

Uma Breve Introdução ao R

Eduardo Lemos, Ricardo Masuda, Samuel Vianna, Vitor Landi

Contents

1	Princípios Básicos	5
1.1	Primeiros Passos	5
1.2	Operações Básicas	7
1.3	Estruturas Básicas	8

Chapter 1

Princípios Básicos

1.1 Primeiros Passos

R é uma linguagem orientada à objetos que são armazenados na memória ativa do computador. Uma variável é um objeto que irá representar um valor ou expressão atribuído a ela. Só é possível armazenar um dado ou expressão pra cada variável, quando for atribuído mais de uma informação, o dado que estava antes armazenado será substituído.

1.1.1 Comandos Básicos

Primeiramente, para a melhor utilização do R, é necessário saber alguns comandos básicos. São eles:

- `control + L`: Limpar o console
- `control + R` ou `control + enter`: Compilar o código escrito
- `rm(list = ls())`: limpar memória
- `#`: fazer comentários no código

1.1.2 Atribuição de Valores

Pode-se atribuir um valor à um objeto dentro do ambiente do R de duas formas diferentes: `<-` e `=`.

Exemplos:

```
# atribuindo o valor 10 para a variavel x
x <- 10
x
```

```
## [1] 10
```

```
# atribuindo o valor 5 para a variável y
y = 5
y
```

```
## [1] 5
```

Observação: Vale ressaltar que o sinal de igual é usado para a atribuição de valores, e não denotar igualdade, para isso é usado dois sinais (==).

1.1.3 Tipos de Variáveis

Toda variável declarada possui uma classe específica, de acordo com o seu conteúdo.

Para verificar a classe de uma determinada variável, utiliza-se a função `class`.

Exemplos:

```
# numérica
x <- 1.5
class(x)
```

```
## [1] "numeric"
```

```
# caractere: palavras, textos, etc
y <- "estatística"
class(y)
```

```
## [1] "character"
# lógico: TRUE, FALSE
z <- 4 < 5
class(z)
```

```
## [1] "logical"
```

1.1.4 Utilizando Ajuda (help)

Para buscar ajuda no R, pode-se usar a função `help()` ou o operador `?`.

Exemplos:

```
# Buscando ajuda sobre a função log
help(log)

?help
```

1.2 Operações Básicas

No ambiente R, existem uma série de operações básicas que são muito usuais e de grande importância. Tais como:

1.2.1 Operações simples

- \wedge : Potencialização
- $/$: Divisão
- $*$: Multiplicação
- $+$: Adição
- $-$: Subtração

1.2.2 Operações lógicas

- $<$: Menor
- $<=$: Menor ou igual
- $>$: Maior
- $>=$: Maior ou igual
- $==$: Igual
- $!=$: Diferente
- $\&$: AND
- $!$: NOT
- $|$: OR
- `FALSE` ou 0: Valor booleano falso (0)
- `TRUE` ou 1: Valor booleano verdadeiro (1)

1.2.3 Operações matemáticas

- `abs(x)`: Valor absoluto de x
- `log(x,b)`: Logaritmo de x com base b
- `log(x)`: Logaritmo natural de x
- `log10(x)`: Logaritmo de x na base 10
- `exp(x)`: Exponencial elevado a x
- `sin(x)`: Seno de x
- `cos(x)`: Cosseno de x

- `tan(x)`: Tangente de x
- `round(x, digits = n)`: Arredonda x com n decimais
- `ceiling(x)`: Arredonda x para o maior valor
- `floor(x)`: Arredonda x para o menor valor
- `sqrt(x)`: Raiz quadrada de x

1.3 Estruturas Básicas

1.3.1 Vetor

Um vetor é um conjunto de valores atribuídos à uma variável. Para criar um vetor, utiliza-se o comando `c()`.

Exemplos de vetores:

```
vetor1 <- c(1, 1, 2, 3, 5, 8)
idades <- c(17, 20, 22, 18, 30)
alunos <- c("Ricardo", "Samuel", "Vitor", "Ellen", "Mariana")
vetor2 <- c(0, vetor1, 0)
```

Existem funções que permitem criar e manipular vetores com características com maior facilidade, a seguir, estão algumas delas:

Sequências

Para criar um vetor baseado em uma sequência, pode-se usar a função `seq()`, que cria um vetor do valor A até o valor Z .

Exemplos:

```
# Criar um vetor de 1 a 10
vetor1 <- seq(from = 1, to = 10)
vetor1

## [1] 1 2 3 4 5 6 7 8 9 10

#outra forma de criar o vetor de 1 a 10
vetor1.1 <- 1:10
```

Perceba que, por padrão, o intervalo entre os números gerados é 1. Porém, também pode-se alterar a distância entre os elementos (ou a “distância de passos”), com o argumento `by = N`, e a quantidade de elementos criados, com o argumento `length.out = N`.

Exemplos:

```
# Criar vetor de 1 a 10, com tamanho do passo = 2
vetor2 <- seq(from = 1, to = 10, by = 2)
vetor2
```



```
## [1] 1 3 5 7 9
# Criar vetor de 1 a 10, com 4 elementos
vetor3 <- seq(from = 1, to = 10, length.out = 4)
vetor3
```

```
## [1] 1 4 7 10
```

Operações em vetores

É possível aplicar uma série de operações em vetores, a seguir, algumas das operações mais utilizadas:

- `length(x)`: número de elementos do vetor `x`
- `sum(x)`: soma dos elementos do vetor `x`
- `prod(x)`: produto dos elementos do vetor `x`
- `max(x)`: seleciona o maior elemento do vetor `x`
- `min(x)`: seleciona o menor elemento do vetor `x`
- `range(x)`: retorna o menor e o maior elemento do vetor `x`

Criando vetores com a função `paste`

É possível também manipular vetores “colando” partes com a função `paste`.

Pode-se usá-lo para adicionar tanto um prefixo quanto um sufixo, usando as seguintes sintaxes:

- Prefixo: `paste("prefixo", vetor, sep = "separador")`
- Sufixo: `paste(vetor, "sufixo", sep = "separador")`

Exemplos:

```
x <- 1:10
# adicionando o prefixo "número", separando com "_"
paste("número", x, sep = "_")

## [1] "número_1" "número_2" "número_3" "número_4" "número_5" "número_6"
## [7] "número_7" "número_8" "número_9" "número_10"
# adicionando sufixo e atribuindo o resultado à variável "y"
y <- c(paste(11:20, "número", sep = "%"))
y

## [1] "11%número" "12%número" "13%número" "14%número" "15%número" "16%número"
## [7] "17%número" "18%número" "19%número" "20%número"
```

Caso deseja-se adicionar um elemento “grudado” ao valor, pode-se tanto usar o argumento `sep=""` dentro da função `paste`, como a função `paste0`.

Exemplo:

```
# usando sep = ""
z <- c(paste("numero", 21:30, sep = ""))
z
```

```
## [1] "numero21" "numero22" "numero23" "numero24" "numero25" "numero26"
## [7] "numero27" "numero28" "numero29" "numero30"
```

```
#usando paste0
w <- c(paste0("numero", 21:30))
w
```

```
## [1] "numero21" "numero22" "numero23" "numero24" "numero25" "numero26"
## [7] "numero27" "numero28" "numero29" "numero30"
```

Repetições

É possível repetir um elemento ou um vetor com a função `rep()`. A seguir, alguns dos argumentos mais utilizados dentro da função:

- **times**: define o número de vezes que o número ou vetor inteiro será repetido
- **each**: define o número de vezes que cada elemento em um vetor será repetido
- **length.out**: define o tamanho do vetor de saída

Exemplos:

```
# repetindo um número 10 vezes
r1 <- rep(5, times = 10) # ou somente rep(5,10)
r1
```

```
## [1] 5 5 5 5 5 5 5 5 5 5
```

```
x <- c("a", "b", "c")
```

```
# repetindo o vetor inteiro 5 vezes
rep(x, times = 5)
```

```
## [1] "a" "b" "c" "a" "b" "c" "a" "b" "c" "a" "b" "c" "a" "b" "c"
# repetindo cada elemento do vetor 5 vezes
rep(x, each = 5)
```

```
## [1] "a" "a" "a" "a" "a" "b" "b" "b" "b" "b" "c" "c" "c" "c" "c"
# criando um vetor de tamanho 7
rep(x, length.out = 7)
```

```
## [1] "a" "b" "c" "a" "b" "c" "a"
```

Selecionando um elemento no vetor

Caso deseja-se saber qual o elemento se encontra em uma determinada posição de um vetor, denotada por i , pode-se localizá-lo utilizando a sintaxe `vetor[i]`

Vale ressaltar que a contagem é iniciada a partir do valor 1, diferente de certas linguagens de programação em que a contagem começa na posição 0.

Exemplo:

```
# localizando o décimo terceiro número par entre 10 e 50
valores <- seq(10, 50, by = 2)
valores[13]

## [1] 34
```

1.3.2 Matriz

Uma matriz é uma generalização de um vetor, tendo duas dimensões (linhas e colunas). Podemos pensar em um vetor como uma matriz com uma de suas dimensões igual a 1. A sintaxe é dada abaixo, em que “L” é o número de linhas, “C” é o número de colunas e se “Q” = 1 ativa disposição por linhas, se “Q” = 0 mantém disposição por colunas (ou T ou F).

```
x <- matrix(data = dados, nrow = L, ncol = C, byrow = Q)
```

Exemplos:

```
# Criando uma matriz de 2 linhas, 5 colunas e disposição por linhas
m1 <- matrix(data = c(1:10), nrow = 2, ncol = 5, byrow = 1)
m1
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
```

```
# Criando uma matriz de 2 linhas, 5 colunas e disposição por colunas:
mc <- matrix(data = c(1:10), nrow = 2, ncol = 5, byrow = 0)
mc
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
```

Selecionando elemento da matriz

Para selecionar um elemento de uma matriz utilizamos a indexação por colchetes na variável que representa a matriz com os índices separados por vírgula.

Exemplos:

```
# Selecionando a linha 2 e coluna 4 da matriz m1
m1[2,4]
```

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

```
# Selecionando a linha 2 da matriz m1
m1[2,]
```

```
## [1] 6 7 8 9 10
```

```
# Seleccionando as colunas 2,3 e 4 da matriz ml
ml[:,2:4]
```

```
##      [,1] [,2] [,3]
## [1,]    2    3    4
## [2,]    7    8    9
```

```
# Outra forma de ler a matriz ml
ml[:,]
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
```

Operações de matrizes

- $A*B$: Produto elemento a elemento de A e B - $A\% * B$: Produto matricial de A por B - $\text{apern}(A)$: Matriz transposta de A - $\text{t}(A)$: Matriz transposta de A - $\text{solve}(A)$: Matriz inversa de A - $\text{solve}(A,B)$: Resolve o sistema linear $Ax = B$ - $\text{det}(A)$: Retorna o determinante de A - $\text{diag}(v)$: Retorna uma matriz diagonal onde o vetor v é a diagonal - $\text{diag}(A)$: Retorna um vetor que é a diagonal de A - $\text{diag}(n)$: Sendo n um inteiro, retorna uma matriz identidade de ordem n - $\text{eigen}(A)$: Retorna os autovalores e autovetores de A