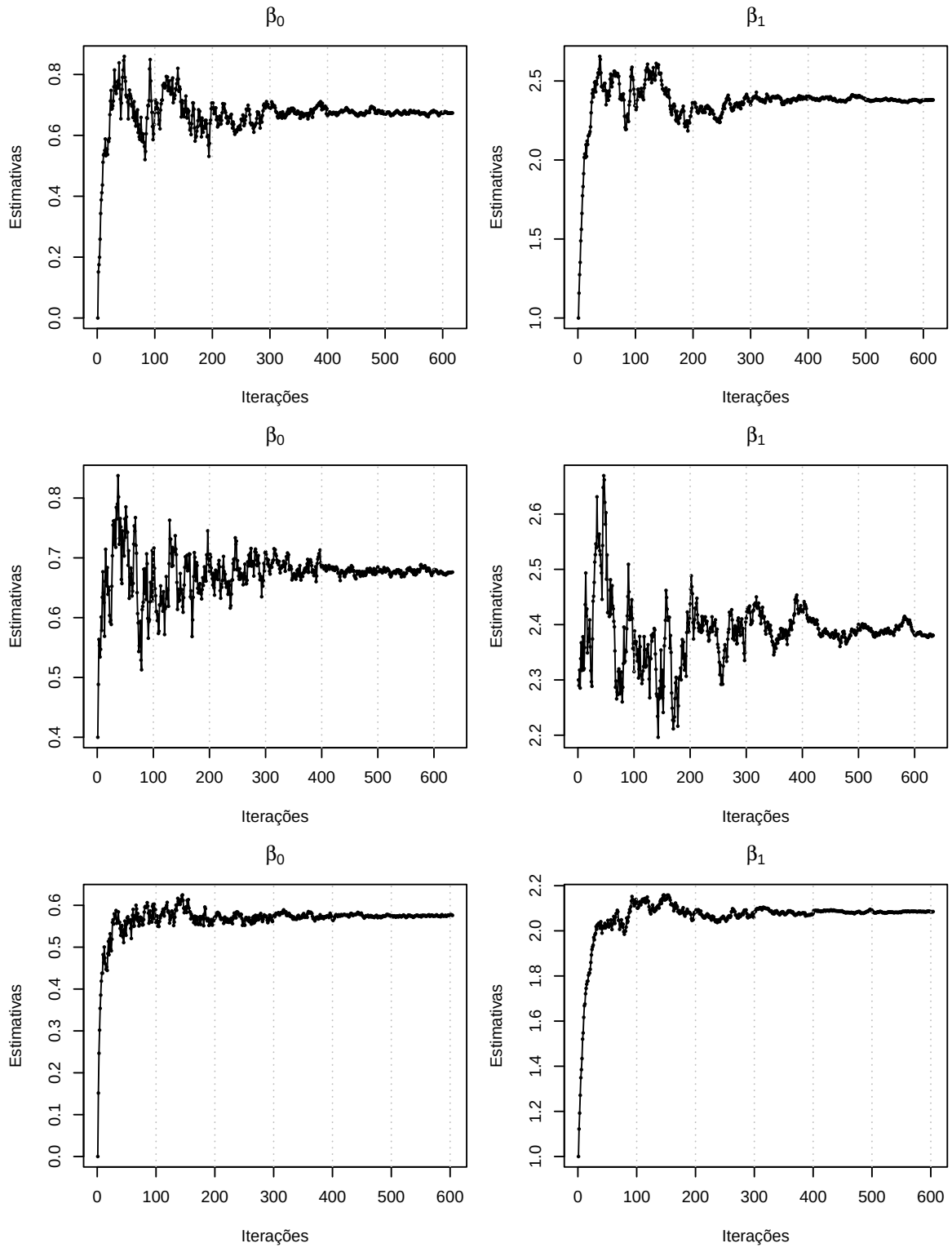

Figura 4.13: Estimativas dos parâmetros no modelo Probito com o algoritmo EMMC com amostra de tamanho 100 e 500 (última linha).

4.3.4 Estimação da variância

O algoritmo EM não fornece automaticamente uma matriz de covariância dos estimadores de máxima verossimilhança. Uma caminho para estimar a matriz de covariância é calcular a informação observada $-I''(\hat{\theta}|\mathbf{x})$, em que I'' é a matriz hessiana de segundas derivadas de $\log L(\theta|\mathbf{x})$. Em alguns casos, a matriz Hessiana pode ser calculada analiticamente, mas em outros pode ser difícil de obter.

4.3.4.1 Princípio da Informação de Louis

Note que,

$$\log f_{\mathbf{X}}(\mathbf{x}|\theta) = \log f_{\mathbf{Y}}(\mathbf{y}|\theta) - \log f_{\mathbf{Z}|\mathbf{X}}(\mathbf{z}|\mathbf{x},\theta).$$

Então,

$$\begin{aligned} E[\log f_{\mathbf{X}}(\mathbf{x}|\theta)|\mathbf{x},\theta^{(t)}] &= E[\log f_{\mathbf{Y}}(\mathbf{y}|\theta)|\mathbf{x},\theta^{(t)}] - E[\log f_{\mathbf{Z}|\mathbf{X}}(\mathbf{z}|\mathbf{x},\theta)|\mathbf{x},\theta^{(t)}] \\ \log f_{\mathbf{X}}(\mathbf{x}|\theta) &= Q(\theta|\theta^{(t)}) - H(\theta|\theta^{(t)}), \end{aligned}$$

em que as esperanças são calculadas com respeito a distribuição de $\mathbf{Z}|(\mathbf{x},\theta^{(t)})$ e $H(\theta|\theta^{(t)}) = E[\log f_{\mathbf{Z}|\mathbf{X}}(\mathbf{z}|\mathbf{x},\theta)|\mathbf{x},\theta^{(t)}]$. O princípio da falta de informação nos diz que a informação observada é dada por

$$-I''(\theta|\mathbf{x}) = -Q''(\theta|\omega)|_{\omega=\theta} + H''(\theta|\omega)|_{\omega=\theta},$$

em que as derivadas são com respeito a θ. A equação acima pode ser reescrita como

$$\hat{i}_{\mathbf{X}}(\theta) = \hat{i}_{\mathbf{Y}}(\theta) - \hat{i}_{\mathbf{Z}|\mathbf{X}}(\theta),$$

em que $\hat{i}_{\mathbf{X}}(\theta) = -l(\theta|\mathbf{x})$ é informação observada, e $\hat{i}_{\mathbf{Y}}(\theta)$ e $\hat{i}_{\mathbf{Z}|\mathbf{X}}(\theta)$ podem ser chamadas de informação completa e informação faltante, respectivamente. Através deste resultado vemos que a informação observada é igual à informação completa menos a informação faltante. Trocando a ordem da integração e da diferenciação quando possível, temos que a informação completa é

$$\hat{i}_{\mathbf{Y}}(\theta) = -Q''(\theta|\omega)|_{\omega=\theta} = -E\left[l''(\theta|\mathbf{Y}) \mid \mathbf{x},\theta\right].$$

Pode ser mostrado que

$$\hat{i}_{\mathbf{Z}|\mathbf{X}}(\theta) = -H''(\theta|\omega)|_{\omega=\theta} = Var\left[S_{\mathbf{Z}|\mathbf{X}}(\theta)|\mathbf{x},\theta^{(t)}\right],$$

com $S_{\mathbf{Z}|\mathbf{X}}(\theta) = \frac{d\log f_{\mathbf{Z}|\mathbf{X}}(\mathbf{z}|\mathbf{x},\theta)}{d\theta}$ e onde a variância é calculada com respeito a $f_{\mathbf{Z}|\mathbf{X}}$. Assim,

$$\hat{i}_{\mathbf{X}}(\theta) = -E\left[l''(\theta|\mathbf{Y}) \mid \mathbf{x},\hat{\theta}\right] - Var\left[S_{\mathbf{Z}|\mathbf{X}}(\theta)|\mathbf{x},\hat{\theta}\right].$$

Dado que o escore esperado é zero em $\hat{\theta}$, segue que

$$\hat{i}_{\mathbf{Z}|\mathbf{X}}(\hat{\theta}) = E\left[S_{\mathbf{Z}|\mathbf{X}}(\hat{\theta})S_{\mathbf{Z}|\mathbf{X}}(\hat{\theta})^T \mid \mathbf{x}, \hat{\theta}\right].$$

Se $\hat{i}_{\mathbf{Y}}$ e $\hat{i}_{\mathbf{Z}|\mathbf{X}}$ são difíceis de calcular analiticamente, eles podem ser estimados via método Monte Carlo. Isto é,

$$\hat{i}_{\mathbf{Y}} \approx -\frac{1}{m}\sum_{i=1}^{m}\frac{d^2 \log f_{\mathbf{Y}}(\mathbf{y}_i|\theta)}{d\theta d\theta^T},$$

onde, para $i = 1, \ldots, m$, $\mathbf{y}_i = (\mathbf{x}, \mathbf{z}_i)$ é um vetor simulado de dados completos consistindo dos dados observados e dados gerados iid de $f_{\mathbf{Z}|\mathbf{X}}$. Similarmente, um estimador Monte Carlo de $\hat{i}_{\mathbf{Z}|\mathbf{X}}$ é a variância amostral de $\frac{d \log f_{\mathbf{Z}|\mathbf{X}}(\mathbf{z}_i|\mathbf{x},\theta)}{d\theta}$ obtidos de uma amostra de $\mathbf{z}_i$.

Além disso, Louis (1982) mostrou que

$$\begin{aligned}\hat{i}_{\mathbf{X}}(\theta) = -E\left[l''(\theta|\mathbf{Y}) \mid \mathbf{x}, \hat{\theta}\right] - E\left[S_{\mathbf{Y}}(\theta)S_{\mathbf{Y}}(\theta)^T \mid \mathbf{x}, \hat{\theta}\right] \\ + E\left[S_{\mathbf{Y}}(\theta) \mid \mathbf{x}, \hat{\theta}\right] E^T\left[S_{\mathbf{Y}}(\theta) \mid \mathbf{x}, \hat{\theta}\right],\end{aligned}$$

em que $S_{\mathbf{Y}}(\theta) = \frac{dl(\theta|\mathbf{Y})}{d\theta}$.

4.3.4.2 Informação Empírica

A matriz de informação empírica (Meilijson, 1989) é um estimador consistente para a Informação de Fisher esperada no caso iid que pode ser utilizada para calcularmos o erro padrão dos estimadores. Quando os dados são independentes e identicamente distribuídos (iid), note que $\frac{d \log f_{\mathbf{X}}(\mathbf{x}|\theta)}{d\theta} = l'(\theta|\mathbf{x}) = \sum_{i=1}^{n} l'(\theta|x_i)$, em que $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n)$. Disto, a informação empírica é definida como

$$IE(\theta) = \frac{1}{n}\sum_{i=1}^{n} l'(\theta|x_i)l'(\theta|x_i)^T - \frac{1}{n^2}l'(\theta|\mathbf{x})l'(\theta|\mathbf{x})^T.$$

Sabemos que $\theta^{(t)}$ maximiza a quantidade $\left[Q(\theta|\theta^{(t)}) - l(\theta|\mathbf{x})\right]$ com respeito a θ. Tomando as derivadas, temos

$$Q'(\theta|\theta^{(t)})|_{\theta=\theta^{(t)}} = l'(\theta|\mathbf{x})|_{\theta=\theta^{(t)}}.$$

Dado que Q' é calculado em cada passo M do algoritmo, como

$$Q(\theta|\theta^{(t)}) = \sum_{i=1}^{n} E[\log L(\theta|Y_i)|x_i, \theta^{(t)}] = \sum_{i=1}^{n} Q_i(\theta|\theta^{(t)}),$$

os termos individuais da informação empírica estão disponíveis. Assim, podemos calcular $IE(\theta^{(t)})$ substituíndo $l'(\theta|\mathbf{x})|_{\theta=\theta^{(t)}}$ e $l'(\theta|x_i)|_{\theta=\theta^{(t)}}$ por $Q'(\theta|\theta^{(t)})|_{\theta=\theta^{(t)}}$ e $Q_i'(\theta|\theta^{(t)})|_{\theta=\theta^{(t)}}$, respectivamente.

4.3.4.3 Bootstrap

O método bootstrap para obter uma estimativa da matriz de covariância para o EM para variáveis observadas iid é dado por:

Algoritmo
Passo 1: Calcule $\widehat{\theta}_{EM}$ utilizando o algoritmo EM para $\mathbf{x}_1, \ldots, \mathbf{x}_n$. Faça $j = 1$ e faça $\widehat{\theta}_1 = \widehat{\theta}_{EM}$;
Passo 2: Incremente j e amostre pseudo-dados $\mathbf{x}_1^*, \ldots, \mathbf{x}_n^*$ aleatoriamente de $\mathbf{x}_1, \ldots, \mathbf{x}_n$ com reposição;
Passo 3: Calcule $\widehat{\theta}_j$ utilizando o algoritmo EM para os pseudo-dados $\mathbf{x}_1^*, \ldots, \mathbf{x}_n^*$;
Passo 4: Pare se j é tão grande quanto desejado (B); Caso contrário retorne ao passo 2.
Passo 5: Calcule a matriz de variância-covariância amostral do vetor $\widehat{\theta}_1, \ldots, \widehat{\theta}_B$.

Esta metodologia se torna ineficiente quando a solução de cada EM é lenta por causa da alta proporção de dados faltantes ou alta dimensionalidade.

4.3.5 Exercícios

Exercício 4.4. Implemente o algoritmo EMC para o exemplo de misturas de distribuições Gama e ilustre o uso do algoritmo com dados simulados.

Exercício 4.5. Considere um modelo em que

$$\begin{aligned} &Y_i = \mu + e_i, \\ &e_i \sim N(0, \sigma^2 w_i^{-1}), \\ &w_i \sim Gama(\nu/2, \nu/2), \text{ (mesma parametrização do Exemplo 5.2)} \end{aligned}$$

para $i = 1, \ldots, n$.

a) Calcule a distribuição condicional de $W_i | Y_i = y_i$.
b) Implemente o algoritmo EM para estimar μ e σ^2, considerando que ν é conhecido, que w_i são variáveis não observadas e tamanho amostral igual a n. Apresente as contas necessárias.
c) Ilustre o uso do algoritmo com dados simulados das distribuição t-Student com graus de liberdade ν igual a 2, 3, 10 e 30. Utilize tamanho amostral $n =$ 10, 30 e 100.

Referências Bibliográficas

Booth, J. G. and Hobert, J. P. (1999). Maximizing generalized linear mixed model likelihoods with an automated monte carlo em algorithm. *J. Roy. Stat. Soc. B*, 61(1):265–285.

Caffo, B. S., Jank, W., and Jones, G. L. (2005). Ascent-based monte carlo expectation–maximization. *J. Roy. Stat. Soc. B*, 67(2):235–251.

Celeux, G. and Diebolt, J. (1985). The SEM algorithm: a probabilistic theacher algorithm derived from EM algorithm for the mixture problem. *Comput. Stat. Quart.*, 2:73–82.

Chan, J. S. K. and Kuk, A. Y. C. (1997). Maximum likelihood estimation for probit-linear mixed models with correlated random effects. *Biometrics*, 53(1):86–97.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *J. Roy. Stat. Soc. B*, 39(1):1–38.

Efron, B. and Tibshirani, R. J. (1994). *An introduction to the bootstrap*. CRC press.

Gilks, W. R. (1992). *Bayesian Statistics 4*, chapter Derivative-free adaptive rejection sampling for Gibbs sampling. Oxford University Pres.

Gilks, W. R. and Wild, P. (1992). Adaptive rejection sampling for gibbs sampling. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 41(2):337–348.

Givens, G. H. and Hoeting, J. A. (2012). *Computational statistics*, volume 703. John Wiley & Sons.

Lange, K. (1995). A gradient algorithm locally equivalent to the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 57(2):425–437.

Levine, R. A. and Casella, G. (2001). Implementations of the Monte carlo EM algorithm. *J. Comput. Graph. Stat.*, 10(3):422–439.

Levine, R. A. and Fan, J. (2004). An automated (markov chain) monte carlo em algorithm. *J. Stat. Comput. Sim.*, 74(5):349–360.

Louis, T. A. (1982). Finding the observed information matrix when using the em algorithm. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 44:226–233.

McCulloch, C. E. (1997). Maximum likelihood algorithms for generalized linear mixed models. *J. Am. Stat. Assoc.*, 92(437):162–170.

Meilijson, I. (1989). A fast improvement to the em algorithm on its own terms. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 127–138.

Meng, X.-L. and Rubin, D. B. (1993). Maximum likelihood estimation via the ecm algorithm: A general framework. *Biometrika*, 80(2):267–278.

Rizzo, M. L. (2008). *Statistical computing with R*. CRC Press.

Ross, S. M. (2013). *Simulation*. Academic Press.

Wei, G. C. and Tanner, M. A. (1990). A Monte Carlo implementation of the EM algorithm and the poor man's data augmentation algorithms. *J. Am. Stat. Assoc.*, 85(411):699–704.