




Amostragem e Análise Ambiental

Introdução ao  — breve revisão dirigida à Estatística

Manuela Neves

ISA/ULisboa

2020/2021


- 1 Breve introdução ao ambiente 
- 2 Estrutura e manipulação de dados
 - Objectos no  - Vector, Matrix, Factor, List, Data Frame
 - Leitura e escrita de ficheiros
- 3 Funções e Programação em 

Porquê a escolha do *software* estatístico ?

- **O que é o  ?**

- É um conjunto integrado de ferramentas computacionais que permitem **a manipulação e análise estatística de dados**, **o cálculo numérico** e **a produção de gráficos**.
- É uma linguagem **interpretada** - os comandos são imediatamente executados.
- É uma linguagem orientada por **objectos** - os dados são armazenados na memória activa do computador na forma de objectos, têm nome e sobre eles aplicam-se acções.


● O que é o ?



- É uma linguagem de programação, permite por isso implementar e tratar novos algoritmos.
- Está em actualização permanente pela introdução constante de novos e diversos procedimentos estatísticos.
- É uma aplicação de distribuição gratuita e de código público disponível em (<http://cran.r-project.org/>) - - aqui existe toda a informação.
- Depois de fazer o *download* da versão adequada ao sistema operativo do computador (por ex. R-3.3.3- win32.exe, para o Windows), para instalar o  basta *executar* esse ficheiro.

● Vantagens do R ?

- É mesmo totalmente gratuito.
- Resulta de uma colaboração internacional de vários investigadores, que mantêm uma rede de discussão na *internet*.
- É possível corrigir com mais facilidade os *bugs* detectados e até obter ajuda para tentar resolver algo mais específico.
- Na *homepage do R*, www.r-project.org encontram-se vários tutoriais, em várias linguas.
- É possível desenvolver um módulo (*package*) para uma aplicação de interesse, torná-la disponível no R e assim poder partilhar conhecimento.
- O R tem várias *mailing-lists* para uma grande variedade de temas, ver a *homepage do R*.

- **Iniciar e terminar uma sessão de R**
 - Criar uma pasta (ex. **MTCSII**) onde irão guardar-se todos os ficheiros de trabalho - ficheiros de dados, ficheiros de resultados, ficheiros do R.
 - Iniciar o R - abre-se uma janela de trabalho.
 - Especificar logo a pasta de trabalho
menu: **File** → **Change dir ...**
 - Os comandos são dados à frente da **prompt >** e são executados após pressionar **“Enter”**.

- **Iniciar e terminar uma sessão de** 
- Para verificar se está na pasta pretendida fazer
`> getwd()`
- Para terminar uma sessão executar `>q()`.
Se pretender guardar o *workspace* (sessão de trabalho que contém o conjunto de objectos de trabalho) fazer **Yes** e fica guardado no ficheiro **.Rdata**

- Todas as funções e comandos do  estão armazenados em (módulos) *packages*.
 - **Para:**
 - ver quais os packages disponíveis `>(.packages())`
 - ver quais os packages instalados `>library()`
 - carregar em memória um package instalado
`>library(nome-package)` ou
menu: Packages → Load Package ...
 - Para instalar um *package* fazer
menu: Packages → Install Package ...
 - Numa sessão de  o conteúdo de um *package* só fica disponível quando ele é carregado em memória.

Ajudas no

- Sobre um *package*
`>help(package=datasets)`
- Sobre um conjunto de dados
`>help(InsectSprays)` ou `> ?InsectSprays`
- Sobre uma função, conhecendo o nome
`>help(mean)` ou `> ?mean`
- Para pesquisar uma sequência de caracteres
`>help.search("norm")` ou `>?"norm"`
Indica o *package* e comando onde aparece a sequência
stats::Normal The Normal Distribution

O R usado como calculadora:

- **Expressões aritméticas** (aqui o resultado é mostrado e não guardado):

```
> 2 + 3/4 * 7^2
```

```
[1] 38.75
```

```
> exp(-2)/log(sqrt(2))
```

```
[1] 0.3904951
```

```
> sin(pi)^2 + cos(pi)^2
```

```
[1] 1
```

```
> sin((pi)^2) + cos((pi)^2)
```

```
[1] -1.332987 # Note a diferença!!!!
```



- **Atribuição** (resultado guardado num objecto):


```
> x <- -3 # O resultado é guardado na variável
```

```
> x # para mostrar o conteúdo de x
```

Exemplo de funções usuais no :

- > `sqrt()` - raiz quadrada
- > `abs()` - valor absoluto
- > `log()` - logaritmo de base e
- > `log10()` - logaritmo de base 10
- > `exp()` - exponencial
- > `sin()` ; > `cos()` ; > `tg()`
- > `factorial(n)` - factorial, $n!$
- > `choose(n,k)` - devolve $\binom{n}{k}$



- Os objectos do  são entidades que o  cria, manipula e podem ser guardados num *workspace*.
- Para ver a lista dos objectos no *workspace*: `> ls()`
- Para ver a informação sobre os objectos no *workspace*:
`> ls.str()`
- Para apagar objectos: `> rm(x, y)`
- Para apagar **todos** os objectos existentes no *workspace*
`> rm(list=ls())`
- Para guardar o *workspace* num ficheiro: `> save.image()` ou
menu: `File`→`Save Workspace ...`
- O ficheiro *workspace* por omissão é `.RData`

Em vez de escrever os comandos directamente na consola do  podem ser escritos e guardados em **ficheiros de texto** já sem erros e até comentados, para facilitar a sua utilização posterior. Estes ficheiros devem ter extensão **.R** e devem ser guardados na pasta de trabalho.

Para:

- Criar um ficheiro de *script*
menu: `File` → `New script ...`
- Utilizar um ficheiro de *script*
menu: `File` → `Open script ...`

Estrutura e manipulação de dados

- Os **objectos** no  são caracterizados por:
 - nome;
 - tipo - ex. **vector**, **matrix**, **factor**, **array**, **data frame**, **ts**, **list**, **function**;
 - atributos:
 - *mode*: numeric, character, complex, logical;
 - *length*: número de elementos no objecto;
- >str(x)** - mostra a estrutura interna do objecto **x**
- **Nome**
 - Deve começar com uma letra (A-Z ou a-z);
 - Pode conter dígitos e/ou pontos;
 - **Case-sensitive** (maiúsculas são diferentes de minúsculas).
 - **Nomes a evitar** (porque são usados internamente pelo ). Alguns:
c. q, t, C, D, F, I, T, pi, diff, df, pt, if, else, for, in, next, repeat, else, while, break, NULL, NA, NaN, Inf, FALSE, TRUE

Vector: estrutura de dados do mesmo tipo (numérico ou caracteres), armazenados enumerados – é o tipo de objecto mais comum.

- Criação de um **vector** - o uso de `c()`

```
> x <- c(1.2, 5.7, 6.3, 8, 14)
```

```
> cores <- c("Red","Green","Blue")
```

```
> u <- c(F,T,F)
```

```
> mais.cores <- c(cores, "Yellow","Black")
```

- Um vector pode conter símbolos especiais: **NA** (valor desconhecido, *missing value*), **NaN** (*Not a Number*), **Inf**, **- Inf**.

```
z <- c(log(0),NA,Inf);z
```

```
[1] -Inf NA Inf
```


- Geração de **sequências** (permite criar certos vectores)

```
> y <- 1:5
> w <- seq(1, 1.4, by = 0.1)
> w1 <- rep(1,7)
> w2 <- rep(1:3,2)
> v <- gl(2,3,labels=c("não","sim"));v
[1] não não não sim sim sim
Levels: não sim
```

- **Operações com vectores**

```
> v1 <- c(1,3,-1,2); v2 <- c(2,4,5,1)
```

Nota: operações realizadas elemento a elemento – se um dos vectores tiver menor dimensão que o outro é concatenado consigo próprio

```
> v1+v2; v1*v2; v1*2; 2/v1
```

- **Operadores lógicos**

> `x>4; x>4 & x<6` (`&` conjunção)

> `x<5 | x >= 8` (`|` disjunção)

> `2==sqrt(4)` [1] TRUE

- **Seleção de elementos de um vector - usa-se []**

> `cores[1]` - devolve a 1ª componente do vector `cores`

> `cores[-c(1,3)]` - mostra o vector resultante da remoção dos elementos na posição 1 e 3 do vector `cores`

> `x[u]` - devolve as componentes de `x` correspondentes às componentes `TRUE` de `u`

> `x[x>2 & x<14]` - devolve as componentes de `x` entre 2 e 14

Algumas funções realizadas elemento a elemento

- > `length(x)` - devolve o numero de elementos do vector x
- > `sort(x)` - devolve um vector com os elementos do vector x ordenados por ordem crescente
- > `sum(x)` - devolve a soma dos elementos do vector vector x
- > `prod(x)` - devolve o produto dos elementos do vector x
- > `cumsum(x)` - devolve um vector cujos elementos são a soma acumulada dos elementos do vector x
- > `cumprod(x)` - procedimento análogo ao anterior, com o produto
- > `max(x); min(x)` - devolve máximo e mínimo dos elementos do vector vector x
- > `factorial(x)` - devolve, para cada componente x_i , $\Gamma(x_i + 1)$
- > `sample(x)` - faz uma permutação dos elementos do vector x

Objectos no R - Matrix

Uma **matriz** é uma estrutura de dados, do mesmo tipo, referenciados por dois índices (a duas dimensões). Define-se pelo número de linhas **nrow** e número de colunas **ncol** e um conjunto de **nrow** × **ncol** valores.

```
>M <- matrix(1:12,nrow=3,ncol=4);M
>rownames(M)<-c("L1","L2","L3")
>colnames(M)<-c("C1","C2","C3","C4")
```

M

| | [,1] | [,2] | [,3] | [,4] | | C1 | C2 | C3 | C4 |
|------|------|------|------|------|----|----|----|----|----|
| [1,] | 1 | 4 | 7 | 10 | L1 | 1 | 4 | 7 | 10 |
| [2,] | 2 | 5 | 8 | 11 | L2 | 2 | 5 | 8 | 11 |
| [3,] | 3 | 6 | 9 | 12 | L3 | 3 | 6 | 9 | 12 |

Os valores são dispostos por coluna, a menos que seja indicado

```
> M <- matrix(1:12,3,4,byrow=T)
```

Pode transformar-se matrizes em vectores e reciprocamente

```
> A <- c(3,2,-4,0,-1,8)      # A é um vector
> dim(A) <- c(2,3)          # transforma A numa matriz 3 x 2
> A
```

```
      [,1] [,2] [,3]
[1,]    3   -4   -1
[2,]    2    0    8
```

```
> v<- as.vector(A);v      # transforma A num vector
```

Matriz criada por concatenação de colunas ou de linhas

```
> x<- 4:7 ; y<-5:8
> u <- cbind(x,y); v <- rbind(x,y)
```

Seleção de elementos de uma matriz

```
> M[3,1]          # elemento que está na linha 3 e coluna 1
> M[3,]          # vector com os elementos da linha 3
> M[,1]          # vector com os elementos da coluna 1
> M[c(1,4),]     # elementos da linha 1 e linha 4
                  # de todas as colunas
> M[-2,c(1,4)]   # submatriz constituída pelas
                  # linhas 1 e 3 e colunas 1 e 4
```

Operações com matrizes

```
> A * M      # faz o produto elemento a elemento
              # (verifique o que dá!!!)
> A %*% M    # faz o produto usual de matrizes
> t(A)       # transposta de A
> diag(k)    # faz a matriz identidade de dimensão k
> rowMeans(A) # devolve o vector das médias por linha
> solve(A)   # inversa de A
> solve(A,b) # devolve o vector x na equação Ax=b

> vec <- rep(2,4)
> M * vec    # por replicação tem o vector (2,2,2,2)
              # e faz produto componente a componente
> M %*% vec  # faz o produto usual (matemático)
              # da matriz M pelo vector (2,2,2,2)
```

Um **factor** é um objecto do **tipo vector** cujas componentes são um conjunto finito de categorias. Um factor tem associado a si um conjunto de níveis, que são os valores possíveis que pode tomar.

```
> sexo <- c("M","H","H","M","H") #alfanuméricos
```

```
> str(sexo)
chr [1:5] "M" "H" "H" "M" "H"
```

```
> sexo <- factor(sexo) # transforma em factor
```

```
> str(sexo)          # os niveis tomam valores numericos
Factor w/ 2 levels "H","M": 2 1 1 2 1
```

```
> levels(sexo) <- c("Masculino", "Feminino")
# altera a designacao dos niveis
```

```
> str(sexo)
```


Funções usadas por um Factor

```
> table(sexo)
```

```
sexo
```

```
Masculino  Feminino
           3         2
```

```
>idade<-factor(c("adulto","jovem","jovem","adulto","adulto"))
```

```
>tabela<-table(sexo,idade)
```

```
>margin.table(tabela,1)
```

```
           idade
sexo      adulto jovem
Masculino     1     2
Feminino     2     0
```

```
           sexo
Masculino  Feminino
           3         2
```

- **List**: colecção ordenada de objectos (componentes da lista) que podem ser de tipos diferentes (vectores numéricos, vectores lógicos, matrizes, funções,...)

```
> aluno <- list(num=12345, nome="Manuel",  
+ notas=c(12.5,13.4,12.1,14.3), curso="Eng.Florestal")  
> str(aluno) #dá designação e mode
```

- As componentes de uma lista podem ser referidas pelo seu índice, com `[[]]` ou pela sua designação

```
> aluno[[2]]           # segunda componente  
> aluno$nome          # componente designada "nome"  
> aluno[2:3]          # resulta uma sublista
```


Nota: O resultado de muitas funções é uma lista.

Objectos no R – Data Frame

Um **data frame** é semelhante a uma matriz em que as colunas podem conter dados de diferentes tipos.

Um **data frame** pode ser visto como **uma tabela de dados**: colunas – são as variáveis; linhas – são os registos (as observações em cada variável)

É a estrutura usual para armazenar tabelas de dados

Leitura de um **data frame** já existente no 

```
> data() # mostra vários data frame existentes
        # no package "datasets"
> data(ToothGrowth) # carrega os dados para
                   # a memória do R
> ToothGrowth      # mostra os valores contidos
                   # no data frame
```

```
> str(ToothGrowth)
      # mostra a estrutura do data frame
> head(ToothGrowth)
      # para visualizar as 6 primeiras linhas
> names(ToothGrowth)
      # dá os nomes das variáveis (colunas)
> dim(ToothGrowth)
      # dá a dimensão (n.linhas, n. colunas)
> ToothGrowth[,2]
      # dá a 2ª coluna
> ToothGrowth$len
      # mostra os valores da variável "len"
```

Consultar um `data frame` pode ser mais simples utilizando a função `attach()`.

Permite aceder directamente às *colunas de um `data frame`*, sem necessidade de referir o nome do data frame.


```
> len                # objecto desconhecido
> attach(ToothGrowth)
> len # permite aceder às colunas do data frame
> detach(ToothGrowth) # operação inversa de attach
> len                # objecto de novo desconhecido
```

Como criar um Data Frame

```
> pauta <- data.frame(N.Aluno = c(18355, 17456, 19334, 17756),  
+   turma = c("T1", "T2", "T3", "T3"),  
+   notas.Est = c(10.3,9.3, 14.2, 15))
```

```
> pauta; pauta$notas.Est
```

| | N.Aluno | turma | notas.Est | |
|-----|---------|-------|-----------|------|
| 1 | 18355 | T1 | 10.3 | |
| 2 | 17456 | T2 | 9.3 | |
| 3 | 19334 | T3 | 14.2 | |
| 4 | 17756 | T3 | 15.0 | |
| ... | | | | |
| [1] | 10.3 | 9.3 | 14.2 | 15.0 |

Uma das formas mais comuns de armazenar dados para trabalhar no  é usar **ficheiros de texto**.

Tendo um ficheiro no formato `txt` ou `dat` ou `csv` (*Comma Separated Values*), i.e., os valores em cada linha estão separados por vírgulas ou ponto e vírgula **deve**

- 1 abrir-se o ficheiro com um editor de texto (Notepad, Wordpad) para visualizar a estrutura
- 2 para ler usar o comando usar `read.table()`

```
>read.table("ficheiro",header=TRUE)
```

dependendo da estrutura dos dados.

Quando se tem dados em Excel **deve guardar-se cada folha num arquivo csv**. Dependendo das configurações do computador as colunas virão separadas por vírgula (,) ou ponto e vírgula (;)

Exemplo

```
>semente<-read.table("sementes.csv",header=TRUE,  
+ dec = ".",sep=";",as.is = TRUE,na.strings = "NA")  
>head(semente)
```

| | melhorada | tradicional |
|---|-----------|-------------|
| 1 | 3.46 | 3.18 |
| 2 | 3.48 | 3.67 |
| 3 | 2.74 | 2.92 |
| 4 | 2.83 | 3.10 |

Existem outras funções de leitura, semelhantes a `read.table()`, cujas diferenças residem no `separador` que é usado por omissão

`read.csv()` – separador decimal é `ponto`;

`read.csv2()` – separador decimal é `vírgula`

Para escrever o conteúdo de um `data frame`, "x", num ficheiro "output.csv", compatível com o Excel, usa-se a função

```
>write.table(x,file="output.csv",sep=";",  
+ dec=".",row.names=FALSE)
```

Para escrever num ficheiro `.txt` compatível com o *Notepad* fazer

```
>write.table(x,file="output.txt",sep=","  
+ ,row.names=FALSE)
```

Funções e Programação em

O  tem um vasto conjunto de funções já definidas, orientadas para o objecto

Estrutura:

>função(argumentos obrigatórios, argumentos opcionais)

- Exemplo de funções standard (já referimos algumas atrás)

abs() log() log10() sqrt() round(x,3)
exp() sin() cos() tan() gamma() choose(n,k)

- Funções de álgebra matricial

t(X) nrow(X) eigen(X) solve(A,b) det(X)

- Funções estatísticas

```
mean()  median()  quantile(x,prob=p)
var()   sd()      plot()  barplot()
summary()  sample() hist()  boxplot()
predict()  lm()    aov()   t.test()
```

Mais adiante veremos mais ...

Criação de uma função em R

- Uma função é definida por um **nome**, uma **lista de argumentos** separados por vírgulas e um **bloco de instruções** (corpo da função)

Expressão geral

```
>function(arguments) {  
  comandos  
}
```

Exemplo 1. O modelo logístico standard

```
>Flogistica<- function(x){1/(1+exp(-x))}  
      # função distribuição cumulativa  
>dlogistica<- function(x){exp(-x)/(1+exp(-x))^2}  
      # função densidade  
>Flogistica(0); dlogistica(0)
```

Exemplo 2. Cálculo do coeficiente de variação de um vector de dados

```
>coef.var<- function(x) {  
  cv<-sd(x)/mean(x)  
  return(cv)  
}  
  
>z<-c(2,4,6,2,4,8,9,1,3,2,7,8,3,2)  
>coef.var(z)  
  
[1] 0.621123
```

A função `for ()`. Sintaxe

```
> for (indice in sequencia) {  
  expressão a executar  
}
```

Exemplo

```
> x<-c(2,4,6,2,4,8,9,1,3,2,7,8,3,2)  
> soma<-0  
> for (i in 1:length(x))  
  {soma<-soma+x[i]}  
> soma
```


A estrutura `if()`

```
>conta<-0;xval<-rnorm(10);xval;soma<-0
>for (i in 1:10)
{
  >if (xval[i]<0) {conta<-conta+1}
    else
      {soma<-soma+xval[i]}
}
>conta;soma
>print(conta);print(soma)
```

A função `ifelse()` permite trabalhar com vectores (evitando ciclos)

```
> x<-c(5,20,70)
> ifelse(x<18,"menor","adulto")
> ifelse(idade<18,"jovem",ifelse(idade<65,"adulto",
"senior"))
```