

# 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



## Programação em R



- Estruturas de controle
  - √ Condicional
  - √ Loop
- Família Apply
  - √ Funções que evitam loops
- Cálculos vetorizados
- Funções
- Operações com matrizes

Estatística Computacional I - 2020

4

## Estruturas de Controle



### Comando if ... else



- Comando não é vetorizado  
✓ Opera com vetores lógicos unitários
- Sintaxe:

```
if (cond=T) {comando1} else {comando2}
```

```
> i = 1
> if(i==0) {
+   print("É")
+ } else {
+   print("Não é")
+ }
[1] "Não é"
```

✓ Evite inserir linhas entre }else

Estatística Computacional I - 2020

8



- Exemplo:

✓ Cálculo vetorizado de raiz quadrada

```
> x <- rnorm(20)
> sqrt(x)
[1] 0.8534541 0.8079840      NaN      NaN      NaN      NaN      NaN
[8] 0.1262007 0.3678077      NaN      NaN      NaN 1.5423946 1.1010460
[15] 0.1482457 0.6212743 1.3166528 0.8976863 0.6757055 0.6311478
Warning message:
In sqrt(x) : NaNs produzidos
> y <- rexp(20)
> sqrt(x)
[1] 0.4186259 1.2099860 2.4738528 0.6503102 0.8785719 1.3200691 0.6887323
[8] 1.2808162 1.1824393 0.4662088 1.0204777 1.1666715 1.1432258 0.3892436
[15] 0.7657090 1.4997770 1.2001891 0.8584922 0.9249117 1.9980625
```

✓ Verificação de condição

```
> if(is.numeric(x) & min(x) > 0) {sx <- sqrt(x)} else {
+ stop("x deve ser numérico e todos seus componentes positivos")}
Erro: x deve ser numérico e todos seus componentes positivos
```

Estatística Computacional I - 2020

9



- Comando if – fluxograma

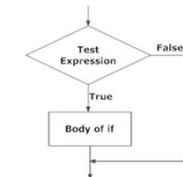


Fig: Operation of if statement

- Comando if ... else – fluxograma

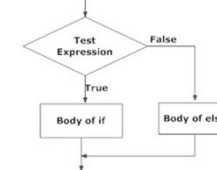


Fig: Operation of if...else statement

Estatística Computacional I - 2020

10



## Comandos if ... else aninhados



- Executa bloco de códigos com mais de duas alternativas

- Sintaxe:

```
if(cond1=T) {
  comando1
}else if(cond2=T) {
  comando2
}else if(cond3=T) {
  comando3
}else {
  comando4
}
```

Estatística Computacional I - 2020

11



- Exemplo:

✓ Verificação de condição de número

```
> x <- 0
> if (x < 0) {
+ print("Número negativo")
+ } else if (x > 0) {
+ print("Número positivo")
+ } else
+ print("Zero")
[1] "Zero"
```

Estatística Computacional I - 2020

12



- Exemplo:

✓ Fatorial de um número

```
> num <- as.integer(readline(prompt="Entre um número: "))
Entre um número: 5
> fatorial <- 1
> # verifica se o número é positivo, negativo ou zero
> if(num < 0) {
+ print("Não é definido fatorial para número negativo")
+ } else if(num == 0) {
+ print("O fatorial de 0 é 1")
+ } else {
+ for(i in 1:num) {
+ fatorial = fatorial * i
+ }
+ print(paste("O fatorial de ", num ,"é",fatorial))
+ }
[1] "O fatorial de  5 é 120"
> factorial(5)
[1] 120
```

Estatística Computacional I - 2020

13



- Exemplo

✓ Função para decidir se número é par ou ímpar

```
# Par ou ímpar - função
par.impar <- function(x){
  # verifica se o número é um decimal comparando x - round(x)
  # se for decimal retorna NA (par-ímpar não faz sentido para decimais)
  if (abs(x - round(x)) > 1e-7){
    return(NA)
  }
  # se o número for divisível por 2 (resto da divisão zero) retorna "par"
  # caso contrário, retorna "ímpar"
  if (x %% 2 == 0){
    return(paste("O número", x, "é par"))
  }else{
    return(paste("O número", x, "é ímpar"))
  }
}
```

Estatística Computacional I - 2020

14



## ✓ Testando a função par.impar

```
> par.impar(4); par.impar(5); par.impar(2.1)
[1] "O número 4 é par"
[1] "O número 5 é ímpar"
[1] NA
> > par.impar(1:5)
[1] "O número 1 é ímpar" "O número 2 é ímpar" "O número 3 é ímpar"
[4] "O número 4 é ímpar" "O número 5 é ímpar"
Warning messages:
1: In if (abs(x - round(x)) > 1e-07) { :
  a condição tem comprimento > 1 e somente o primeiro elemento será usado
2: In if (x%%2 == 0) { :
  a condição tem comprimento > 1 e somente o primeiro elemento será usado
```

- Comandos `if` e `if...else` não são vetorizados.
- Alternativa:
  - Comando `ifelse`.

Estatística Computacional I - 2020

15



## Comando `ifelse`

- Comando opera vetorizado
- Sintaxe:
 

```
ifelse(teste, valor_T, valor_F)
```

```
> ifelse(c(TRUE, FALSE, FALSE, TRUE), 1, -1)
[1] 1 -1 -1 1
```

Estatística Computacional I - 2020

16



## • Exemplos:

### ✓ Verificação vetorizada de condição

```
> xis <- 1:10
> ifelse(xis < 5 | xis > 8, xis, 0)
[1] 1 2 3 4 0 0 0 0 9 10
```

### ✓ Função par-ímpar

```
# função par-ímpar - comando ifelse
par.impar.vet <- function(x){
  # se x for decimal, retorna NA, se não for, retorna ele mesmo (x)
  x <- ifelse(abs(x - round(x)) > 1e-7, NA, x)
  # se x for divisível por 2, retorna 'par', se não for, retorna ímpar
  ifelse(x %% 2 == 0, "par", "ímpar")
}
> par.impar.vet(1:5)
[1] "ímpar" "par" "ímpar" "par" "ímpar"
> par.impar.vet(c(1:5, 1.1))
[1] "ímpar" "par" "ímpar" "par" "ímpar" NA
```

Estatística Computacional I - 2020

17



## • Exemplo - iris

### ✓ Variáveis *dummies*:


- Indicadoras dos níveis de variável categórica

```
> # indicadora setosa
> ind.setosa <- ifelse(iris$Species == "setosa", 1, 0)
> # indicadora versicolor
> ind.versicolor <- ifelse(iris$Species == "versicolor", 1, 0)
> #tabela com os valores convertidos
> ind <- data.frame(iris$Species, ind.setosa, ind.versicolor)
> unique(ind)
      iris.Species ind.setosa ind.versicolor
1      setosa          1          0
51  versicolor          0          1
101 virginica          0          0
```


### ✓ Criação de dummies

```
> mat <- model.matrix(~iris$Species-1)
> mat <- as.data.frame(mat)
> names(mat) <- unique(iris$Species)
> unique(mat)
      setosa versicolor virginica
1          1          0          0
51         0          1          0
101        0          0          1
```

Estatística Computacional I - 2020




## Valores Especiais




- Cálculo com valores especiais
  - ✓ Inf - infinito
  - ✓ NaN – valor indefinido
  - ✓ NA – dado ausente (*missing value*)

19

Estatística Computacional I - 2020



## • Inf – infinito



```
> 2/0
[1] Inf
> 4 - Inf
[1] -Inf
```


✓ Verificação se valor é finito

```
> is.finite(10^(305:310))
[1] TRUE TRUE TRUE TRUE FALSE FALSE
> 4 - Inf
[1] -Inf
```


✓ O R considera infinito tudo maior do que o maior número que um computador pode ter (na maioria das máquinas, isso é aproximadamente  $1,8 \times 10^{308}$ )

20

Estatística Computacional I - 2020



## • NaN – valor indeterminado




```
> Inf / Inf
[1] NaN
> Inf - Inf
[1] NaN
> Inf - 4
[1] NaN
```

✓ Verificação se valor é indefinido


```
> 0:4/0
[1] NaN Inf Inf Inf Inf
> is.nan(0:4/0)
[1] TRUE FALSE FALSE FALSE FALSE
> is.infinite(0:4/0)
[1] FALSE TRUE TRUE TRUE TRUE
> is.finite(c(0:4, NaN))
[1] TRUE TRUE TRUE TRUE TRUE FALSE
```

21

Estatística Computacional I - 2020



## • NA – *missing value*



```
> x <- NA
> x + 4
[1] NA
> log(x)
[1] NA
> mean(c(4, 8, NA))
[1] NA
```

✓ Verificação de *missing value*

```
> is.na(x)
[1] TRUE
> is.na(NaN - 4)
[1] TRUE
```

✓ Constantes que geram NA do tipo apropriado

- NA\_integer\_, NA\_real\_,  
NA\_complex\_ e NA\_character\_

22

Estatística Computacional I - 2020



## • Verificação de valores especiais



Função	Inf	- Inf	NaN	NA
is.finite()	FALSE	FALSE	FALSE	FALSE
is.infinite()	TRUE	TRUE	FALSE	FALSE
is.nan()	FALSE	FALSE	TRUE	FALSE
is.na()	FALSE	FALSE	TRUE	TRUE

Estatística Computacional I - 2020

23



## Comandos usados em *loops*



- for
- while
- Família apply
- repeat
  - √ break
  - √ next

Estatística Computacional I - 2020

24



## Comando **for**



- Loops são controlados por um vetor
  - √ Em cada iteração, um valor no vetor é atribuído a uma variável
  - √ Geralmente, o número de iterações é definido pela quantidade de valores do vetor
  - √ Processamento se dá na mesma ordem
- Sintaxe:
 

```
for(variável em sequência) {
    comandos
}
```

Estatística Computacional I - 2020

25



## • Exemplos:

### √ Sequências de letras

```
> for(i in 1:5){
+ print(letters[i])
+ }
[1] "a"
[1] "b"
[1] "c"
> for(letra in letters[1:5]){
+ print(letra)
+ }
[1] "a"
[1] "b"
[1] "c"
```

### √ Cálculo de 10!

```
> # Cálculo de 10! usando loop for
> fat <- 1
> for(i in 1:10) {
+ fat <- fat*i
+ cat(i, fat, "\n")
+ }
> fat
```

Estatística Computacional I - 2020

26

## Comando for – fluxograma



Fig: operation of for loop

Estatística Computacional I - 2020

27

## Comando seq\_along

### • Cria vetor de índices do objeto

```

> # seq_along
> set.seed(123456)
> vet <- rnorm(10)
> # inteiros de 1 a 10
> seq_along(vet)
[1] 1 2 3 4 5 6 7 8 9 10
> # alternativa
> 1:length(vet)
[1] 1 2 3 4 5 6 7 8 9 10
  
```

Estatística Computacional I - 2020

28

## Loop com vetor vazio

✓ Loop é executado (incorreto)

```

> # vetor vazio
> x <- numeric(0)
> # loop é executado (incorreto)
> 1:length(x)
[1] 1 0
> for(i in 1:length(x)) print(i)
[1] 1
[1] 0
  
```

✓ Comando seq\_along

```

> # loop não é executado (correto)
> for(i in seq_along(x)) print(i)
  
```

– Loop não é executado

Estatística Computacional I - 2020

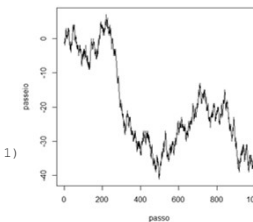
29

## Exemplo – Passeio aleatório

✓ A cada passo anda-se para a esquerda (+1) ou para a direita (-1) com probabilidades iguais

```

> # Passeio aleatório
> set.seed(1)
> # quantidade de passos
> n <- 1000
> # vetor para armazenar o passeio aleatório
> passeio <- numeric(n)
> # primeiro passo
> passeio[1] <- sample(c(-1, 1), 1)
> # demais passos
> for(i in 2:n){
+   passeio[i] <- passeio[i - 1] + sample(c(-1, 1), 1)
+ }
> passeio
> range(passeio)
[1] -41 7
> plot(passeio, type = "l", xlab = "passo")
  
```



Estatística Computacional I - 2020

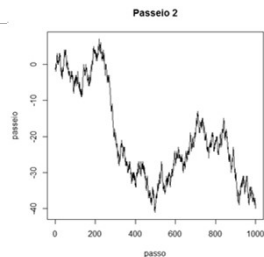
30



## • Exemplo – passeio aleatório

### ✓ Solução vetorizada

```
> # Passeio aleatório
> set.seed(1)
> passeio2 <- cumsum(sample(c(-1, 1), n, replace = TRUE))
> plot(passeio, type = "l", xlab = "passo", main = "Passeio 2")
> # verificação de igualdade dos vetores
> all.equal(passeio, passeio2)
[1] TRUE
```



Estatística Computacional I - 2020

31



## • Exemplo – média das colunas por linhas

### ✓ Conjunto de dados: iris

```
> # Cria objeto vazio
> media.linha <- NULL
> for(i in seq(along = iris[, 1])) {
+   # média por linhas do data frame
+   media.linha <- c(media.linha, mean(as.numeric(iris[i, 1:4])))
+ }
> head(media.linha)
[1] 2.550 2.375 2.350 2.350 2.550 2.850
```

✓ Não é a maneira mais rápida para inserção dos elementos do vetor

– (objeto “cresce” durante loop)

✓ Impacta substancialmente a performance do programa

Estatística Computacional I - 2020

32



## • Pré-alocação de espaço:

✓ Sempre que possível, crie objeto para armazenar resultados de cada iteração, antes de iniciar o loop.

## • Exemplo – Sequência de Fibonacci

✓  $F_1 = 0; F_2 = 1; F_3 = 1, F_4 = 2, F_5 = 3, \dots$

✓ Regra:

–  $F_1 = 0; F_2 = 1; \dots, F_i = F_{i-1} + F_{i-2}, \dots$

Estatística Computacional I - 2020

33



## • Sequência de Fibonacci – uso de for


### ✓ Pré-alocação de espaço

```
> n <- 9
> # vetor para armazenamento dos n resultados
> fib <- numeric(n)
> # condições iniciais
> fib[1] <- 0
> fib[2] <- 1
> # regra para i > 2
> for(i in 3:n){
+   fib[i] <- fib[i - 1] + fib[i - 2]
+ }
> fib
[1] 0 1 1 2 3 5 8 13 21
```


Estatística Computacional I - 2020

34







## ✓ Função com pré-alocação



```
# função do código com pré-alocação de espaço
fib <- function(n){
  # vetor para armazenamento dos resultados
  fib <- numeric(n)
  fib[1] <- 0      # condições iniciais
  fib[2] <- 1
  # cálculo dos valores de 3 a n
  for(i in 3:n){
    fib[i] <- fib[i - 1] + fib[i - 2]
  }
  return(fib)
}
```




## ✓ Função sem pré-alocação




```
# função do código sem pré-alocação de espaço
fib.sem <- function(n){
  fib <- 0      # condições iniciais
  fib <- c(fib, 1)
  # cálculo dos valores de 3 a n
  for(i in 3:n){
    fib <- c(fib, fib[i - 1] + fib[i - 2])
  }
  return(fib)
}
```

Estatística Computacional I - 2020

35



## ✓ Comando para verificar instalação de pacote




```
> # carregamento de pacote
> if(!require(microbenchmark)) install.packages("microbenchmark")
Carregando pacotes exigidos: microbenchmark
--- Please select a CRAN mirror for use in this session ---
tentando a URL
'https://mirrors.ebi.ac.uk/CRAN/bin/windows/contrib/3.3/microbenchmark_1.4-4.zip'
Content type 'application/zip' length 63743 bytes (62 KB)
downloaded 62 KB

package 'microbenchmark' successfully unpacked and MD5 sums checked


The downloaded binary packages are in
D:\Usuários\Lupercio\AppData\Local\Temp\RtmpOcAmBm\downloaded_packages
```

Estatística Computacional I - 2020

36



## • Comparação das implementações:




```
> # comparação das implementações
> library(microbenchmark) # ou require(microbenchmark)
> set.seed(666)
> microbenchmark(fib(5000), fib.sem(5000))
Unit: milliseconds
      expr      min       lq     mean   median      uq     max neval
fib(5000)  7.265189  7.628054  7.96130  7.934029  8.150326 10.84603  100
fib.sem(5000) 31.727421 32.887916 35.47633 33.839619 35.591521 54.40792  100
```


- ✓ Função `fib.sem` chega a ser cerca de 10 vezes mais lenta do que a função `fib`.
- ✓ Quanto maior o número de simulações, maior a queda no desempenho
- ✓ Também pode ser usado o comando `system.time`

Estatística Computacional I - 2020

37



## Comando while



- Sintaxe:
 

```
while(condição) {
  comandos
}
```
- Loops são controlados pela condição
  - ✓ Comandos do loop são executados se resultado do teste(condição) for TRUE
  - ✓ Loop é encerrado se resultado do teste for FALSE

Estatística Computacional I - 2020

38



## Exemplos:

### ✓ Sequência de números

```
> i <- 1
> while (i < 6) {
+ print(i)
+ i = i+1
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

### ✓ Cálculo de 10!

```
> i <- 10
> fat <- 1
> while(i > 1) {
+ fat <- i*fat
+ i <- i-1
+ cat(i, fat, "\n")
+ }
> fat
```

Estatística Computacional I - 2020

39



## Comando while – fluxograma

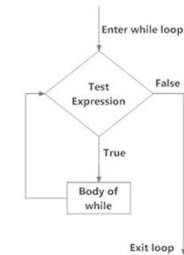


Fig: operation of while loop

Estatística Computacional I - 2020

40



## Sequência de Fibonacci – uso de while

```
# quantidade de termos digitada pelo usuário
ntermos <- as.integer(readline(prompt="Quantos termos? "))
n1 <- 0 # termos iniciais

n2 <- 1
conta <- 2
if(ntermos <= 0) {# verifica se qte. de termos é válida
  print("Entrar um número inteiro positivo")
} else {
  if(ntermos == 1) {
    print("Sequência de Fibonacci:")
    print(n1)
  } else {
    print("Sequência de Fibonacci:")
    print(n1)
    print(n2)
    while(conta < ntermos) {
      n.esimo = n1 + n2
      print(n.esimo)
      # atualização dos valores
      n1 = n2
      n2 = n.esimo
      conta = conta + 1
    } # fim while
  } # fim do 1o. else
} # fim do 2o. else
```

Estatística Computacional I - 2020

41



## Soma dos primeiros números naturais

### ✓ Sem fórmula – uso de while

```
# quantidade de termos digitada pelo usuário
num <- as.integer(readline(prompt = "Entre um número: "))
if(num < 0) {
  print("Entrar um número inteiro positivo")
} else {
  soma <- 0
  # uso de loop while para iterar até zero
  while(num > 0) {
    soma = soma + num
    num = num - 1
  }
  print(paste("A soma é", soma))
}
```

### ✓ Com fórmula

```
# quantidade de termos digitada pelo usuário
num <- as.integer(readline(prompt = "Entre um número: "))
if(num < 0) {
  print("Entrar um número inteiro positivo")
} else {
  soma = (num * (num + 1)) / 2;
  print(paste("A soma é", soma))
}
```

Estatística Computacional I - 2020

42

### Instruções break e next

- Break
  - √ Usado dentro do loop (while ou for) para interromper e desviar o fluxo de controle para fora do loop
  - √ Em loops aninhados, desvia do loop interno que estiver sendo executado
- Sintaxe:
 

```
if(condição) {
    break
}
```

43

### Exemplos:

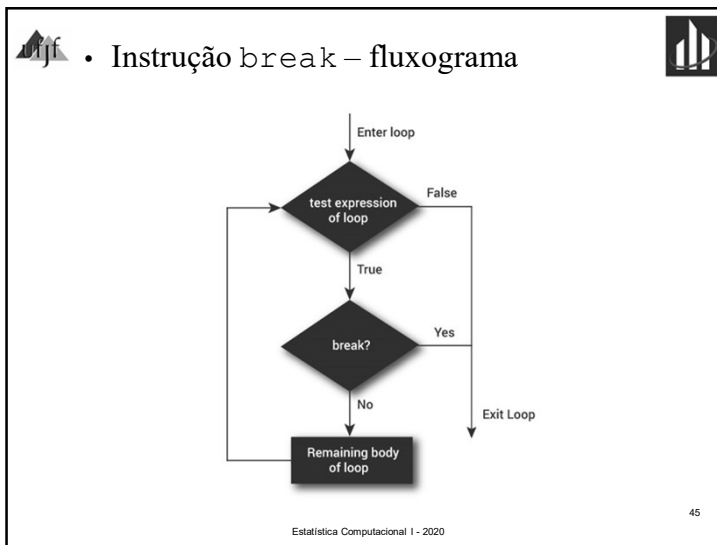
#### Sequência de números

```
# exemplo 1
x <- 1:5
for (valor in x) {
  if (valor == 3){
    break
  }
  print(valor)
}
```

#### Sequência aninhada

```
# exemplo 2
for (i in 1:10){
  for (j in 1:10){
    for (k in 1:10){
      cat(i, " ", j, " ", k, "\n")
      if (k == 5) break
    }
  }
}
```

44



### next

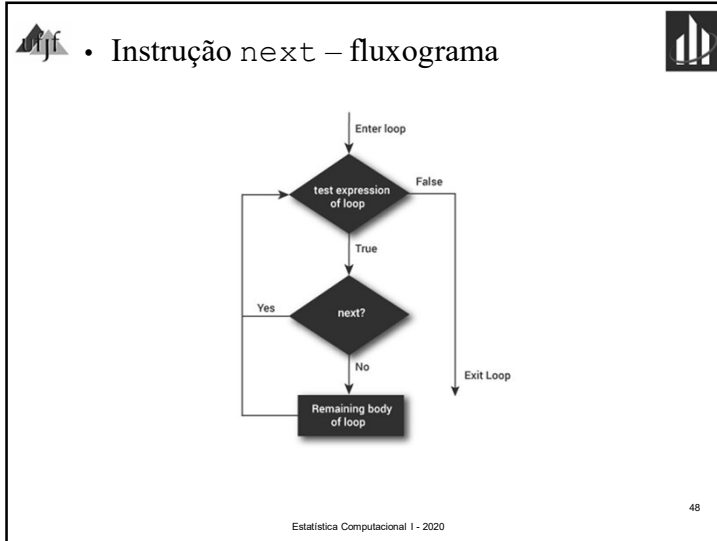
- √ Permite pular a iteração atual de um loop, mas não finaliza-o.

- Sintaxe:
 

```
if(condição) {
    next
}
```
- Exemplo:
 

```
# exemplo 3
> x <- 1: 4
> for (i in x) {
+   if (i == 2) next
+   print(i)
+ }
```

47



**Comando repeat**

- Loop é repetido até que seja especificado um **break**
- **Sintaxe:**

```
repeat {
  comandos
}
```
- ✓ **Necessária** uma segunda instrução para testar se o loop deve ser interrompido
- ✓ **CUIDADO:**
  - Não fazer isso resultará em um loop infinito

Estadística Computacional I - 2020

• Exemplos:

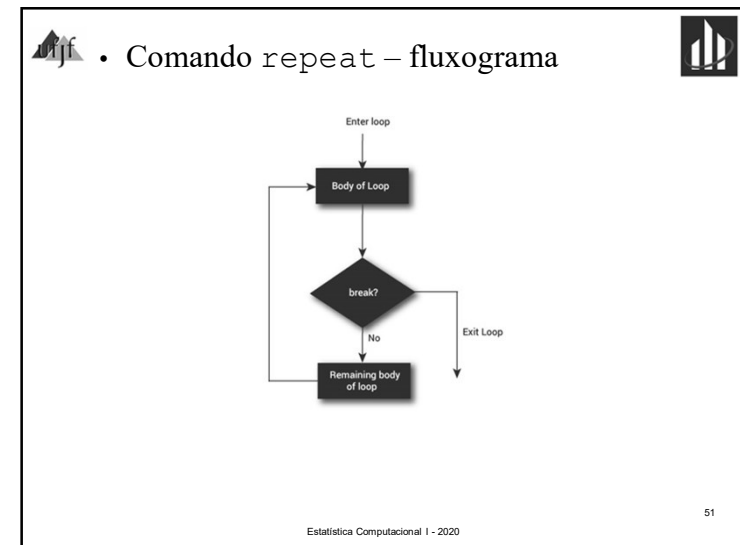
✓ Sequência de números

```
z <- 0
repeat {
  z <- z + 1
  print(z)
  if(z > 100) break
}
```

✓ Cálculo de 10!

```
i <- 1
fat <- 1
repeat {
  fat <- i * fat
  cat(i, fat, "\n")
  i <- i + 1
  if(i > 10) break
}
> fat
```

Estadística Computacional I - 2020





## • Instrução stop:

✓ Interrompe fluxo e imprime mensagem de erro

## • Exemplo:

```
> # instrução stop
> x <- 1:10
> z <- NULL
> for(i in seq(along = x)) {
+ if (x[i] < 5) {
+   z <- c(z, x[i]- 1)
+   cat(i, z, "\n")
+ } else {
+   stop("Valores devem ser < 5")
+ }
+ }
1 0
2 0 1
3 0 1 2
4 0 1 2 3
Erro: Valores devem ser < 5
> z
```

Estatística Computacional I - 2020

52



## • Exemplo:

✓ Preenchimento parte inferior de matriz

```
# Construção de matriz triangular inferior esquerda (zeros na parte superior)
m <- 10; n <- 10
conta <- 0
matriz <- matrix(0, m, n)
for(linha in 1:m) {
  for(coluna in 1:n){
    if(linha == coluna){
      break
    } else {
      # calcula apenas fora da diagonal(i<>j)
      matriz[linha, coluna] = linha * coluna
      conta <- conta + 1
    }
  }
  print(linha * coluna)
}
print(conta) # qte. células preenchidas
matriz
```

Estatística Computacional I - 2020

53



✓ Comando para preenchimento parte inferior de matriz

```
> # comando para preenchimento de parte inferior de matriz
> valores.col <- c(2:10, seq(3*2, 10*2, by = 2), seq(4*3, 10*3, by = 3),
+ seq(5*4, 10*4, by = 4), seq(6*5, 10*5, by = 5),
+ seq(7*6, 10*6, by = 6), seq(8*7, 10*7, by = 7),
+ seq(9*8, 10*8, by = 8), 90)
> matriz <- matrix(0, m, n)
> matriz[lower.tri(matriz, diag = F)] <- valores.col
> matriz
```

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]	[,10]
[1,]	0	0	0	0	0	0	0	0	0	0
[2,]	2	0	0	0	0	0	0	0	0	0
[3,]	3	6	0	0	0	0	0	0	0	0
[4,]	4	8	12	0	0	0	0	0	0	0
[5,]	5	10	15	20	0	0	0	0	0	0
[6,]	6	12	18	24	30	0	0	0	0	0
[7,]	7	14	21	28	35	42	0	0	0	0
[8,]	8	16	24	32	40	48	56	0	0	0
[9,]	9	18	27	36	45	54	63	72	0	0
[10,]	10	20	30	40	50	60	70	80	90	0

Estatística Computacional I - 2020

54



## • Exemplo:

✓ Preenchimento de tabuada

```
# tabuada
tabuada <- matrix(nrow = 30, ncol = 30)
linhas <- 1:dim(tabuada)[1]
colunas <- 1:dim(tabuada)[2]
rownames(tabuada) <- linhas
colnames(tabuada) <- colunas
for(i in linhas){
  for(j in colunas){
    tabuada[i, j] = i * j
  }
}
tabuada
```

✓ Comando para preenchimento da matriz

```
> outer(1:30, 1:30) $ ou 1:30 %o% 1:30
[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
[1,] 1 2 3 4 5 6 7 8 9 10 11 12 13
[2,] 2 4 6 8 10 12 14 16 18 20 22 24 26
[3,] 3 6 9 12 15 18 21 24 27 30 33 36 39
[4,] 4 8 12 16 20 24 28 32 36 40 44 48 52
```

Estatística Computacional I - 2020

55

## Comando switch

- Permite seleccionar uma das opções com base no valor de uma expressão
- Sintaxe:
 

```
switch(expressão, opção_1, opção_2, ...)
```

✓ expressão deve ser uma *string* ou um número, mas não um vetor

Estatística Computacional I - 2020 56

## Exemplos:

### ✓ Expressão é string

```
> # Exemplos - expressao = string
> switch("cor", "cor" = "vermelho", "shape" = "quadrado",
+       "tamanho" = 5)
[1] "vermelho"
> switch("tamanho", "cor" = "vermelho", "shape" = "quadrado",
+       "tamanho" = 5)
[1] 5
```

### ✓ Expressão é número

```
> # Exemplos - expressao = numero
> switch(2, "vermelho", "verde", "azul")
[1] "verde"
> switch(1, "vermelho", "verde", "azul")
[1] "vermelho"
> switch(3, 6 + 4, mean(1:8), rnorm(4))
[1] 1.0123142 -0.7082136 -0.1727563 0.4819342
> x <- switch(4, "vermelho", "verde", "azul")
> x
NULL
```

Estatística Computacional I - 2020 57

## Comando switch - fluxograma

©tutorialgateway.org

✓ Se as opções falharem, será executado a instrução default.

Estatística Computacional I - 2020 58

## Exemplo:

### ✓ Cálculo de medidas de tendência central

```
# Cálculo de medidas de posição central
central <- function(y, medida){
  switch(medida, media = mean(y), mediana = median(y),
         geometrica = prod(y)^(1/length(y)),
         aparada = mean(x, trim = 0.1) , "Inválida")
}
> set.seed(666)
> y <- rexp(100)
> central(y, "media")
[1] 0.8571762
> central(y, "mediana")
[1] 0.524719
> central(y, "geometrica")
[1] 0.4254744
> central(y, "harmonica")
[1] "Inválida"
```

Estatística Computacional I - 2020 59



## Exemplo – Envasamento de leite

✓ Volume da embalagem ~ Normal com média 1.000 ml e desvio padrão 4 ml

✓ Amostras aleatórias de tamanho 5, coletadas a cada 30 min, num período de 8 horas

```
> # exemplo -leite
> set.seed(666)
> n <- 5; m <- 16
> amostras <- matrix(rnorm(n*m, 1000, 4), ncol = n, byrow = T)
> amostras <- round(amostras, 2)
> dim(amostras)
[1] 16 5
> rownames(amostras) <- paste(seq(0.5, 8, by = 0.5), "h")
> colnames(amostras) <- paste("Emb.#", 1:5)
> head(amostras)
Emb.# 1 Emb.# 2 Emb.# 3 Emb.# 4 Emb.# 5
0.5 h 1003.01 1008.06 998.58 1008.11 991.13
1 h 1003.03 994.78 996.79 992.83 999.83
1.5 h 1008.60 992.92 1003.46 993.12 1000.54
```

Estatística Computacional I - 2020

60



## Qual a média de cada amostra?

– Média das 5 embalagens a cada ½ h

```
> # média por linha - manual (mais difícil)
> mean(amostras[1, ]); mean(amostras[2, ])
[1] 1001.778
[1] 997.452
> mean(amostras[3, ]); mean(amostras[4, ])
[1] 999.728
[1] 1001.064
> mean(amostras[5, ]); mean(amostras[6, ])
[1] 999.29
[1] 994.576
> mean(amostras[7, ]); mean(amostras[8, ])
[1] 1000.918
[1] 1001.158
> mean(amostras[9, ]); mean(amostras[10, ])
[1] 1001.69
[1] 999.75
> mean(amostras[11, ]); mean(amostras[12, ])
[1] 999.654
[1] 1000.14
```

✓ Funciona, mas é a maneira mais difícil

Estatística Computacional I - 2020

61



## Vetor com as com as 16 médias amostrais

– Geração com loop

```
> # média por linha - for
> tamanho <- nrow(amostras)
> medias.row <- numeric(tamanho)
> for (ii in seq_along(medias.row)) {
+   medias.row[ii] = mean(amostras[ii, ])
+ }
> medias.row
[1] 1001.778 997.452 999.728 1001.064 999.290 994.576 1000.918 1001.158
[9] 1001.690 999.750 999.654 1000.140 999.518 1001.412 999.364 996.022
```

– Criação de função para cálculo das médias amostrais

```
# media por linha - função
medias.row.func <- function(matriz) {
  tamanho <- nrow(matriz)
  medias.row <- numeric(tamanho)
  for (ii in seq_along(medias.row)) {
    medias.row[ii] = mean(matriz[ii, ])
  }
  return(medias.row)
}
> medias.row.func(amostras)
[1] 1001.778 997.452 999.728 1001.064 999.290 994.576 1000.918 1001.158
[9] 1001.690 999.750 999.654 1000.140 999.518 1001.412 999.364 996.022
```

Estatística Computacional I - 2020

62



## Geração de vetor com as médias por linha ou por coluna



– Função

```
# media por linha e coluna - função
medias.func <- function(matriz, rc = 2) {
  # rc: 1 = medias por linha, 2 = medias por coluna

  if (rc == 1) {
    tamanho <- nrow(matriz)
    medias <- numeric(tamanho)
    for (ii in seq_along(medias)) {
      medias[ii] = mean(matriz[ii, ])
    }
  } else {
    tamanho <- ncol(matriz)
    medias <- numeric(tamanho)
    for (ii in seq_along(medias)) {
      medias[ii] = mean(matriz[, ii])
    }
  }
  return(medias)
}
```

Estatística Computacional I - 2020

63



### ✓ Geração de vetor com as medias por linha ou por coluna

– Médias por linhas



```
> medias.func(amostras, 1)
[1] 1001.778 997.452 999.728 1001.064 999.290 994.576 1000.918 1001.158
[9] 1001.690 999.750 999.654 1000.140 999.518 1001.412 999.364 996.022
```

– Médias por colunas

```
> medias.func(amostras, 2)
[1] 999.9094 998.9438 998.6206 1000.4650 1000.0344
```

Estatística Computacional I - 2020 64

## 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 285