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

Programando em R

Família `Apply`

Família `apply`

- Podem ser uma alternativa para loops

Função	Descrição
`apply`	Aplica funções nas margens de arrays
`by`	Aplica uma função em dataframe estratificado por fatores
`eapply`	Aplica uma função em valores de um environment
`lapply`	Aplica uma função em uma lista, vetor ou data frame
`mapply`	Aplica uma função a múltiplos argumentos de lista ou vetor
`rapply`	Aplica uma função recursivamente a uma lista
`sapply`	Aplica uma função em uma lista, vetor ou data frame
`sweep`	
`tapply`	Aplica uma função sobre uma matriz irregular

Comando `apply`

- Aplica uma função a cada linha ou coluna de uma matriz
- Sintaxe:

```
apply(X, MARGIN, FUN, ...)
```

  - √ `X`: um array
  - √ `MARGIN`: (1 = linha, 2 = coluna, c(1,2) = linha e coluna)
  - √ `FUN`: função a ser aplicada
  - √ `...` : argumentos opcionais da função

- 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
```


- Exemplo – Caixas de leite
  - √ Função `apply` (maneira simples!):

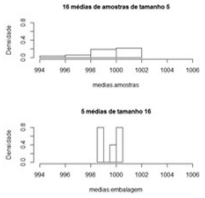
```
> # média por linha - apply (maneira simples)
> medias.amostras <- apply(amostras, 1, mean)
> medias.amostras
[1] 1001.778  997.452  999.728 1001.064  999.290  994.576 1000.918 1001.158
[9] 1001.690  999.750  999.654 1000.140  999.518 1001.412  999.364  996.022
> is.vector(medias.amostras)
[1] TRUE
> all.equal(medias.amostras, medias.row)
[1] TRUE
> # média por coluna - apply
> medias.embalagem <- apply(amostras, 2, mean)
> medias.embalagem
[1]  999.9094  998.9438  998.6206 1000.4650 1000.0344
```

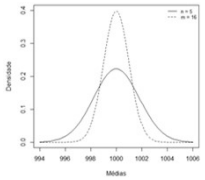
- Qual dos vetores de média apresenta menor variabilidade?

```
> # variabilidade das medias amostrais]
> length(medias.amostras)
[1] 16
> sd(medias.amostras)
[1] 2.030615
> length(medias.embalagem)
[1] 5
> sd(medias.embalagem)
[1] 0.7781893
```

$$\frac{\sigma}{\sqrt{n}} = \frac{4}{\sqrt{5}} = 1.79$$

$$\frac{\sigma}{\sqrt{m}} = \frac{4}{\sqrt{16}} = 1$$





√ Códigos dos gráficos

– Histogramas em um mesmo painel

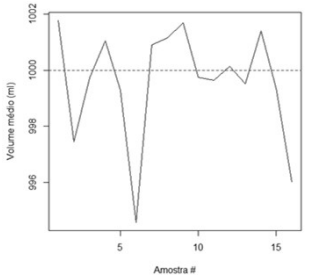
```
> op <- par(mfrow = c(2, 1))
> hist(medias.amostras, freq = F, ylab = "Densidade", ylim = c(0, 0.8),
+ xlim = c(994, 1006), main = "16 médias de amostras de tamanho 5",
cex.main = 1)
> hist(medias.embalagem, freq = F, ylab = "Densidade", ylim = c(0, 0.8),
+ xlim = c(994, 1006), main = "5 médias de tamanho 16", cex.main = 1)
> par(op)
> op
$mfrow
[1] 1 1
```

– Densidades sobrepostas no mesmo gráfico

```
> curve(dnorm(x, mean = 1000, sd = 4/sqrt(5)), 994, 1006, ylim = c(0, 0.4),
+  ylab = "Densidade", xlab = "Médias")
> curve(dnorm(x, mean = 1000, sd = 4/sqrt(16)), 994, 1006, add = TRUE, lty
= 2,
+ col = 4)
> legend("topright", c("n = 5", "m = 16"), lty = c(1, 2), col = c(1, 4),
+  cex = 0.8, bty = "n")
> palette()
[1] "black"   "red"     "green3"  "blue"    "cyan"    "magenta" "yellow"
[8] "gray"
```


√ Gráfico sequencial da média de cada amostra

```
> # grafico sequencial das médias amostrais
> plot(medias.amostras, type = "l", ylab = "Volume médio (ml)",
+ xlab = "Amostra #")
> abline(1000, 0, lty =2, col = "red")
```



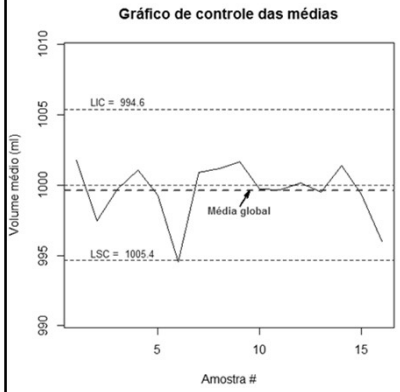
√ Ideal centralizar o gráfico em torno da média da produção (valor alvo)

– 1.000 ml

√ Gráfico de controle das médias

```
> # grafico de controle das médias - a cada 1/2 h
> lm <- media
> lsc <- media + 3*dp/sqrt(n)
> lic <- media - 3*dp/sqrt(n)
> LIC <- paste("LIC = ", round(lic, 1))
> LSC <- paste("LSC = ", round(lsc, 1))
> plot(medias.amostras, type = "l", ylim = c(lic - 4, lsc + 4),
+ ylab = "Volume médio (ml)", xlab = "Amostra #")
> abline(h = c(lic, 1000, lsc), lty = 2, col = c("blue", "red", "blue"))
> text(x = c(1, 1), y = c(lic, lsc) + 0.5, c(LSC, LIC), cex = 0.8, pos = 4,
offset = 1,
+ col = "blue")
> title(main = "Gráfico de controle das médias")
> abline(h = mean(medias.amostras), lty = 2, lwd = 2, col = "red")
> text(9, 1000 - 1, "Média global", pos = 1, cex = 0.8, font = 2, col =
"red")
> arrows(9, 1000 - 1.5, 9.5, mean(medias.amostras), lwd = 2, length = 0.1,
+  angle = 15)
```

√ Gráfico de controle das médias

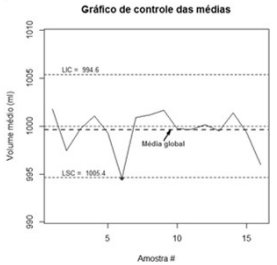


√ Alguma da médias saiu da região de controle do gráfico?

√ Qual?

√ Identificação ponto fora da curva

```
> # identificação pontos fora da região de controle
> fora <- medias.amostras[medias.amostras<lic | medias.amostras > lsc]
> fora; lic
[1] 994.576
[1] 994.6334
> which(medias.amostras == fora)
[1] 6
> points(6, medias.amostras[6], pch = 16, col = "red")
```



√ Tente o comando `identify(1)`.

- Para as médias de linhas e colunas há comandos específicos!

√ Comandos `rowMeans` e `colMeans`

```
> # media por linha e coluna - comandos rowMeans e colMeans
> rowMeans(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
> colMeans(amostras)
[1]  999.9094  998.9438  998.6206 1000.4650 1000.0344
```

√ Outros comandos por linhas e colunas

– `rowSums` e `colSums`.

- Suponha que não tenha sido anotado o volume da 3ª. embalagem da amostra da 6ª. meia hora

```
> # comando apply - comandos opcionais
> amostras[6, 3] <- NA
> apply(amostras, 1, mean)
[1] 1001.778  997.452  999.728 1001.064  999.290       NA 1000.918 1001.158
[9] 1001.690  999.750  999.654 1000.140  999.518 1001.412  999.364  996.022
> apply(amostras, 1, mean, na.rm = TRUE)
[1] 1001.778  997.452  999.728 1001.064  999.290  994.985 1000.918 1001.158
[9] 1001.690  999.750  999.654 1000.140  999.518 1001.412  999.364  996.022
```

√ Argumentos opcionais da função são dispostos no comando `apply`

- Deseja-se calcular o coeficiente de variação por linha e por coluna
  - √ Não há comando no R para esse cálculo

```
> # cálculo de função construída pelo usuário
> apply(amostras, 1, function(x) sd(x, na.rm = T)/mean(x, na.rm = T))
 [1] 0.0071406716 0.0040634290 0.0067715184 0.0024154519 0.0037076493
 [6] 0.0007792574 0.0035788397 0.0050205090 0.0038794314 0.0027465927
[11] 0.0031722062 0.0037257097 0.0055418110 0.0031216872 0.0028533020
[16] 0.0045046186
> apply(amostras, 2, function(x) sd(x, na.rm = T)/mean(x, na.rm = T))
[1] 0.003696620 0.004012343 0.004656646 0.005019586 0.004160150
```


- Script com funções e objeto
  - √ `funcoes.R`

```
# Simulação de volumes de leite
set.seed(666)
n <- 5; m <- 16; media <- 1000; dp = 4
amostras <- matrix(rnorm(n*m, mean = media, sd = dp), ncol = n, byrow = T)
amostras <- round(amostras, 2)


central <- function(y, medida){
        #   Cálculo de medidas de posição central
        switch(medida, media = mean(y), mediana = median(y),
                    geometrica = prod(y)^(1/length(y)),
                    aparada = mean(x, trim = .1),
                    "Inválida")
}
```


- Aplicação de função do usuário
  - √ Função `central`, para cálculo de médias geométricas por linha

```
> # aplicação de função construída anteriormente
> source("funcoes.R")
> apply(amostras, 1, FUN = central, medida = "geometrica")
 [1] 1001.7575  997.4454  999.7097 1001.0617  999.2845  994.5755 1000.9129
 [8] 1001.1479 1001.6840  999.7470  999.6500 1000.1345  999.5057 1001.4081
[15]  999.3607  996.0139
```


- `apply`:
  - √ Aplica função a todos os elementos de objeto

Função	Objetos[a]	Elementos[b]	Tipo de resultado
`apply`	Matriz	Linhas ou colunas	Vetor, matriz, array ou lista
	Array	Linhas, colunas ou qualquer dimensão	Vetor, matriz, array ou lista
	Data frame	Linhas ou colunas	Vetor, matriz, array ou lista

  - √ [a] Objeto com os quais a função trabalha
  - √ [b] Elementos do objeto vistos pela função

Comando `lapply`

- Aplica uma função a cada linha ou coluna de uma matriz
  - √ Saída é uma lista
- Sintaxe:

```
lapply(X, FUN, ...)
```

  - √ `X`: vetor, lista ou data frame
  - √ `FUN`: função a ser aplicada
  - √ `...` : argumentos opcionais da função

- `lapply`:
  - √ Aplica função a todos os elementos de objeto

Função	Objetos[a]	Elementos[b]	Tipo de resultado
`lapply`	Vetor	Elementos	Lista
	Data frame	Variáveis	Lista
	Lista	Elementos	Lista

  - √ [a] Objeto com os quais a função trabalha
  - √ [b] Elementos do objeto vistos pela função

- Diferença com o comando `apply`:
  - √ Pode ser usada com objetos do tipo data frame, lista ou vetor
  - √ A saída do comando é uma lista
    - Possui o mesmo número de elementos do objeto

- Exemplo – lista de objetos
  - √ Elementos são objetos diferentes

```
> #cria lista com dois elementos
> lista <- list(a = 1:10, b = 11:20, c = matrix(1:6, ncol = 2))
> # média dos valores em cada elemento da lista
> lapply(lista, mean)
$a
[1] 5.5

$b
[1] 15.5

$c
[1] 3.5
> class(lapply(lista, mean))
[1] "list"
> # soma dos valores em cada elemento
> lapply(lista, sum)
$a
[1] 55

$b
[1] 155

$c
[1] 21
```

- Exemplo – data frame
  - √ Elementos são as variáveis

```
>   # exemplo - data frame
> lapply(iris[1:4], mean)
$Sepal.Length
[1] 5.843333

$Sepal.Width
[1] 3.057333

$Petal.Length
[1] 3.758

$Petal.Width
[1] 1.199333
> class(lapply(iris[1:4], mean))
[1] "list"
```


- Exemplo – vetor
  - √ Função é aplicada em cada elemento do vetor

```
>   # exemplo - vetor
> lapply(1:3, function(x) x^2)
[[1]]
[1] 1

[[2]]
[1] 4

[[3]]
[1] 9
> # transforma lista em vetor
> unlist(lapply(1:3, function(x) x^2))
[1] 1 4 9
```

- Importante:
  - √ Se você quer uma lista como saída, use `lapply`; se quer um vetor, use `sapply`

- Exemplo – lista de matrizes

```
> #criação de uma lista de matrizes
> A <- matrix(1:9, ncol = 3)
> B <- matrix(4:15, ncol = 3)
> C <- matrix(8:10, nrow = 3, ncol = 2)
> lista.mat <-list(A, B, C)
> # extrair a 2a. coluna de cada matriz
> lapply(lista.mat, "[", , 2)
[[1]]
[1] 4 5 6

[[2]]
[1]  8  9 10 11

[[3]]
[1]  8  9 10
> unlist(lapply(lista.mat, "[", , 2))
[1] [1]  4  5  6  8  9 10 11  8  9 10
```


Comando `sapply`

- Aplica uma função a cada linha ou coluna de uma matriz
  - √ Saída é um vetor
- Sintaxe:

```
sapply(X, FUN, ..., simplify = T)
```

  - √ `X`: vetor, lista ou data frame
  - √ `FUN`: função a ser aplicada
  - √ `...` : argumentos opcionais da função
  - √ `simplify`: T = saída é vetor, F = saída é lista

- Diferença com o comando `lapply`:
  - √ Tenta simplificar a saída para a estrutura de dados mais elementar possível

- `sapply`:
  - √ Aplica função a todos os elementos de objeto

Função	Objetos[a]	Elementos[b]	Tipo de resultado
`sapply`	Vetor	Elementos	Vetor, matriz ou lista
	Data frame	Variáveis	Vetor, matriz ou lista
	Lista	Elementos	Vetor, matriz ou lista

  - √ [a] Objeto com os quais a função trabalha
  - √ [b] Elementos do objeto vistos pela função

- Exemplo
  - √ Saídas como vetores

```
>   # exemplo - lista
> # média dos valores em cada elemento da lista
> sapply(lista, mean)
   a    b    c
 5.5 15.5  3.5
> class(sapply(lista, mean))
[1] "numeric"
> # soma dos valores em cada elemento
> sapply(lista, sum)
  a   b   c
 55 155  21
>   # exemplo - data frame
> sapply(iris[1:4], mean)
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
    5.843333     3.057333     3.758000     1.199333
>   # exemplo - vetor
> sapply(1:3, function(x) x^2)
[1] 1 4 9
```


- Exemplos
  - √ Uso do argumento `simplify`

```
> # extrair a 1o. elemento da 2a. linha de cada matriz
> sapply(lista.mat, "[", 1, 2)
[1] 4 8 8
> # resultado de sapply será uma lista
> sapply(lista.mat, "[", 1, 2, simplify = F)
[[1]]
[1] 4

[[2]]
[1] 8

[[3]]
[1] 8
```

- Exemplo:
  - √ Rendimento total de cevada, em bushels por acre, para 10 variedades em 6 locais, em 2 anos

```
>   # barley data frame
> library (lattice)
> dim (barley)
[1] 120   4
> head(barley)
     yield   variety year           site
1 27.00000 Manchuria 1931 University Farm
2 48.86667 Manchuria 1931          Waseca
3 27.43334 Manchuria 1931          Morris
4 39.93333 Manchuria 1931       Crookston
5 32.96667 Manchuria 1931    Grand Rapids
6 28.96667 Manchuria 1931          Duluth
> unique(barley$site)
[1] University Farm Waseca          Morris          Crookston
[5] Grand Rapids    Duluth
> unique(barley$yield)
> barley$yield[duplicated(barley$yield)]
[1] 29.66667 48.56666 32.96667 35.13333 27.43334 20.63333
```


- √ Quantidade de valores únicos por variável

```
>   # valores únicos por variável
> # saída: lista
> lapply (barley, function(x) length(unique(x)))
$yield
[1] 114
$variety
[1] 10
$year
[1] 2
$site
[1] 6
> is.list(lapply (barley, function(x) length(unique(x))))
[1] TRUE
> # saída: vetor
> sapply (barley, function(x) length(unique(x)))
 yield variety    year    site
   114      10       2       6
> # funciona em data frames mas não em listas
> apply (barley, 2, function(x) length(unique(x)))
  yield variety    year    site
   114      10       2       6
> is.vector(apply (barley, 2, function(x) length(unique(x))))
[1] TRUE
```


X, INDEX

Comando `tapply`

- Aplica uma função a estratos de um vetor
  - √ Saída é um vetor
- Sintaxe:

```
tapply(X,    INDEX,    FUN,    ...,
simplify = T)
```

  - √ `X`: vetor, lista ou data frame
  - √ `INDEX`: lista de um ou mais fatores, com mesma dimensão que X.
  - √ `FUN`: função a ser aplicada
  - √ `...` : argumentos opcionais da função
  - √ `simplify`: F = saída é lista

- Exemplo - `mtcars`

```
> str(mtcars$cyl)
num [1:32] 6 6 4 6 8 6 8 4 4 6 ...
> is.factor(mtcars$cyl)
[1] FALSE
> levels(as.factor(mtcars$cyl))
[1] "4" "6" "8"
> tapply(mtcars$mpg,mtcars$cyl,mean)
       4        6        8
26.66364 19.74286 15.10000
> # media de mpg por nivel das variáveis cyl e gear
> with(mtcars, tapply(mpg,list(cyl, gear), mean, na.rm = T))
      3      4    5
4 21.50 26.925 28.2
6 19.75 19.750 19.7
8 15.05     NA 15.4
```

• Exemplo – `barley`:

```
> library(lattice)
> dim(barley)
[1] 120   4
> tapply (barley$yield, barley$site, mean)
   Grand Rapids          Duluth University Farm          Morris       Crookston
       24.93167        27.99667        32.66667        35.40000        37.42000
         Waseca
       48.10833
> is.array(tapply (barley$yield, barley$site, mean))
[1] TRUE
> tapply (barley$yield, list (barley$year, barley$site), mean)
     Grand Rapids   Duluth University Farm   Morris Crookston   Waseca
1932     20.81000 25.70000        29.50667 41.51333     31.18 41.87000
1931     29.05334 30.29333        35.82667 29.28667     43.66 54.34667
> is.matrix(tapply (barley$yield, list (barley$year, barley$site), mean))
[1] TRUE
```


√ `barley` – função definida pelo usuário

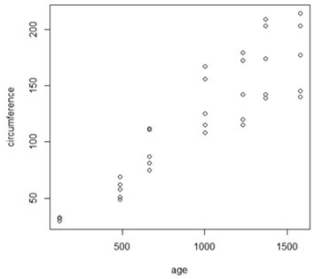
```
> objeto <- tapply(barley$yield, barley$site, function(x) c(media = mean(x),
+  dp = sd(x)))
> objeto
$`Grand Rapids`
    media        dp
24.931667  6.598109

$Duluth
    media        dp
27.996667  4.037034
...
> is.vector(objeto[1])
[1] TRUE
> tabela <- t(sapply(objeto, rbind))
> colnames(tabela) <- c("Média", "Desio padrão")
> tabela
                  Média Desio padrão
Grand Rapids    24.93167     6.598109
Duluth          27.99667     4.037034
University Farm 32.66667     6.159344
Morris          35.40000     8.186352
Crookston       37.42000     8.089373
Waseca          48.10833     9.464096
```


• Exemplo - `Orange`

```
> str(Orange)
Classes 'nfnGroupedData', 'nfGroupedData', 'groupedData' and 'data.frame':
35 obs. of  3 variables:
 $ Tree         : Ord.factor w/ 5 levels "3"<"1"<"5"<"2"<..: 2 2 2 2 ...
 $ age          : num  118 484 664 1004 1231 ...
 $ circumference: num  30 58 87 115 120 142 145 33 69 111 ...

> head(Orange)
   Tree  age circumference
1     1  118            30
2     1  484            58
3     1  664            87
4     1 1004           115
5     1 1231           120
6     1 1372           142
7     1 1582           145
8     2  118            33
> with(Orange, tapply(circumference, Tree, mean))
        3         1         5         2         4
 94.00000  99.57143 111.14286 135.28571 139.28571
```


√ Diagrama de dispersão entre circunferência e idade

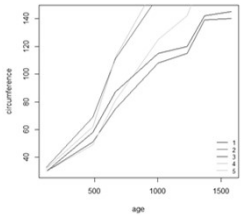
```
> # relação entre circunfeência e idade
> with(Orange, plot(circumference ~ age ))
```



- √ Há associação entre a circunferência e a idade das árvores?
- √ Como foi a curva de crescimento de cada uma das 5 árvores?

√ Gráfico sequencial de cada árvore

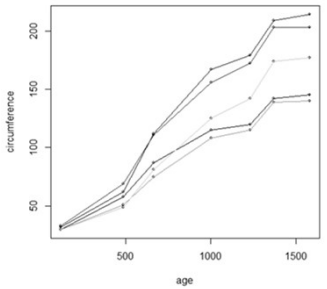
```
> # gráfico sequencial do crescimento de cada árvore
> with(Orange[Orange$Tree == 1, ], plot(circumference ~ age, type = "l"))
> with(Orange[Orange$Tree == 2, ], lines(circumference ~ age, type = "l",
+ col = "red"))
> with(Orange[Orange$Tree == 3, ], lines(circumference ~ age, type = "l",
+ col = "blue"))
> with(Orange[Orange$Tree == 4, ], lines(circumference ~ age, type = "l",
+ col = "green"))
> with(Orange[Orange$Tree == 5, ], lines(circumference ~ age, type = "l",
+ col = "gray"))
> legend("bottomright", c("1", "2", "3", "4", "5"), lty = 1, cex = 0.8,
+  col = c("black", "red", "blue", "green", "gray"), bty = "n")
```



√ E como desenhar uma curva média de crescimento das árvores?

√ Gráfico sequencial de cada árvore – alternativa:

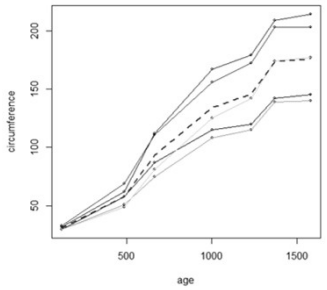
```
> # gráfico sequencial do crescimento de cada árvore - alternativa
> with(Orange, plot(circumference ~ age, cex = 0.5))
> sequencial <- function(i){
+ with(Orange[Orange$Tree == i, ], lines(circumference ~ age, type = "l",
+ col = i))
+ }
> lapply(1:5, sequencial)
```



√ Alguma ideia para calcular as médias?

√ Gráfico sequencial de cada árvore – alternativa:

```
> # circunferência média por idade
> with(Orange, tapply(circumference, age, mean) )
  118   484   664  1004  1231  1372  1582
 31.0  57.8  93.2 134.2 145.6 173.4 175.8
> circunferencia.media <- with(Orange, tapply(circumference, age, mean))
> idade <- unique(Orange$age)
> lines(circunferencia.media ~ idade, col = "red", lty = 2, lwd = 2)
```



√ Não é a única maneira

X, INDEX

Comando `by`

- Aplica uma função a estratos de objeto
  - √ Saída é um objeto da classe by
- Sintaxe:

```
by(data, INDEX, FUN, ...)
```

  - √ `data`: data frame ou matriz
  - √ `INDEX`: fator ou lista de fatores.
  - √ `FUN`: função a ser aplicada
  - √ `...` : argumentos opcionais da função

- Comentário:
  - √ Aplica uma função a um data frame estratificado por um fator

- Exemplo - `iris`

```
> # média por variável para espécie setosa
> y1 <- with(iris, subset(iris, Species == "setosa", select = -Species))
> apply(y1, 2, mean)
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
       5.006        3.428        1.462        0.246
> # média por variável para todas as espécies
> by(iris[,1:4],iris$Species,colMeans)
iris$Species: setosa
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
       5.006        3.428        1.462        0.246
------------------------------------------------------------
iris$Species: versicolor
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
       5.936        2.770        4.260        1.326
------------------------------------------------------------
iris$Species: virginica
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
       6.588        2.974        5.552        2.026

> by(iris[,1:4],iris$Species,colMeans)$setosa
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
       5.006        3.428        1.462        0.246
> is.vector(by(iris[,1:4],iris$Species,colMeans)$setosa)
 [1] TRUE
```


- Médias das variáveis por Species
  - √ Construção de matriz com as médias

```
> # média por variável para todas as espécies
> do.call(rbind, as.list(by(iris[,1:4], iris$Species, colMeans)))
           Sepal.Length Sepal.Width Petal.Length Petal.Width
setosa            5.006       3.428        1.462       0.246
versicolor        5.936       2.770        4.260       1.326
virginica         6.588       2.974        5.552       2.026
> is.matrix(do.call(rbind, as.list(by(iris[,1:4], iris$Species, colMeans))))
 [1] TRUE
```


Comando `do.call`

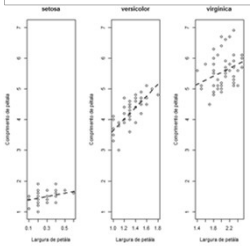
- Executa uma função cujos argumentos estão em uma lista
  - √ Diferente de `lapply` (ou `sapply`)
- `lapply`:
  - √ Aplica uma função a todos os elementos de uma lista

- Exemplo:

```
> # do.call
> do.call(sum, list(c(1, 2, 4, 1, 2, NA), na.rm = TRUE))
[1] 10
> # lapply
> lapply(c(1, 2, 4, 1, 2, NA), function(x) x + 1)
 [[1]]
[1] 2

[[2]]
[1] 3

[[3]]
[1] 5

[[4]]
[1] 2

[[5]]
[1] 3

[[6]]
[1] NA
```


√ Gráfico de regressões por `Species`:

```
# grafico das regressões por Species
par(mfrow = c(1, 3))
by(iris, iris$Species, function(x){
                # Plot para cada espécie
                plot(x$Petal.Length ~ x$Petal.Width, xlab = "Largura de petála",
                       ylab = "Comprimento de pétala", ylim = c(1, 7))
                abline(lm(Petal.Length ~ Petal.Width, x), lty = 2, col = "red",
                         lwd = 2)
                title(main = unique(x$Species))
        }
)
```



√ Loop dos gráficos usando a função `by`

√ Colocando os resultados em data frame

```
> # cálculo das médias das variáveis por espécies
> iris.medias <- by(iris[-5], iris$Species, function(x) colMeans(x))
> iris.medias
iris$Species: setosa
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
       5.006        3.428        1.462        0.246
------------------------------------------------------------
iris$Species: versicolor
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
       5.936        2.770        4.260        1.326
------------------------------------------------------------
iris$Species: virginica
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
       6.588        2.974        5.552        2.026
> # saaply para colocar as médias em uma matriz
> (iris.medias2 <- t(sapply(iris.medias, I)))
           Sepal.Length Sepal.Width Petal.Length Petal.Width
setosa            5.006       3.428        1.462       0.246
versicolor        5.936       2.770        4.260       1.326
virginica         6.588       2.974        5.552       2.026
> # criando um data frame
> iris.new <- as.data.frame(iris.medias2)
> is.data.frame(iris.new)
[1] TRUE
```


X, INDEX

Comando `aggregate`

- Aplica função a estratos de um data frame
  - √ Saída é um data frame
- Sintaxe:

  `aggregate(X, by, FUN, ...)`

  √ `X`: data frame ou matriz

  √ `by`: fator ou lista de fatores.

  √ `FUN`: função a ser aplicada

  √ `...` : argumentos opcionais da função

• Exemplo – `iris`:

```
>   # exemplo - iris
> iris.x <- subset(iris, select = -Species)
> iris.s <- subset(iris, select = Species)
> aggregate(iris.x, iris.s, mean)
     Species Sepal.Length Sepal.Width Petal.Length Petal.Width
1     setosa        5.006       3.428        1.462       0.246
2 versicolor        5.936       2.770        4.260       1.326
3  virginica        6.588       2.974        5.552       2.026
> # interface por fórmula
> aggregate( . ~ Species, iris, mean)
           Sepal.Length Sepal.Width Petal.Length Petal.Width
setosa            5.006       3.428        1.462       0.246
versicolor        5.936       2.770        4.260       1.326
virginica         6.588       2.974        5.552       2.026
```


√ Média e desvio padrão – `iris`:

```
> # função definida pelo usuário
> # calculo de mais de uma função por estrato
> ag <- aggregate(. ~ Species, iris, function(x) c(mean = mean(x), sd = sd(x)))
> is.data.frame(ag)
[1] TRUE
> dim(ag)
[1] 3 5
> ag
     Species Sepal.Length.mean Sepal.Length.sd Sepal.Width.mean Sepal.Width.sd
1     setosa         5.0060000       0.3524897        3.4280000      0.3790644
2 versicolor         5.9360000       0.5161711        2.7700000      0.3137983
3  virginica         6.5880000       0.6358796        2.9740000      0.3224966
  Petal.Length.mean Petal.Length.sd Petal.Width.mean Petal.Width.sd
1         1.4620000       0.1736640        0.2460000      0.1053856
2         4.2600000       0.4699110        1.3260000      0.1977527
3         5.5520000       0.5518947        2.0260000      0.2746501
> ag[[1]]
[1] setosa     versicolor virginica
Levels: setosa versicolor virginica
> ag[-1][[4]]
 mean        sd
[1,] 0.246 0.1053856
[2,] 1.326 0.1977527
[3,] 2.026 0.2746501
```


√ Comandos `split` e `apply` – `iris`:

```
> # comandos split e apply
> sapply(split(iris.x, iris.s), function(x) apply(x, 2, mean))
             setosa versicolor virginica
Sepal.Length  5.006      5.936     6.588
Sepal.Width   3.428      2.770     2.974
Petal.Length  1.462      4.260     5.552
Petal.Width   0.246      1.326     2.026
```

- √ Dados são divididos em subconjuntos para cada uma das espécies
  - Cálculo da média para cada coluna de cada subconjunto
- √ Saída é uma matriz
  - Transposta da saída anterior

• Exemplo – datas:

```
>   # exemplo - datas
> datas <- data.frame(data = as.Date("01-01-2018", format = "%d-%m-%Y") + 0:729)
> head(datas)
 data
1 2018-01-01
2 2018-01-02
3 2018-01-03
> mean(datas$data)
[1] "2018-12-31"
> as.Date("28/08/1991", format = "%d/%m/%Y") - as.Date("17/01/1991",
+  format = "%d/%m/%Y")
Time difference of 223 days
> as.Date("31/07/2017", format = "%d/%m/%Y") + 223
[1] "2018-03-11"
> # último dia de cada mês
> ultimo <- aggregate(x = datas["data"],
+                     by = list(mes = substr(datas$data, 1, 7)),
+                     FUN = mean)
> head(ultimo)
      mes       data
1 2018-01 2018-01-16
2 2018-02 2018-02-14
3 2018-03 2018-03-16
```

```
> substr("abcdef", 2, 4)
[1] "bcd"
```

• Exemplo – mtcars:

```
> options(digits=3)
> mtcars.medias <-aggregate(mtcars, by = list(mtcars$cyl, mtcars$ gear),
+ FUN = mean, na.rm = TRUE)
> mtcars.medias
 Group.1 Group.2  mpg cyl disp  hp drat   wt qsec  vs   am gear carb
1       4       3 21.5   4  120  97 3.70 2.46 20.0 1.0 0.00    3 1.00
2       6       3 19.8   6  242 108 2.92 3.34 19.8 1.0 0.00    3 1.00
3       8       3 15.1   8  358 194 3.12 4.10 17.1 0.0 0.00    3 3.08
4       4       4 26.9   4  103  76 4.11 2.38 19.6 1.0 0.75    4 1.50
5       6       4 19.8   6  164 116 3.91 3.09 17.7 0.5 0.50    4 4.00
6       4       5 28.2   4  108 102 4.10 1.83 16.8 0.5 1.00    5 2.00
7       6       5 19.7   6  145 175 3.62 2.77 15.5 0.0 1.00    5 6.00
8       8       5 15.4   8  326 300 3.88 3.37 14.6 0.0 1.00    5 6.00
> # nome dos grupos
> aggregate(mtcars[, -c(2, 10)], by = list(Cilindros = mtcars$cyl,
+ Marcha = mtcars$gear), FUN = mean, na.rm = TRUE)
> # interface por fórmula
> aggregate(. ~ cyl + gear, mtcars, mean, na.rm = T)
> options(digits = 7)
```


• Exemplo – ChickWeight:

```
> help(ChickWeight)
> dim(ChickWeight)
[1] 578   4
> head(ChickWeight)
  weight Time Chick Diet
1     42    0     1    1
2     51    2     1    1
3     59    4     1    1
4     64    6     1    1
5     76    8     1    1
6     93   10     1    1
> is.data.frame(ChickWeight)
[1] TRUE
```


√ Estrutura do conjunto de dados

```
> str(ChickWeight)
Classes 'nfnGroupedData', 'nfGroupedData', 'groupedData' and
'data.frame':      578 obs. of  4 variables:
 $ weight: num  42 51 59 64 76 93 106 125 149 171 ...
 $ Time  : num  0 2 4 6 8 10 12 14 16 18 ...
 $ Chick : Ord.factor w/ 50 levels "18"<"16"<"15"<..: 15 15 15 15 15 15 ...
 $ Diet  : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
 - attr(*, "formula")=Class 'formula'  language weight ~ Time | Chick
  .. ..- attr(*, ".Environment")=<environment: R_EmptyEnv>
 - attr(*, "outer")=Class 'formula'  language ~Diet
  .. ..- attr(*, ".Environment")=<environment: R_EmptyEnv>
 - attr(*, "labels")=List of 2
  ..$ x: chr "Time"
  ..$ y: chr "Body weight"
 - attr(*, "units")=List of 2
  ..$ x: chr "(days)"
  ..$ y: chr "(gm)"
```

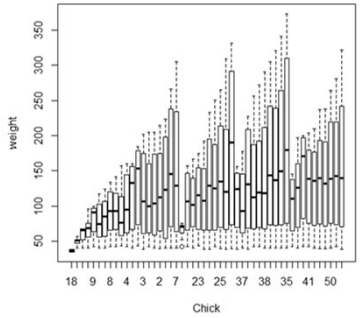

√ Exploração – ChickWeight:

```
> # quantidade de pintinhos
> unique(ChickWeight$Chick)
[1] 1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
50 Levels: 18 < 16 < 15 < 13 < 9 < 20 < 10 < 8 < 17 < 19 < 4 < 6 < 11 < ... < 48
> length(unique(ChickWeight$Chick))
[1] 50
> # quantidade de dietas
> unique(ChickWeight$Diet)
[1] 1 2 3 4
Levels: 1 2 3 4
> is.numeric(unique(ChickWeight$Diet))
[1] FALSE
> # quantidade de instantes de tempos
> unique(ChickWeight$Time)
[1]  0  2  4  6  8 10 12 14 16 18 20 21
> is.numeric(unique(ChickWeight$Time))
[1] TRUE
```

√ Box-plot `weight` por `Chick`:

```
> # box-plot por peso por pintinho
> plot(weight ~ Chick, ChickWeight)
```

√ Os crescimentos são iguais para cada pintinho?

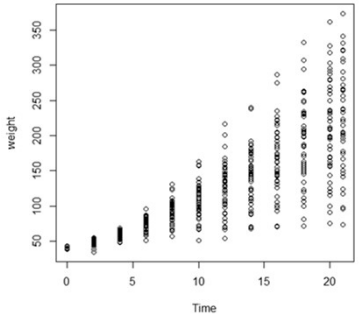


121

√ Plot `weight` por `Time`:

```
> # relação entre peso e tempo
> plot(weight ~ Time, ChickWeight)
```

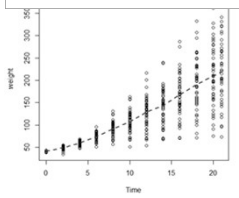
√ Como é a curva de crescimento média desses pintinhos?



122

√ Linha média de crescimento:

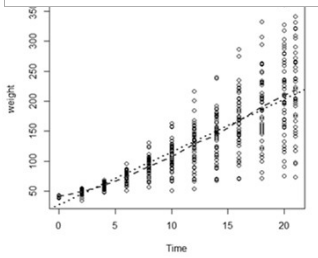
```
> # peso médio por instante de tempo
> tapply(ChickWeight$weight, ChickWeight$Time, mean)
> length(unique(ChickWeight$Chick))
       0        2        4        6        8       10       12       14
 41.06000  49.22000  59.95918  74.30612  91.24490 107.83673 129.24490 143.81250
      16       18       20       21
168.08511 190.19149 209.71739 218.68889
> peso.medio <- with(ChickWeight, tapply(weight, Time, mean))
> instantes <- unique(ChickWeight$Time)
> is.numeric(instantes)
[1] TRUE
> # ganho médio de peso (todos os pintinhos)
> lines(peso.medio ~ instantes, lty = 2, lwd = 2, col = "red")
```



123

√ Linha de regressão `weight` vs. `Time`:

```
> # linha de regressão
> abline(lm(weight ~ Time, data = ChickWeight), lty = 3, lwd = 2, col =
"blue")
> # parâmetros da regressão
> lm(weight ~ Time, data = ChickWeight)
Call:
lm(formula = weight ~ Time, data = ChickWeight)
Coefficients:
(Intercept)         Time
     27.467        8.803
```



√ A linha média de crescimento e a reta de regressão são próximas?

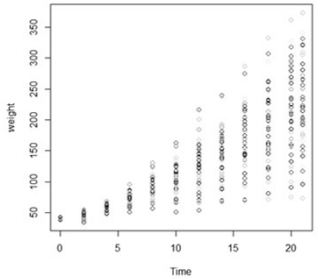
√ Qual a taxa de aumento de peso por unidade de tempo?

– Em que regiões do gráfico você acha que essa estimativa é mais imprecisa

124

√ Ganho de peso por pintinho:

```
> # ganho de peso por pintinho
> plot(weight ~ Time, ChickWeight, col = as.numeric(Chick))
```



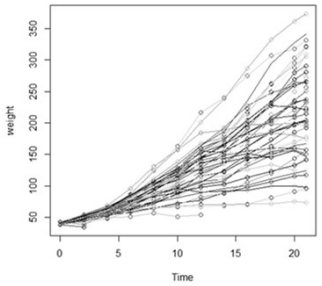
√ Difícil perceber o crescimento de cada pintinho

√ Ideal:

- Gráfico sequencial

√ Gráfico sequencial por pintinho:

```
> # Gráfico para cada pintinho
> pintinho <- levels(ChickWeight$Chick)
> linhas.pintinho <- function(i){
+ with(ChickWeight[ChickWeight$Chick == pintinho[i], ],
+ lines(Time, weight, col = as.numeric(pintinho[i])))
+ }
> invisible(lapply(seq_along(pintinho), linhas.pintinho))
```

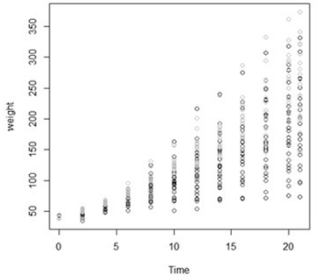


√ O que poderia ser acrescentado ao gráfico?

√ Há outra maneira para construir esse gráfico?

√ Visualização do efeito das dietas:

```
> # ganho de peso por dieta
> plot(weight ~ Time, ChickWeight,
+ col = c('red', 'blue', 'green', "gray")[as.numeric(ChickWeight$Diet)])
```



√ As dietas têm efeitos diferentes no crescimento dos pintinhos?

√ Estatísticas por dieta tempo:

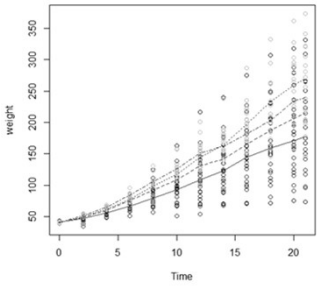
```
> # média de peso por dieta
> with(ChickWeight, aggregate(weight, list(dieta = Diet), mean))
  dieta        x
1     1 102.6455
2     2 122.6167
3     3 142.9500
4     4 135.2627
> # desvio padrão por dieta
> with(ChickWeight, aggregate(weight, list(dieta = Diet), sd))
  dieta        x
1     1 56.65655
2     2 71.60749
3     3 86.54176
4     4 68.82871
> #peso médio por instante de tempo
> aggregate(weight ~ Time, data = ChickWeight, mean)
   Time    weight
1     0  41.06000
2     2  49.22000
3     4  59.95918
4     6  74.30612
5     8  91.24490
6    10 107.83673
7    12 129.24490
8    14 143.81250
```

√ Média por instante de tempo e dieta:

```
> # peso médio por dieta e por instante de tempo
> aggregate(weight ~ Time + Diet, data = ChickWeight, mean)
Time Diet    weight
1     0    1  41.40000
2     2    1  47.25000
3     4    1  56.47368
4     6    1  66.78947
5     8    1  79.68421
6    10    1  93.05263
7    12    1 108.52632
8    14    1 123.38889
9    16    1 144.64706
10   18    1 158.94118
11   20    1 170.41176
12   21    1 177.75000
13    0    2  40.70000
14    2    2  49.40000
15    4    2  59.80000
16    6    2  75.40000
17    8    2  91.70000
18   10    2 108.50000
19  ...
```


√ Gráficos dos crescimentos médios por dieta:

```
> # gráfico das linhas de crescimento médio por dieta
> dieta.media <- with(ChickWeight, tapply(weight, list(Time, Diet), mean))
> invisible(lapply(1:4, function(x) lines(dieta.media[, x] ~ instantes, lty = x,
          col = c('red', 'blue', 'green', "gray")[as.numeric(ChickWeight$Diet)])))
```



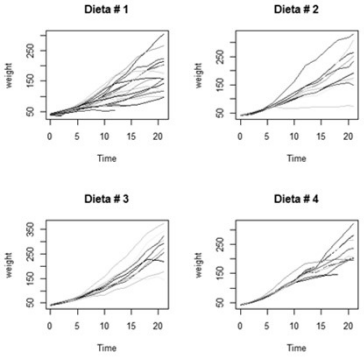
- √ Os crescimentos médios são diferentes?
- √ Em média, qual dieta leva a um crescimento mais rápido?

√ Crescimentos por dieta e por pintinhos:

```
> # Gráficos dos crescimentos por dieta e por pintinhos
> dados <- split(ChickWeight, ChickWeight$Diet)
> is.list(dados)
[1] TRUE
> # função para traçar uma linha por pintinho
> cada.linha <- function(data, individuo, i){
+ with(data[data$Chick == individuo[i], ],
+ lines(Time, weight, col = as.numeric(individuo[i])))
+ }
> # função para traçar linhas dos pintinhos por dieta
> linhas.dieta <- function(x){
+ pintinhos <- unique(x$Chick)
+ plot(weight ~ Time, data = x, type = "n")
+ invisible(lapply(seq_along(pintinhos), FUN = function(i) {
+ cada.linha(data = x, individuo = pintinhos, i)}
+ ))
+ title(main = paste("Dieta #", unique(x$Diet)))
+
+ }
```


√ Gráficos dos crescimentos por dieta e por pintinhos:

```
> # gráficos dos crescimentos separados por dieta
> par(mfrow = c(2, 2))
> invisible(lapply(dados, linhas.dieta))
```



- √ Como você escreveria esse código?
- √ O que fazer para melhorar os gráficos?
- √ O que poderia ser acrescentado?

X, INDEX

Comando `rapply`

- Aplica recursivamente função a todos os elementos de uma lista
  - √ Saída é um vetor ou uma lista
- Sintaxe:

```
rapply(object, f, ...)
```

  - √ `object`: lista
  - √ `f`: função com um único argumento
  - √ `class`: classes a que função deve ser aplicada
  - √ `...` : argumentos opcionais da função

- Exemplo 1:
  - √ Função e `how = "unlist"` (default):

```
>   # exemplo 1
> lista <- list(a = 1:10, b = 11:20)
> # função aplicada a cada elemento da lista - saída vetor
> rapply(lista, mean)
   a    b
 5.5 15.5
```

  - √ Argumento `how = "list"`

```
> # função aplicada a cada elemento da lista - saída lista
> rapply(lista, mean, how = "list")
$a
[1] 5.5

$b
[1] 15.5
```


- √ `how = "unlist"` (default):

```
> # log2 de cada valor na lista
> rapply(lista, log2)
      a1       a2       a3       a4       a5       a6       a7       a8
0.000000 1.000000 1.584963 2.000000 2.321928 2.584963 2.807355 3.000000
      a9      a10       b1       b2       b3       b4       b5       b6
3.169925 3.321928 3.459432 3.584963 3.700440 3.807355 3.906891 4.000000
      b7       b8       b9      b10
4.087463 4.169925 4.247928 4.321928
```

- √ Argumento `how = "list"`

```
> # log2 de cada elemento em cada lista
> rapply(lista, log2, how = "list")
$a
 [1] 0.000000 1.000000 1.584963 2.000000 2.321928 2.584963 2.807355 3.000000
 [9] 3.169925 3.321928

$b
 [1] 3.459432 3.584963 3.700440 3.807355 3.906891 4.000000 4.087463 4.169925
 [9] 4.247928 4.321928
```


- Exemplo 2
  - √ Argumento `class`:

```
>   # exemplo 2
> lista.1 <- list(1, 2, 3, 4)
> rapply(lista.1, function(x){x^2}, class = c("numeric"))
[1]  1  4  9 16
> # lista com elemento de classe não especificada
> lista.2 <- list(1, 2, 3, 4, "a")
> rapply(lista.2, function(x){x^2}, class = c("numeric"))
[1]  1  4  9 16
```

  - √ Aplica função apenas aos elementos `numeric`

√ Argumento `how`:

```
>   # exemplo 2 - lista com sublistas
> lista.3 <- list(1, list(2, 3), 4, list(5, list(6, 7)))
> str(lista.3)
List of 4
 $ : num 1
 $ :List of 2
  ..$ : num 2
  ..$ : num 3
 $ : num 4
 $ :List of 2
  ..$ : num 5
  ..$ :List of 2
  .. ..$ : num 6
  .. ..$ : num 7
> # saida é vetor
> rapply(lista.3, function(x){x^2}, class = c("numeric"))
[1]  1  4  9 16 25 36 49
> # saida é lista (estrutura original)
> objeto.1 <- rapply(lista.3, function(x){x^2}, class = c("numeric"),
+  how = "list")
> str(objeto.1)
List of 4
 $ : num 1
 $ :List of 2
  ...
```


- Exemplos

√ Argumento `deftl`:

```
>   # exemplo 3 - substituição
> lista.4 <- list(1, list(2), "f")
> objeto.2 <- rapply(lista.4, function(x){x^2}, class = c("numeric"),
+   how = "list", deflt = "p")
> objeto.2
[[1]]
[1] 1

[[2]]
[[2]][[1]]
[1] 4


[[3]]
[1] "p"
```

√ Se a lista contiver um elemento de uma classe diferente, ele será substituído pelo elemento especificado no argumento `deflt`

√ Argumento `how = "unlist"`:

```
> # saida não é lista
> objeto.3 <- rapply(lista.4, function(x){x^2}, class = c("numeric"),
+ how = "unlist", deflt = "p")
> objeto.3
[1] "1" "4" "p"
> is.vector(objeto.3)
[1] TRUE
```

√ Argumento `how = "replace"`:

```
> # mantém valores originais da classe não "numeric"
> objeto.4 <- rapply(lista.4, function(x){x^2}, class = c("numeric"),
+ how = "replace", deflt = "p")
> objeto.4
[1]]
[1] 1

[[2]]
[[2]][[1]]
[1] 4

[[3]]
[1] "f"
```


√ Outros argumentos da função a ser aplicada:

```
> # outros argumentos de f (função a ser aplicada)
> objeto.5 <- rapply(lista.4, function(x, y){x^y}, class =
c("numeric"),
+ how = "unlist", deflt = "p", y = 3)
> objeto.5
[1] "1" "8" "p"
> is.vector(objeto.5)
[1] TRUE
```

X, INDEX

Comando `vapply`

- Similar a `sapply`,
  - √ Requer especificação do tipo de dados desejados na saída
  - √ Saída é um vetor ou matriz
- Sintaxe:

```
vapply(X, FUN, FUN.VALUE, ...)
```

  - √ `X`: vetor ou lista
  - √ `FUN`: função com um único argumento
  - √ `FUN.VALUE`: tipo de dados desejados
  - √ `...` : argumentos opcionais da função

- Exemplo 1:
  - √ Vetores e listas:

```
>   # exemplo 1
> # saida numerica - vetor
> vetor.v <- 1:10
> vapply(vetor.v, sum, numeric(1))
[1]  1  2  3  4  5  6  7  8  9 10
> # saida numérica - lista
> A <- 1:9
> B <- 1:12
> C <- 1:15
> lista.v <- list(A, B, C)
> vapply(lista.v, sum, numeric(1))
[1]  45  78 120
```

  - √ Saída será um erro se função retornar mais de um valor numérico
    - `FUN.VALUE = numeric (1)` pode ser útil caso se espere apenas um resultado elemento

- Exemplo 2:

```
>    # exemplo 2
> lista.v2 <- list(a = rnorm(10), b = rnorm(10))
> # fivenum dos valores
> cinco <- vapply(lista.v3, fivenum,
+ c(Min. = 0, "1o. Qu." = 0, Mediana = 0, "3o. Qu." = 0, Max. = 0))
> class(cinco)
[1] "matrix"
> cinco
                a          b
Min.    -1.4534842 -0.8821279
1o. Qu. -0.6873098 -0.2106322
Mediana -0.1527475  0.6392059
3o. Qu.  0.4422310  0.9666165
Max.     1.4889964  1.8130571
```

  - √ Saída é uma matriz
    - Nomes das colunas: elementos da lista original
    - Nomes das linhas: *template* da saída

X, INDEX

Comando `mapply`

- Versão multivariada de `lapply` e `sapply`
  - √ Caso `lapply` e `sapply`:
    - Função atua somente sobre os elementos de uma única lista
  - √ Caso `mapply`
    - Função é aplicada sobre o 1º elemento de cada um dos argumentos, em seguida ao 2º, etc.
    - Argumentos podem ser listas ou vetores
    - Saída é um vetor ou matriz

X, INDEX

- Sintaxe:

```
mapply(FUN, ..., MoreArgs = NULL,
  SIMPLIFY = TRUE, USE.NAMES = TRUE)
```

  - √ `FUN`: função a ser aplicada
  - √ `...`: argumentos para vetorização (vetores ou listas)
  - √ `MoreArgs`: argumentos adicionais de `FUN (lista)`
  - √ `USE.NAMES`: se primeiro argumento (…) tem nomes
  - √ `SIMPLIFY`: tenta reduzir resultado a um vetor, matriz ou array de dimensão mais alta

- Exemplo 1:
  - √ Soma de vetores:

```
> # exemplo 1
> a <- 1:4
> b <- 5:8
> mapply(sum, a, b)
[1]  6  8 10 12
> sapply(list(a, b), sum)
[1] 10 26
> mapply(sum, list(a, b))
[1] 10 26
```


- Exemplo 2:
  - √ `paste` em vetores de caracteres:

```
> # exemplo 2
> L1 <- list(a = LETTERS[c(4,6,12,6)], b = LETTERS[c(1,5,21,1)])
> L1
$a
[1] "D" "F" "L" "F"

$b
[1] "A" "E" "U" "A"
> L2 <- list(c = LETTERS[c(4,14,22,20)], d = LETTERS[c(15,15,1,15)])
> L2
$c
[1] "D" "N" "V" "T"

$d
[1] "O" "O" "A" "O"
> mapply(paste, L1$a, L1$b, L2$c, L2$d)
        D         F         L         F
"D A D O" "F E N O" "L U V A" "F A T O"
```


- Exemplo 3:
  - √ Soma em listas:

```
> # exemplo 3
> (L3 <- list(a = 1:10, b = 11:20))
$a
 [1]  1  2  3  4  5  6  7  8  9 10

$b
 [1] 11 12 13 14 15 16 17 18 19 20
> (L4 <- list(c = 21:30, d = 31:40))
$c
 [1] 21 22 23 24 25 26 27 28 29 30

$d
 [1] 31 32 33 34 35 36 37 38 39 40
> mapply(sum, L3$a, L3$b, L4$c, L4$d)
[1]  64  68  72  76  80  84  88  92  96 100
> mapply(sum, L3, L4)
  a   b
310 510
> sum(L3$a, L4$c)
[1] 310
> sum(L3$b, L4$d)
[1] 510
```

- Exemplo 4 – criando e salvando variável:

```
> # exemplo 4 - criando nova variável
> dados <- as.data.frame(matrix(c(1:10, 11:20, 21:30), nrow = 10, ncol = 3))
> colnames(dados)
[1] "V1" "V2" "V3"
>   # criando nova variável
> dados$V4 <- mapply(function(x, y) x/y, x = dados$V1, y = dados$V3)
> dados$V4
[1] 0.04761905 0.09090909 0.13043478 0.16666667 0.20000000 0.23076923
[7] 0.25925926 0.28571429 0.31034483 0.33333333
> dados
   V1 V2 V3         V4
1   1 11 21 0.04761905
2   2 12 22 0.09090909
3   3 13 23 0.13043478
4   4 14 24 0.16666667
5   5 15 25 0.20000000
6   6 16 26 0.23076923
7   7 17 27 0.25925926
8   8 18 28 0.28571429
9   9 19 29 0.31034483
10 10 20 30 0.33333333
>   # salvando resultado em objeto
> vetor <- vector(mode = "numeric", length = 10)
> (vetor <- mapply(function(x, y) x*y, dados$V2, dados$V3))
[1] 231 264 299 336 375 416 459 504 551 600
```


- Exemplo 5 – `state`:

```
> # exemplo 5 - aplicando em conjuntos de dados
> library(MASS)
> help(state)
> head(state.x77, 4)
           Population Income Illiteracy Life Exp Murder HS Grad Frost   Area
Alabama          3615   3624        2.1    69.05   15.1    41.3    20  50708
Alaska            365   6315        1.5    69.31   11.3    66.7   152 566432
Arizona          2212   4530        1.8    70.55    7.8    58.1    15 113417
Arkansas         2110   3378        1.9    70.66   10.1    39.9    65  51945
> dim(state.x77)
[1] 50  8
> str(state.x77)
num [1:50, 1:8] 3615 365 2212 2110 21198 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:50] "Alabama" "Alaska" "Arizona" "Arkansas" ...
  ..$ : chr [1:8] "Population" "Income" "Illiteracy" "Life Exp" ...
> populacao <- state.x77[1:50]
> area <- state.area
> pop.dens <- mapply(function(x, y) x/y, populacao, area
 [1]  51609 589757 113909  53104 158693 104247   5009   2057  58560  58876
[11]   6450  83557  56400  36291  56290  82264  40395  48523  33215  10577
[21]   8257  58216  84068  47716  69686 147138  77227 110540   9304   7836
[31] 121666  49576  52586  70665  41222  69919  96981  45333   1214  31055
[41]  77047  42244 267339  84916   9609  40815  68192  24181  56154  97914
```


X. INDEX

Comando `sweep`

- Aplica funções diferentes nos elementos `MARGIN` de uma matriz
  - √ Aplicação de função em linha (coluna) de matriz, quando o outro argumento da função tem valor diferente para cada linha (coluna)
  - √ Saída é um vetor ou matriz

X. INDEX

- Sintaxe:

  `sweep(x, MARGIN, STATS, FUN="-",...)`

  - √ `X`: array
  - √ `MARGIN`: 1 = linha, 2 = coluna
  - √ `STATS`: estatística do elemento escolhido
  - √ `FUN`: função que aplicará STATS
  - √ `...` : argumentos opcionais da função

- Exemplo:

```
>   # exemplo 1
> # geração de matriz do exemplo
> (dados <- matrix(1:12, ncol = 4, nrow = 3, byrow = TRUE))
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
> # algumas estatísticas por elemento da matriz
> (mediaCol <- apply(dados, 2, mean))
[1] 5 6 7 8
> colMeans(dados)
[1] 5 6 7 8
> (prodCol <- apply(dados, 2, prod))
[1]  45 120 231 384
> (medianaCol <- apply(dados, 2, median))
[1] 5 6 7 8
> (dpCol <- apply(dados, 2, sd))
[1] 4 4 4 4
> (madCol <- apply(dados, 2, mad, constant = 1))
[1] 4 4 4 4
```

$$\text{mad} = \text{mediana}\left(|x_i - \tilde{x}|\right)$$

√ Função `sweep`:

```
> # função sweep
> sweep(dados, 2, c(3, 4, 5, 6), "-")
     [,1] [,2] [,3] [,4]
[1,]   -2   -2   -2   -2
[2,]    2    2    2    2
[3,]    6    6    6    6
> sweep(dados, 2, mediaCol, "-")
[1] 5 6 7 8
> sweep(dados, 2, prodCol, "/")
          [,1]       [,2]       [,3]       [,4]
[1,] 0.02222222 0.01666667 0.01298701 0.01041667
[2,] 0.11111111 0.05000000 0.03030303 0.02083333
[3,] 0.20000000 0.08333333 0.04761905 0.03125000
> sweep(dados, 2, mediaCol, ">")
      [,1]  [,2]  [,3]  [,4]
[1,] FALSE FALSE FALSE FALSE
[2,] FALSE FALSE FALSE FALSE
[3,]  TRUE  TRUE  TRUE  TRUE
```


√ Padronização:

$$z_{ij} = \frac{x_{ij} - \bar{x}_j}{s_j}$$

```
> # padronização de dados centrado na média e escalado no dp
> (dados.centrados <- sweep(dados, 2, mediaCol, "-"))
     [,1] [,2] [,3] [,4]
[1,]   -4   -4   -4   -4
[2,]    0    0    0    0
[3,]    4    4    4    4
> (dados.padronizados <- sweep(dados.centrados, 2, dpCol, "/")
     [,1] [,2] [,3] [,4]
[1,]   -1   -1   -1   -1
[2,]    0    0    0    0
[3,]    1    1    1    1
```


√ Padronização – alternativa:

– Comando `scale`:

```
> # padronização - alternativa
> scale(dados)
 [,1] [,2] [,3] [,4]
[1,]   -1   -1   -1   -1
[2,]    0    0    0    0
[3,]    1    1    1    1
attr(,"scaled:center")
[1] 5 6 7 8
attr(,"scaled:scale")
[1] 4 4 4 4
> attr(dados.escalados, "scaled:scale")
[1] 4 4 4 4
> dados.escalados[, ]
[,1] [,2] [,3] [,4]
[1,]   -1   -1   -1   -1
[2,]    0    0    0    0
[3,]    1    1    1    1
> attr(dados.escalados, "scaled:center") <- NULL
> attr(dados.escalados, "scaled:scale") <- NULL
> dados.escalados
> scale(dados, scale = F)
> scale(dados, center = c(3, 4, 5, 6), scale = F)
```

• Exemplo – attitude:

```
>   # exemplo - attitude
> mediana.att <- apply(attitude, 2, median)
> mad.att <- apply(attitude, 2, mad, constant = 1)
> att.centrada <- sweep(attitude, 2, mediana.att, FUN = "-")
> att.padronizada <- sweep(att.centrada, 2, mad.att, "/")
> head(att.padronizada)
      rating complaints  privileges learning      raises critical    advance
1 -3.2142857       -1.4 -3.07142857    -1.75 -0.33333333      2.9  0.6666667
2 -0.3571429       -0.1 -0.07142857    -0.25 -0.06666667     -0.9  1.0000000
3  0.7857143        0.5  2.35714286     1.25  1.66666667      1.7  1.1666667
4 -0.6428571       -0.2 -0.92857143    -0.95 -1.26666667      1.3 -1.0000000
5  2.2142857        1.3  0.64285714     0.95  1.00000000      1.1  1.0000000
6 -3.2142857       -1.0 -0.35714286    -1.25 -1.26666667     -5.7 -1.1666667
> round(head(scale(attitude)), 5)
       rating complaints privileges learning   raises critical  advance
[1,] -1.77722   -1.17163   -1.89068 -1.47965 -0.34945  1.74164  0.20087
[2,] -0.13418   -0.19527   -0.17436 -0.20164 -0.15709 -0.17854  0.39526
[3,]  0.52303    0.25536    1.21505  1.07637  1.09324  1.13526  0.49245
[4,] -0.29849   -0.27038   -0.66474 -0.79805 -1.02271  0.93314 -0.77107
[5,]  1.34455    0.85619    0.23429  0.82077  0.61234  0.83208  0.39526
[6,] -1.77722   -0.87121   -0.33782 -1.05365 -1.02271 -2.60403 -0.86827
```


• Exemplo – iris3:

```
>   # exemplo - iris
> is.array(iris3)
[1] TRUE
> dim(iris3)
[1] 50  4  3
> dimnames(iris3)
[[1]]
NULL
[[2]]
[1] "Sepal L." "Sepal W." "Petal L." "Petal W."
[[3]]
[1] "Setosa"     "Versicolor" "Virginica"
> iris3[1:4, , 1]
     Sepal L. Sepal W. Petal L. Petal W.
[1,]      5.1      3.5      1.4      0.2
[2,]      4.9      3.0      1.4      0.2
[3,]      4.7      3.2      1.3      0.2
[4,]      4.6      3.1      1.5      0.2
> iris3[1:4, , "Versicolor"]
Sepal L. Sepal W. Petal L. Petal W.
[1,]      7.0      3.2      4.7      1.4
[2,]      6.4      3.2      4.5      1.5
[3,]      6.9      3.1      4.9      1.5
[4,]      5.5      2.3      4.0      1.3
```


√ Família apply – iris3:

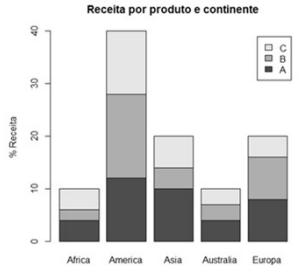
```
> # comandos apply e sweep
> iris.medias <- apply(iris3, c(2, 3), mean)
> iris.medias
         Setosa Versicolor Virginica
Sepal L.  5.006      5.936     6.588
Sepal W.  3.428      2.770     2.974
Petal L.  1.462      4.260     5.552
Petal W.  0.246      1.326     2.026
> iris3.centrada <- sweep(iris3, c(2,3), iris.medias, "-")
> iris3.centrada[1:2, , ]
, , Setosa
     Sepal L. Sepal W. Petal L. Petal W.
[1,]    0.094    0.072   -0.062   -0.046
[2,]   -0.106   -0.428   -0.062   -0.046
, , Versicolor
     Sepal L. Sepal W. Petal L. Petal W.
[1,]    1.064     0.43     0.44    0.074
[2,]    0.464     0.43     0.24    0.174
, , Virginica
     Sepal L. Sepal W. Petal L. Petal W.
[1,]   -0.288    0.326    0.448    0.474
[2,]   -0.788   -0.274   -0.452   -0.126
```


• Receitas globais de produtos:

```
>   # exemplo - Receita por produto e continente
> # montagem do conjunto de dados
> produto <- c("A", "B", "C", "Total")
> continente <- c("Africa", "America", "Asia", "Australia", "Europa")
> valores <- c(0.4, 0.2, 0.4, 0.1, 0.3, 0.4, 0.3, 0.4, 0.5, 0.2,
+              0.3, 0.2, 0.4, 0.3, 0.3, 0.1, 0.4, 0.4, 0.2, 0.2)
> (receitas <- matrix(valores, ncol = 5, dimnames = list(produto, continente)))
      Africa America Asia Australia Europa
A        0.4     0.3  0.5       0.4    0.4
B        0.2     0.4  0.2       0.3    0.4
C        0.4     0.3  0.3       0.3    0.2
Total    0.1     0.4  0.2       0.1    0.2
> # receitas absolutas
> (receitas.global <- sweep(receitas[1:3, ], 2, receitas[4, ], "*"))
Africa America Asia Australia Europa
A   0.04    0.12 0.10      0.04   0.08
B   0.02    0.16 0.04      0.03   0.08
C   0.04    0.12 0.06      0.03   0.04
> sum(receitas.global)
[1] 1
```

– Gráfico das receitas

```
> # gráfico das receitas
> barplot(receitas.global*100, legend = rownames(receitas.global),
+         main = "Receita por produto e continente", ylab = "% Receita")
```



√ E se você quiser utilizar os valores da Europa como baseline?

161

Estatística Computacional I - 2020

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.

286

Estatística Computacional I - 2020