

Estatística Computacional I

Lupércio França Bessegato
Dep. de Estatística/UFJF



Roteiro Geral



1. Programando em R
2. Preparação, limpeza e manipulação de dados
3. Gráficos em R
4. Tópicos especiais
5. Referências

Estatística Computacional I - 2020

2

Programando em R

Funções



Funções



- Objetos que avaliam múltiplas expressões, usando argumentos

✓ Em geral, têm como saída um objeto

- Sintaxe:

```
nome.função<- function(argumentos) {
  conjunto de comandos da função
}

✓ Argumentos separados por vírgula
  - argumentoi, ou
  - argumentoi = valor default
```

Estatística Computacional I - 2020

196



- Comentários:

✓ As chaves são opcionais, se o conjunto de funções for uma única expressão

✓ Ao denominar as funções tenha cuidado para não sobrepor as funções internas do R

✓ Tenha cuidado com as funções de nomenclatura, pode sobrepor as funções R existentes!

✓ Uma função funciona como qualquer outro objeto

– Pode ser usada como argumento de outras funções

Estatística Computacional I - 2020

197



- ✓ Use # para inserir anotações em sua função
- ✓ Se você digitar o nome de sua função, você verá o código da função.
- ✓ Usar o comando args para visualizar os argumentos requeridos para sua função ():

Estatística Computacional I - 2020

198



- Exemplo – 2^a. Lei de Ohm

$$R = \rho \frac{l}{A}$$

```
> # Exemplo - resistencia elétrica
> # função - área em m², comprimento em m e ro em ohm . m
> # ro para o cobre: 1.72e-8 ohm . m
> resistencia <- function(ele, area, ro = 1.72E-8) ele/area * ro
> # cabo de cobre com 80m, 0,5 mm²
> resistencia(80, 0.5E-6)
[1] 2.752
> resistencia(0.5E-6, 80)
[1] 1.075e-16
> resistencia(area = 0.5E-6, ele = 80)
[1] 2.752
> # resistencia - área em mm² e condutividade e-8
> resistencia <- function(ele, area, ro = 1.72E-8) ele/area * ro * 1E6
> resistencia(80, 0.5)
[1] 2.752
>
> # resistencia - comprimento em m, área em mm² e condutividade e-8 ohm.m
> resistencia <- function(ele = 1, area = 1, ro = 1.72) ele/area * ro
> resistencia()
[1] 1.72
```

Estatística Computacional I - 2020

199

✓ Resistência é vetorizada

```
> # resistencia é vetorizada
> # 80 m de cabo, com 0.5 mm2, para cobre e outro material
> resistencia(80, 0.5, ro = c(1.72, 6.25))
[1] 2.752 10.000
> # 80m de cobre com seções 1 e 0.5 mm2
> resistencia(80, a = c(1, 0.5))
[1] 1.376 2.752
> # 80 m de cabo de cobre com 1 mm2 e de outro material com 0.5mm2
> resistencia(80, a = c(1, 0.5), ro = c(1.72, 6.25))
[1] 1.376 10.000
```

200

Estatística Computacional I - 2020

• Exemplo – mediana acumulada

```
> # Exemplo – mediana acumulada
> # função para cálculo da mediana acumulada
> mediana.acum <- function(vet) {
+ # Cálculo da mediana média de vetor
+ # vet: vetor com observações
+ sapply(seq_along(vet), function(x) median(vet[1:x]))
+ }
> # amostra exponencial (é assimétrica)
> set.seed(666)
> amostra <- rexp(30)
> mediana.acum(amostra)
[1] 0.5487370 1.2564124 0.5487370 0.3441432 0.5487370 0.5502616 0.5517862
[8] 1.0376944 1.5236027 1.0730578 1.5236027 1.0730578 0.6225129 0.6176420
[15] 0.6225129 0.6912014 0.6225129 0.6176420 0.6127711 0.6110223 0.6092735
[22] 0.5805298 0.6092735 0.5805298 0.6092735 0.5805298 0.5517862 0.5502616
[29] 0.5487370 0.5205005
> medianas <- mediana.acum(amostra)
> (medias <- cumsum(amostra) / seq_along(amostra))
[1] 0.5487370 1.2564124 0.8841248 0.6654010 0.6426781 1.1212309 1.3341062
[8] 1.3577933 1.4375167 1.3560164 1.5002057 1.4021441 1.2963314 1.2475056
[15] 1.2149979 1.3352515 1.2699043 1.2052816 1.1563514 1.1289975 1.0760831
[22] 1.0394352 1.0328215 0.9947037 1.0136499 0.9847223 0.9561155 0.9228395
[29] 0.8926699 0.8793230
```

201

Estatística Computacional I - 2020

– Comparação com as médias acumuladas:

```
> # Gráfico comparativo medianas e médias
> plot(amostra, type = "l")
> lines(medianas, lty = 2, lwd = 2, col = "blue")
> lines(medias.acum, lty = 3, lwd = 2, col = "red")
```

✓ Médias são afetadas por valores extremos.
✓ Qual a mediana de uma distribuição exponencial com média 1?

202

Estatística Computacional I - 2020

• Comando args:

✓ Visualização dos argumentos necessários para sua função:

```
> # comando args
> args(resistencia)
function (ele, area = 0.5, ro = 1.72)
NULL
> args(mediana.acum)
function (vet)
NULL
```

203

Estatística Computacional I - 2020

Armazenamento de Objetos

- Comandos `save` e `load`:
 - ✓ Armazenamento em arquivo binário:

```
> # armazenamento de objetos - binário
> save(resistencia, mediana.acum, amostra, file = "minhas-bin.R")
> # carregamento de objetos binários armazenados
> load("minhas-bin.R")
> # tamanho do arquivo
> file.size("minhas-bin.R")
[1] 3457
> # tamanho de objetos
> object.size(mediana.acum)
6496 bytes
> # todos os objetos
> ls()
[1] "amostra"        "mediana.acum"    "medianas"       "medias"
[4] "resistencia"
```

204
Estatística Computacional I - 2020

- Comandos `dump` e `source`:
 - ✓ Armazenamento em arquivo texto:

```
> # armazenamento de objetos - texto
> dump(c("resistencia", "mediana.acum", "amostra"), file = "minhas-texto.R")
> # armazena objetos com padrão de nomes
> dump(is(pattern = "mediana|amostra|resistencia"), file = "minhas-texto-2.R")
> # Acrescenta objetos no arquivo
> amostra2 <- 1:100
> dump("amostra2", file = "minhas-texto.R", append = TRUE)
> # carregamento arquivo texto de objetos
> source("minhas-texto.R")
> file.size("minhas-texto.R")
> # alternativa para carregamento
> source(file.choose())
[1] 842
> # deleta arquivo
> unlink("minhas-texto-2.R")
```

205
Estatística Computacional I - 2020

• Exemplo – Razão de chances

- ✓ Razão de chance de tabela de contingência 2x2

		Doença	
		Sim	Não
Grupo	Tratamento	a	b
	Controle	c	d

✓ Razão de chances amostral: $\hat{\theta} = \frac{a}{b} = \frac{ad}{bc}$

✓ Erro padrão assintótico para $\log(\hat{\theta})$:

$$\text{ep}(\hat{\theta}) = \sqrt{\frac{1}{a} + \frac{1}{b} + \frac{1}{c} + \frac{1}{d}}$$

✓ Intervalo de confiança assintótico para $\log \theta$:

$$\log \hat{\theta} \pm z_{\alpha/2} \text{ep}(\hat{\theta})$$

– Exponenciam-se os limites para se obter IC para θ

206
Estatística Computacional I - 2020

- ✓ Função para cálculo de IC para θ :

```
# função para cálculo de IC para theta
razao <- function(X, nivel = 0.95) {
  # -----
  # Calcula a razão de chances de tabela 2x2 e
  # o intervalo de confiança assintótico para a razão de chances
  # Argumentos:
  #   X = matriz 2x2
  #   nível = nível de confiança
  # Adaptado de Christian, N. "Basic Programming-Lecture 3"
  #

  RC <- (X[1, 1]* X[2, 2])/(X[1, 2]*X[2, 1])
  ep.logRC <- sqrt(sum(1/X))
  alfa <- 1 - nivel
  IC.inf <- exp(log(RC) - qnorm(1 - alfa/2) * ep.logRC)
  IC.sup <- exp(log(RC) + qnorm(1 - alfa/2) * ep.logRC)
  cat("Razão de chances = ", RC, "\n",
      "Intervalo com", nível*100, "% de confiança = (",
      IC.inf, "; ", IC.sup, ")\n", sep="")
}
```

207
Estatística Computacional I - 2020

✓ Tabela de resultados de estudo

```
> # resultado de estudo
> tabela <- matrix(c(189, 104, 10845, 10933), nrow = 2,
+ dimnames = list("Grupos" = c("Placebo", "Aspirina"),
+ "Infarto do Miocárdio" = c("Sim", "Não")))
> tabela
   Infarto do Miocárdio
Grupos   Sim   Não
Placebo 189 10845
Aspirina 104 10933
```

✓ Cálculo do Intervalo de confiança

```
> razao(tabela)
Razão de chances = 1.832054
Intervalo com 95% de confiança = (1.440042, 2.33078)
> # intervalo com 99% de confiança
> razao(tabela, nivel = 0.99)
Razão de chances = 1.832054
Intervalo com 99% de confiança = (1.335117, 2.513954)
> # intervalo com 90% de confiança
> razao(tabela, nivel = 0.90)
Razão de chances = 1.832054
Intervalo com 90% de confiança = (1.496877, 2.242282)
```

Estatística Computacional I - 2020 208

Objetos de Saída

- Em geral, desejamos que a função gere objeto que possa ser utilizado
 - Comandos: `return` e `invisible`
 - `return`: imprime e retorna seus argumentos
 - `invisible`: retorna valores, mas não imprime
 - Usar uma lista para retornar vários objetos
 - Se não houver `return` na função, será retornado o valor da última expressão avaliada

Estatística Computacional I - 2020 209

• Função com saída de objetos

```
# função da razão de chances com objeto na saída
razao <- function(X, nivel = 0.95) {
  # Calcula a razão de chances de tabela 2x2 e ...
  # ...
  # Adaptado de Christian, N. "Basic Programming-Lecture 3"
  # -----
  RC <- (X[1, 1]*X[2, 2])/(X[1, 2]*X[2, 1])
  ep.logRC <- sqrt(sum(1/X))
  alfa <- 1 - nivel
  IC.inf <- exp(log(RC) - qnorm(1 - alfa/2) * ep.logRC)
  IC.sup <- exp(log(RC) + qnorm(1 - alfa/2) * ep.logRC)
  cat("Razão de chances = ", RC, "\n",
      "Intervalo com ", nivel*100, "% de confiança = (",
      IC.inf, ", ", IC.sup, ") \n", sep="")
}
# diferentes abordagens de saída de objeto
# RC
#return(RC)
#invisible(RC)
saída <- list(RChance = RC, IC = c(IC.inf, IC.sup), confiança = nivel)
return(saída)
```

Acrecentar ao código anterior

Estatística Computacional I - 2020 210

✓ Execução da função e criação de objeto:

```
> # resultados da função como objeto
> razao.obj <- razao(tabela)
Razão de chances = 1.832054
Intervalo com 95% de confiança = (1.440042, 2.33078)
> razao.obj$RC
[1] 1.832054
> razao.obj$IC
[1] 1.440042 2.330780
> razao.obj$confiança
[1] 0.95
```

Estatística Computacional I - 2020 211

✓ Para imprimir texto, use os comandos `cat` ou `print`

✓ `cat`:

- Válido apenas para nomes e vetores (lógicos, inteiros, reais, complexos, caracteres).
- Na prática, converte argumentos em caracteres e concatena
- Pode ser usado para redirecionar saída para arquivo

✓ `print`:

- É função genérica que pode ser definida em implementação específica

212

Estatística Computacional I - 2020

✓ Comando `cat`:

```
> # concatena e não pula linha
> cat("a", "b", "c")
a b c
> # concatena e separa com espaço
> cat("a", "b", "c", fill = TRUE)
a b c
> # adiciona linha
> cat("a", "b", "c", "\n")
a b c
> # especifica largura da string para inserção de nova linha
> cat("a", "b", "c", fill = 2)
a
b
c
> cat("a", "b", "c", fill = 2, sep = "")
ab
c
> cat("a", "b", "c", fill = 2)
a
b
c
> # separa argumentos com outro separador
> cat("a", "b", "c", sep = "/", fill = TRUE)
a/b/c
```

213

Estatística Computacional I - 2020

✓ Comando `cat` – continuação:

```
> # imprime direto em arquivo
> cat(''Pacotes'', ''Estatísticos'', file = "arquivo.txt", sep = "\n")
> # grava incrementalmente no arquivo
> cat(''PACOTES'', file = "arquivo.txt", append = TRUE)
> cat(''ESTATÍSTICOS'', file = "arquivo.txt", append = TRUE)
```

214

Estatística Computacional I - 2020

✓ Comando `print`:

```
> # print
> a <- "Essa é uma string"
> b <- print(paste(a, 1))
[1] "Essa é uma string 1"
> # sem aspas
> print(b, quote = FALSE)
[1] Essa é uma string 1
> # qte de dígitos a ser apresentada
> c <- 10.4678
> print(c, digits = 3)
[1] 10.5
> # em número, NA pode ser impresso com valor especial
> d <- c(1, 2, NA, 4, NA, 6, 7)
> print(d, na.print = "-999")
[1] 1 2 -999 4 -999 6 7
> # Strings podem ser alinhadas à direita
> f <- c("um", "dois", "três")
> print(f)
[1] "um"   "dois" "três"
> print(f, right = TRUE)
[1] "um"   "dois" "três"
```

215

Estatística Computacional I - 2020



✓ Comando print – data frames:



```
> # print data frames
> df <- data.frame(a = c(1, 2, 3, 4, 5), b = c('a', 'b', 'c', 'd', 'e'),
+ c = c(10L, 20L, 30L, 40L, 50L))
> print(df)
  a b c
1 1 a 10
2 2 b 20
3 3 c 30
4 4 d 40
5 5 e 50
> print(df, row.names = F)
a b c
1 a 10
2 b 20
3 c 30
4 d 40
5 e 50
> # imprimindo em um arquivo
> sink("saída.txt")
> print(df)
> sink()
```

Estatística Computacional I - 2020

216



• Cálculo de estatística genérica acumulada



```
# Generalização de estatística acumulada
fun.acum <- function(vet, fun = median, ...) {
  # -----
  # Cálculo de mediana média de vetor
  # vet: vetor com observações
  # fun: comando que se deseja acumular
  # fun: função a ser aplicada
  # Adaptado de Gardener, M. Beginning R: The statistical programming
  # language. John Wiley & Sons, 2012.
  # -----
  saída <- sapply(seq_along(vet), function(x) fun(vet[1:x]))
  cat('\n', deparse(substitute(fun)), 'de', deparse(substitute(vet)), '\n')
  print(saída)
} # Fim
```

- ✓ ...: permite a introdução de outras instruções do comando em fun
- ✓ substitute: retorna a expressão
- ✓ deparse: transforma expressão em string

Estatística Computacional I - 2020

218



✓ Resultados de fun.acum:



```
> fun.acum(amostra)
median de amostra
[1] 0.5487370 1.2564124 0.5487370 0.3441432 0.5487370 0.5502616 0.5517862
[8] 1.0376944 1.5236027 1.0730578 1.5236027 1.0730578 0.6225129 0.6176420
[15] 0.6225129 0.6912014 0.6225129 0.6176420 0.6127711 0.6110223 0.6092735
[22] 0.5805298 0.6092735 0.5805298 0.6092735 0.5805298 0.5517862 0.5502616
[29] 0.5487370 0.5205005
> fun.acum(amostra, mean, na.rm = T)
mean de amostra
[1] 0.5487370 1.2564124 0.8841248 0.6654010 0.6426781 1.1212309 1.3341062
[8] 1.3577933 1.4375167 1.3560164 1.5002057 1.4021441 1.2963314 1.2475056
[15] 1.2149979 1.3352515 1.2699043 1.2052816 1.1563514 1.1289975 1.0760831
[22] 1.0394352 1.0328215 0.9947037 1.0136499 0.9847223 0.9561155 0.9228395
[29] 0.8926699 0.8793230
> # desvio padrão acumulado
> fun.acum(amostra, fun = sd)
> # produtorio acumulado
> Out <- fun.acum(amostra, fun = prod)
> out
> prod(amostra)
```

Estatística Computacional I - 2020

219



• Mensagem e aguarda ação de usuário



```
# Mostra mensagem e aguarda usuário - estatística acumulada
fun.acum.usr <- function(vet) {
  # -----
  # Cálculo de mediana média de vetor
  # vet: vetor com observações
  # -----
  # fun: função a ser aplicada
  nome <- readline(prompt = "Que função você quer aplicar?: ")
  # transforma string em comando
  fun <- eval(parse(text = nome))
  # calcula estatística acumulada
  saída <- sapply(seq_along(vet), function(x) fun(vet[1:x]))
  cat('\n', nome, 'de', deparse(substitute(fun)), '\n')
  print(saída)
} # Fim
```

- ✓ eval: avalia a expressão
- ✓ parse: transforma string em expressão

Estatística Computacional I - 2020

220

Verificação de Argumentos

- Pode ser necessário verificar se os argumentos apropriados foram inseridos
 - ✓ Quando valor de argumento não for válido:
 - Parada na execução e mensagem de erro.

221

Estatística Computacional I - 2020

- Comandos:
 - ✓ missing:
 - testa se valor foi especificado como argumento da função
 - TRUE: valor não foi especificado
 - ✓ stop:
 - Parar a execução e imprime mensagem de erro
 - ✓ warning:
 - Gera mensagem de aviso
 - ✓ message:
 - Gera mensagem de diagnóstico

222

Estatística Computacional I - 2020

✓ stopifnot:

- Se algum dos argumentos não for TRUE, então comando stop é executado
- É produzida uma mensagem de erro, indicando o primeiro elemento da lista de argumentos que não for TRUE
- Comando stop permite que seja fornecido uma mensagem informativa de erro
- Comando stopifnot requer codificação menor

223

Estatística Computacional I - 2020

- Função com mensagens de erro e de aviso

```
# função da razão de chances com mensagem de avisos
razao <- function(X, nivel = 0.95) {
  # Verificação dos argumentos
  stopifnot(!missing(X), is.matrix(X), dim(X) == c(2,2), X>0)
  # Mensagem se quantidade esperada de qquer célula < 5
  qte.esp <- (apply(X, 1, sum) %o% apply(X, 2, sum))/sum(X)
  if(any(qte.esp < 5)) warning("Qte esperada de célula < 5")
  # Calcula razão de chances e IC assintótico
  RC <- (X[1, 1]* X[2, 2])/(X[1, 2]*X[2, 1])
  ep.logRC <- sqrt(sum(1/X))
  alfa <- 1 - nivel
  IC.inf <- exp(log(RC) - qnorm(1 - alfa/2) * ep.logRC)
  IC.sup <- exp(log(RC) + qnorm(1 - alfa/2) * ep.logRC)
  cat("Razão de chances = ", RC, "\n",
      "Intervalo com ", nivel*100, "% de confiança = (",
      IC.inf, ", ", IC.sup, ") \n", sep="")
  # diferentes abordagens de saída de objeto
  # RC
  #return(RC)
  #invisible(RC)
  saida <- list(RChance = RC, IC = c(IC.inf, IC.sup), confianca = nivel)
  return(saida)
}
```

224

Estatística Computacional I - 2020

 ✓ Operador %○%:
– Produto entre vetores

```
> # produto entre vetores
> 1:3 %% 1:3
[,1] [,2] [,3]
[1,] 1 2 3
[2,] 2 4 6
[3,] 3 6 9
> # tabela - contagens marginais
> apply(tabela, 1, sum)
Placebo Aspirina
11034 1103
> apply(tabela, 2, sum)
Sim Não
293 21778
> # Quantidade esperada por célula
> (apply(tabela, 1, sum) %% apply(tabela, 2, sum))/sum(tabela)
Sim Não
Placebo 146.4801 10887.52
Aspirina 146.5199 10890.48
> sum((apply(tabela, 1, sum) %% apply(tabela, 2, sum))/sum(tabela))
[1] 22071
```

Estatística Computacional I - 2020 225

 ✓ Resultados da função razao:

```
> razao(tabela)
Razão de chances = 1.832054
Intervalo com 95% de confiança = (1.440042, 2.33078)
$RChance
[1] 1.832054

$IC
[1] 1.440042 2.330780

$Confianca
[1] 0.95
> # célula como contagem zerada
> Y <- matrix(c(0, 104, 10845, 10933), nrow = 2,
+ dimnames = list("Grupos" = c("Placebo", "Aspirina"),
+ "Infarto do Miocárdio" = c("Sim", "Não")))
> Y
Infarto do Miocárdio
Grupos Sim Não
Placebo 0 10845
Aspirina 104 10933
> razao(Y)
Erro: X > 0 are not all TRUE
```

Estatística Computacional I - 2020 226

 ✓ Resultados da função razao:
– Valor esperado de célula < 5

```
> # valor esperado de célula < 5
> Z <- matrix(c(1, 9, 4, 95 -9), nrow = 2,
+ dimnames = list("Grupo" = c("Controle", "Tratamento"),
+ "Doença" = c("Sim", "Não")))
> Z
      Doença
Grupo   Sim Não
Controle 1 4
Tratamento 9 86
> qte.esp <- (apply(Z, 1, sum) %% apply(Z, 2, sum))/sum(Z)
> qte.esp
      Sim Não
Controle 0.5 4.5
Tratamento 9.5 85.5
> razao(Z)
Razão de chances = 2.388889
Intervalo com 95% de confiança = (0.240378, 23.7409)
```

Estatística Computacional I - 2020 227

 Comando source

- Carregamento de objetos criados pelo usuário

```
> # Armazenamento de objetos dessa sessão (parcial)
> dump(c("tabela", "razao"), file = "minhas-texto.R", append = T)
> # Remoção de todos os objetos
> rm(list = ls())
> # Carregamento dos objetos
> source("minhas-texto.R")
> # objetos ativos
> ls()
[1] "amostra"       "amostra2"       "mediana.acum"  "razao"          "resistencia"
[6] "tabela"
```

Estatística Computacional I - 2020 228

Anotações

- Anote a função de maneira que consiga lembrar-se o que está acontecendo
 - ✓ O nome dos objetos deve facilitar o entendimento de seu código

229

Estatística Computacional I - 2020

✓ Exemplo – barplot com barras de erros:

```
> # anotações
> # Barplot com barras de erro.
> # Adaptado de Gardner, M. Beginning R: The statistical programming language.
> # John Wiley & Sons, 2012.
> # dados: denominação do conjunto de interesse
> dados <- iris[-5]
> # Cálculos por coluna
> # médias
> media <- apply(dados, 2, mean, na.rm = TRUE)
> # desvios padrão
> dp <- apply(dados, 2, sd, na.rm = TRUE)
> # soma
> tot <- apply(dados, 2, sum, na.rm = TRUE)
> # Cálculo qte. de observações (length não aceita na.rm=T).
> n <- media/tot
> # Cálculo dos erros padrão
> ep <- dp/sqrt(n)
> # maior valor para escala do y
> max.y <- round(max(media + ep) + 0.5, 0)
> # Constrói gráfico e a escala do eixo y
> bp <- barplot(media, ylim = c(0,max.y))
> # adiciona barras dos erros
> arrows(bp, media + ep, bp, media - ep, length = 0.1, angle = 90, code = 3)
> # Se o eixo y ainda é pequeno, mude max.y para um valor maior
> # Fim
```

230

Estatística Computacional I - 2020

- Exemplo com dados = `íris[-5]`:

The figure shows a bar plot with four categories on the x-axis: Sepal.Length, Sepal.Width, Petal.Length, and Petal.Width. The y-axis ranges from 0 to 25. Each category has a grey bar representing the mean and a vertical error bar representing the standard deviation. The values are approximately: Sepal.Length ~ 5.5, Sepal.Width ~ 2.5, Petal.Length ~ 4.5, and Petal.Width ~ 1.5.

231

Estatística Computacional I - 2020

Funções bem Definidas

- Uma função é bem definida:
 - ✓ Quando valor retornado pela função é completamente determinado pelos argumentos da função.
 - Ex.: Não são necessárias variáveis globais para avaliar a função.
 - ✓ Não afeta cálculos posteriores
 - Ex.: usar `options` que é um comando que altera configurações globais

232

Estatística Computacional I - 2020

✓ Tem o código comentado

- Funções que são transparentes e bem documentadas código são mais confiáveis.

✓ Em geral, a primeira versão de uma função é para uso imediato.

- Posteriormente é possível refiná-la.

233

Estatística Computacional I - 2020

• Exemplo – Log-verossimilhança

```
> # Escrevendo funções confiáveis
>
> # armazena os defaults correntes
> defaults <- options()
> # função para cálculo da logverossimilhança
> logVeros.exp_ok <- function(lambda){
+ options(digits = 3)
+ LV <- sum(log(dexp(x, rate = lambda)))
+ LV
+ }
> # amostra exponencial, com lambda = 3, de tamanho 100
> x <- rexp(100, rate = 3)
> # x não está definido como argumento da função
> logVeros.exp_ok(3)
[1] 20.7
> # variância definida com muitos algarismos significativos .
> signif(var(x), 7)
[1] 0.0737
> # restaura os defaults
> options(defaults)
```

234

Estatística Computacional I - 2020

• Exemplo – Log-verossimilhança

✓ Função melhor escrita

```
> # Função para fornecer LV com 3 dígitos
> logVeros.exp_melhor <- function(x, theta) {
+ LL <- sum(log(dexp(x, rate=theta)))
+ print(LL, digits=3)
+ }
> # Não está restrita aos vetores denominados x
> logVeros.exp_melhor(x, 3)
[1] 20.7
```

235

Estatística Computacional I - 2020

Extração de Componentes da Função

Comando	Descrição
<code>body</code>	Obtém ou define o corpo de função
<code>formals</code>	Obtém ou define os argumentos formais de função
<code>args</code>	Exibe os nomes dos argumentos de função e seus correspondentes defaults
<code>nargs</code>	Fornece o número de argumentos fornecidos à função (usado dentro da função)
<code>match.call</code>	Executa função com todos seus argumentos especificados por seus nomes completos (usado dentro da função)
<code>update</code>	Atualiza modelo ajustado

236

Estatística Computacional I - 2020

• Argumentos formais de função:

- ✓ Argumentos incluídos na definição da função
- ✓ Comando `formals`:

- Retorna lista com argumentos formais de função

```
> formals(lm)
$formula
$x
[1] FALSE
$y
[1] FALSE
$qr
[1] TRUE
$singular.ok
[1] TRUE
$contrasts
NULL
$offset
$...
$...
```

237

Estatística Computacional I - 2020

✓ Comando args:

- Exibe os nomes dos argumentos e valores default correspondentes de função

```
> args(lm)
function (formula, data, subset, weights, na.action, method = "qr",
         model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE,
         contrasts = NULL, offset, ...)
NULL
```

238

Estatística Computacional I - 2020

✓ Comando body:

- Código da função

```
> body(lm)
{
  ret.x <- x
  ret.y <- y
  cl <- match.call()
  mf <- match.call(expand.dots = FALSE)
  m <- match(c("formula", "data", "subset", "weights", "na.action",
              "offset"), names(mf), 0L)
  mf <- mf[c(1L, m)]
  mf$drop.unused.levels <- TRUE
  mf[[1]] <- quote(stats::model.frame)
  mf <- eval(mf, parent.frame())
  if (method == "model.frame")
    return(mf)
  else if (method != "qr")
    warning(gettextrf("method = '%s' is not supported. Using 'qr'", method),
            domain = NA)
  mt <- attr(mf, "terms")
  y <- model.response(mf, "numeric")
  ...
}
```

239

Estatística Computacional I - 2020

✓ Comando body – Exemplo:

- Permite também atribuição de código

```
> # exemplo
> f <- function(x) { x^5 }
> body(f)
{
  x^5
}
> f(8)
[1] 32768
> # pode-se atribuir função com comando body
> e <- expression(y <- x^2, return(y))
> body(f) <- as.call(c(as.name("("), e))
> body(f)
{
  y <- x^2
  return(y)
}
> f(8)
[1] 64
```

240

Estatística Computacional I - 2020



✓ Comando nargs:

- Exibe número de argumentos fornecidos
- Atua no código da função

```
> # comando nargs
> fun <- function(x = 1, y = 2, z = FALSE, ...) {nargs()}
> fun()
[1] 0
> fun(2, 3)
[1] 2
> fun(w)
[1] 1
> fun(2, 3, TRUE, w)
[1] 4
```



Estatística Computacional I - 2020

241



✓ Capturando a execução da função

- sys.call: Exibe o que o usuário digitou
- match.call: exibe apenas os argumentos

```
> # comando match.call e sys.call
> funcao <- function(abc = 1, def = 2, ghi = 3) {
+   list(sys = sys.call(), match = match.call())
+ }
> funcao(d = 2, 2)
$sys
funcao(d = 2, 2)

$match
funcao(abc = 2, def = 2)
```



Estatística Computacional I - 2020

242



✓ Comando update:

- Reajusta modelo, modificando argumentos

```
> mod <- lm(mpg ~ wt, data = mtcars)
> mod
Call:
lm(formula = mpg ~ wt, data = mtcars)
Coefficients:
(Intercept)          wt
            37.285       -5.344
> update(mod, formula = . ~ . + cyl)
Call:
lm(formula = mpg ~ wt + cyl, data = mtcars)
Coefficients:
(Intercept)          wt          cyl
            39.886       -3.191      -1.508
> update(mod, formula = . ~ . + cyl - wt)
Call:
lm(formula = mpg ~ cyl, data = mtcars)
Coefficients:
(Intercept)          cyl
            37.885       -2.876
```



Estatística Computacional I - 2020

243



✓ Comandos edit e fix:

- fix: edita objeto e atribui nova versão a ele mesmo
- edit: necessário atribuir nova versão a um objeto

```
> # comandos edit e fix
> source("minhas-texto.R")
> fix(tabela)
> tabela
  Sim  Não
Placebo 109 10845
Aspirina 104 10933
> source("minhas-texto.R")
> tabela.new <- edit(tabela) # reescrever valor de 189
> tabela <- tabela.new
> rm(tabela, tabela.new)
> source("minhas-texto.R")
> tabela
  Infarto do Miocárdio
Grupos  Sim  Não
Placebo 189 10845
Aspirina 104 10933
```



Estatística Computacional I - 2020

244

Função Recursiva

- Função que executa a si mesma
- Recursão
 - ✓ Técnica que divide problema em subproblemas menores e mais simples

245

Estatística Computacional I - 2020

✓ Exemplo – fatorial:

```
> # Exemplo - factorial
> factorial <- function(n) {
+ if(n == 1) {
+ 1
+ } else {
+ n * factorial(n-1)
+ }
+ }
> factorial(10)
[1] 3628800
> factorial(10)
[1] 3628800
```

246

Estatística Computacional I - 2020

✓ Exemplo – sequência de Fibonacci:

$$f(x) = \begin{cases} 0 & , x = 0 \\ 1 & , x = 1 \\ f(x-1) + f(x-2) & \forall x \in \mathbb{N}, x > 1 \end{cases}$$

```
> # Exemplo - Fibonacci
> fibonacci <- function(n) {
+ # Calcula f(n)
+ if(n <= 1) {
+ return(n)
+ } else {
+ return(fibonacci(n - 1) + fibonacci(n - 2))
+ }
+ }
> fibonacci(0)
[1] 0
> fibonacci(4)
[1] 3
```

247

Estatística Computacional I - 2020

✓ Geração de sequência de n termos:

```
> # Função para calcular sequencia de n termos
> seq.fibonacci <- function() {
+ # Gera sequencia de Fibonacci com n termos
+
+ # entrada digitada no console pelo usuário
+ ntermos <- as.integer(readline(prompt = "Quantos termos? "))
+ # verifica se quantidade de termos é válida
+ if(ntermos <= 0) {
+ print("Entre um inteiro positivo")
+ } else {
+ print("Sequência de Fibonacci:")
+ for(i in 0:(ntermos - 1)) {
+ print(fibonacci(i))
+ }
+ }
+ }
> seq.fibonacci()
Quantos termos? 5
[1] "Sequência de Fibonacci"
[1] 0
[1] 1
[1] 1
[1] 2
[1] 3
```

Digite no console a quantidade de termos

248

Estatística Computacional I - 2020



Intervalo de Confiança



- Seja uma amostra aleatória X_1, X_2, \dots, X_n , com média \bar{x} e desvio padrão s .

✓ Intervalo com $(1 - \alpha)100\%$ de confiança para μ

$$\bar{x} \pm t_{1-\alpha/2, n-1} \frac{s}{\sqrt{n}}$$

- $t_{(1-\alpha/2); n-1}$: percentil $1 - \alpha/2$ de uma t com $n - 1$ graus de liberdade

✓ Esse intervalo de confiança:

- É exato, se amostra provém de população normal
- É aproximadamente correto para qualquer população se a mostra for grande

249

Estatística Computacional I - 2020



- Perguntas:

✓ Quão grande deve ser o tamanho amostral para a aproximação ter-se uma boa aproximação?

✓ Como esse tamanho amostral é afetado pela forma da distribuição?

- Solução:

✓ Executar simulações para diferentes distribuições e tamanhos amostrais



250

Estatística Computacional I - 2020



✓ Função para simulação de IC's:



```
# Função para Simulação de IC's

sim.IC.t <- function(gera.VA, n, nsim = 1000, alfa = 0.05, mi, ...){
  # -----
  # Argumentos da função:
  #   gera.VA - função geradora de valores aleatórios de VA
  #   n       - tamanho da amostra (pode ser um vetor)
  #   nsim    - quantidade de iterações (default: 1000)
  #   alfa    - nível de confiança 100(1 - alfa)%
  #   mi      - média populacional
  #   ...
  #   argumentos adicionais de gera.VA
  #
  # Adaptado de Christian, N. "Basic Programming-Lecture 3"
  # -----
  
  # Verifica os argumentos da função
  stopifnot(!missing(gera.VA), is.function(gera.VA), !missing(n),
            !missing(mi))
  # Cria matriz de resultados
  resultados <- matrix(nrow = length(n), ncol = 1,
                        dimnames = list(n, deparse(substitute(gera.VA))))
  # Loop para diferentes tamanhos amostrais
  ...
  
  # continua próximo slide
}
```

Estatística Computacional I - 2020



✓ Função para simulação de IC's – continuação:



```
# Continuação
```

```
...
# Loop para diferentes tamanhos amostrais
for(i in seq_along(n)) {
  CP <- array(dim = nsim)
  # Loop de simulação
  for(j in 1:nsim) {
    # Gera amostra aleatória
    x <- gera.VA(n[i], ...)
    # Calcula Intervalo de Confiança t
    xbarra <- mean(x)
    inferior <- xbarra - qt(1-alfa/2, n[i]-1)*sd(x)/sqrt(n[i])
    superior <- xbarra + qt(1-alfa/2, n[i]-1)*sd(x)/sqrt(n[i])
    CP[j] <- ifelse(inferior < mi & mi < superior, 1, 0)
  }
  resultados[i,] <- mean(CP)
}
} # end for
return(resultados)
} # end function
```

Estatística Computacional I - 2020

252



✓ Exemplo de simulação:

```
> set.seed(666)
> sim.IC.t(rnorm, n = c(10, 25, 50, 100), mi = 0, mean = 0)
  rnorm
10 0.953
25 0.953
50 0.954
100 0.952
> sim.IC.t(rexp, n = c(10, 25, 50, 100), mi = 1, rate = 1)
  rexp
10 0.908
25 0.920
50 0.940
100 0.940
```



Estatística Computacional I - 2020

253



✓ Comparação entre distribuições:

```
> # Aplica função a distribuição normal, uniforme e gama
> set.seed(666)
> startTime <- proc.time() [3]
> corridal1 <- sim.IC.t(rnorm, n = c(10, 25, 50, 100), mi = 0, mean = 0)
> corrida2 <- sim.IC.t(runif, n = c(10, 25, 50, 100), mi = 0.5, min = 0, max = 1)
> corrida3 <- sim.IC.t(rgamma, n = c(10, 25, 50, 100), mi = 2, shape = 1,
scale = 2)
> cbind(corridal1, corrida2, corrida3)
  rnorm runif rgamma
10 0.953 0.934 0.884
25 0.953 0.946 0.925
50 0.954 0.956 0.935
100 0.952 0.946 0.942
> tempoDecor <- proc.time() [3] - startTime
> cat("Tempo decorrido:", floor(tempoDecor/60), "min", tempoDecor%%60, "seg\n")
Tempo decorrido: 0 min 0.74 seg
```



Estatística Computacional I - 2020

254

Referências



Bibliografia Recomendada

- ALBERT, J.; RIZZO, M. *R by Example*. Springer, 2012.
- CHRISTIAN, N. *Basic Programming*, Lecture Notes
- DALGAARD, P. *Introductory statistics with R*. Springer, 2008.
- KLEIBER, C.; ZEILEIS, A. *Applied econometrics with R*. Springer, 2008.
- GARDENER, M. *Beginning R: The statistical programming language*. John Wiley & Sons, 2012.



Estatística Computacional I - 2020

287