Introdução a R

Rafael A. Rosales

27 de fevereiro de 2024

R é uma linguagem de programação com objetivos específicos e ambente para desenvolvimento de programas direcionados a estatística. R é iniciado geralmente desde uma linha de comando (terminal em linux/MAC OS) ou em uma GUI (por exemplo RStudio). No primeiro caso basta digitar o comando `R`.

Uma vez iniciado, o primeiro comando abaixo permite verificar em qual diretório estamos trabalhando e o segundo permite mudar de diretório

```
getwd()
setwd("C://.../meudiretorio")
```

Em linux/MAC o argumento para o segundo comando possue usualmente a forma `"/home/rrosales/aulas"`. Para sair do `R` basta digitar

```
q()
```

R é uma liguagem baseada em objetos. Todos os objetos em R têm um nome associado e podem armazenar diferentes tipos de coisas: números, strings, vetores, matrizes, gráficos, funções, bases de dados, etc. Para listar objetos armazenados em qualquer momento usamos

```
ls()
```

Para remover o objeto de nome `meu.vetor` escrevemos

```
rm(meu.vetor)
```

e para remover tudo

```
rm(list=ls())
```

Todos os objetos do workspace podem ser armazenados permanentemente em um arquivo com o comando

```
save.image("meusobjetos.Rdata")
```

`save.image()` pode ser chamado sem argumento; neste caso os objetos são salvos no arquivo `.Rdata` no diretório determinado por `getwd()`. Todas as instruções digitadas na sessão atual podem ser salvas a um aquivo de texto (ascii) escrevendo

```
savehistory("meuscomandos.R")
```

Por defeito `savehistory()` sem nenhum argumento salva no arquivo com nome `.Rhistory`. A combinação dos arquivos .Rdata e .R é poderosa e permite recriar o trabalho iniciado em outro momento! Para isto é suficiente digitar

```
load("meusobjetos.RData")
loadhistory("meuscomandos.R")
```

1 Ajuda

Informações relativas a uma função específica, por exemplo da função `plot` podem ser obtidas digitando

```
help(plot)
?plot
help.start(browser="C://...") # ajuda em formato html
help.search # ajuda de varias formas
??plot
??help.search # mais ajuda e exemplos
```

O site http:www.r-project.org, além de disponibilizar o próprio R, fornece diferentes manuais inclusive em português.

2 Dados

A função `read.csv()` permite importar dados de arquivos no formato .csv. O conteúdo do arquivo é armazenado em um objeto do tipo `data.frame`. Estes últimos podem ser enxergados como uma tabela para a qual existem um grande número de ferramentas que permitem a sua análise. A função `read.table` funciona da mesma maneira para arquivos de texto onde os dados estão separados em colunas. Por exemplo,

```
dados  <- read.table("/home/rrosales/aulas/Covid-RP.txt",
  header=TRUE)
attach(dados)
```

importam os dados do arquivo `Covid-RP.txt`. Estes dados foram tomados do site oficial da secretaria da saude https://susanalitico.saude.gov.br/#/dashboard/. O operador "`<-`" e o equivalente a "`=`" em outras linguagens. `read.table` possui vários argumentos, por exemplo `header` pode ser `TRUE` ou `FALSE`. Na primeira opção, a linha inicial do arquivo de dados é interpretada como cabeçalho podendo fornecer os nomes de cada uma das colunas, `FALSE` informa ao R que os dados se iniciam na primeira linha. `attach(dados)` permite trabalhar com cada coluna do arquivo pelo seu próprio nome. Isto último só funciona com `header=TRUE`. `read.table` também pode ser utilizada para importar dados da web, por exemplo

```
dados  <- read.table("http://dcm.ffclrp.usp.br/~rrosales/aulas/Covid-RP.txt",
   header=TRUE)
attach(dados)
```

head permite ver as primeira linhas do data frame dados,

```
head(dados)
   dia Nov
 1   1   0
 2   2   3
 3   3   0
 4   4   2
 5   5   1
 6   6   2
```

A função str(dados) mostra um resumo da estrutura do data.frame dados. A primeira coluna deste data frame pode ser escolhida como

```
coluna1 <- dados[1,]
```

ou

```
coluna1 <- dia
```

pois attach(dados) permite termos acesso as colunas pelo seu nome. Outras formas de acessar elementos de um data frame são

```
dados$dia                # coluna com nome `dados'
dados$Nov                # coluna com nome `Nov'
dados[["dia"]]           # coluna com nome `dados'
dados[["Nov"]]           # coluna com nome `Nov'
dados[2:3,]              # selecionar a 2a e 3a linha
dados[dados$Nov > 300,]  # seleciona linhas onde Nov > 300
dados[1:6,2]             # seleciona os valores das linas 1-6 da 2a coluna
```

3 Vetores e Matrizes

Um vetor é uma estrutura de dados que permite armazenar um conjunto de valores do mesmo tipo. Os seguintes exemplos mostram como criar um vetor; c abaixo é uma função para concatenar elementos e criar vetores

```
x <- 2
y <- c(1,2,3,4)
z <- c("a", "b", 3)
meu.vetor <- c()        # vetor vazio
meu.vetor[3] <- 721
```

Se digitamos o nome do vetor, R imprime o seu conteúdo, por exemplo

```
meu.vetor
 [1]  NA  NA 721
```

O seguinte modifica a primeira entrada do vetor `meu.vetor`

```
meu.vetor[1] <- 58
meu.vetor
 [1]  58  NA 721
```

Muitas operações aritméticas básicas e muitas outras são vetorizadas quando aplicadas em vetores, ou seja executadas componente a componente, por exemplo

```
x <- c(1,2,3)
y <- x
z <- x + y
z
 [1] 2 4 6
sqrt(z)
 [1] 1.414214 2.000000 2.449490
x/y
 [1] 1 1 1
```

Existem mutras outras formas de gerar um vetor. Por exemplo

```
v <- seq(0,1,0.01)
```

gera um vetor dos números do 0 até o 1 com acréscimo de 0.01 entre dois números consecutivos.

Uma matriz é geralmente criada com `matrix(value, nrow, ncol, byrow, dimnames)`, onde o primeiro argumento é um vetor com as entradas, `nrow` é o número de linhas, `ncol` é o número de colunas e `dimnames` é nome a cada coluna e linha da matriz (opcional). Por exemplo

```
M <- matrix(3,4,byrow=T)
M
      [,1]
 [1,]    3
 [2,]    3
 [3,]    3
 [4,]    3
A <- matrix(c(1,2,2,1),2,2,byrow=T)
A
      [,1] [,2]
 [1,]    1    2
 [2,]    2    1
B <- matrix(c(6,9,0,1),2,2,byrow=F,
       dimnames = list(c("l1","l2"), c("c1","c2")))
B
    c1 c2
```

```
 l1  6  0
 l2  9  1
```

Os elementos das matrizes podem ser acessados da várias maneiras, por exemplo,

```
A[1,1]    # elemento da fila 1, coluna 1
  [1] 1
A[1,]     # primeira fila
 [1] 1 2
A[,1]     # primeira coluna
 [1] 1 2
```

As operações soma, subtração e multiplicação de matrizes são realizados com os operadores `+`, `-` e `%*%` respectivamente; por exemplo

```
A%*%A
      [,1] [,2]
 [1,]    5    4
 [2,]    4    5
```

A multiplicação por um escalar é feita utilizando '`*`',

```
3*A
      [,1] [,2]
 [1,]    3    6
 [2,]    6    3
```

sendo a matriz transposta e o determinante obtidas respectivamente com as funções `t()` e `det()`,

```
t(B)
    l1 l2
 c1  6  9
 c2  0  1
det(B)
 [1] 6
```

4 Estruturas de controle

Condições `if` e `if-else`. A primeira destas tem a forma `if (<condicao>) {...}` e a segunda `if (<condicao>) {...} else {...}`; por exemplo

```
# generamos um numero aleatorio uniformemente no intervalo [0,10]
x <- runif(1, 0, 10)
if(x > 3) {
   y <- 10
} else {
   y <- 0
}
```

O valor de `y` é 10 se *x* é maior do que 3 e 0 no caso contrário. Caso o corpo do `if` ou do `else` possua mais de uma instrução, cada uma destas ultimas deve ser separada por ‘;’. Um exemplo de um loop `for` é o seguinte

```
x <- c("a", "b", "c", "d")
for(i in 1:4) {
   print(x[i])
}
[1] "a"
[1] "b"
[1] "c"
[1] "d"
```

Não é necessário utilizar uma variável do tipo índice no loop `for`, de fato

```
for(letter in x) {
   print(x[i])
}
[1] "a"
[1] "b"
[1] "c"
[1] "d"
```

O seguinte código implementa um loop `for` dentro de outro,

```
C <- matrix(1:6, 2, 3)
for(i in seq_len(nrow(C))) {
   for(j in seq_len(ncol(C))) {
      print(C[i, j])
   }
}
```

Além das estruturas `for` e `if-else`, R também possui `while`, `repeat` os quais podem ser utilizados com `next` e `break`. Sugiro procurar no sistema de ajuda para obter exemplos destes últimos casos.

5 funções

A sintaxe básica para definirmos uma função é `f <- function(...) { ... }`, por exemplo, a seguinte função transforma graus Celcius em Farenheit,

```
CemF <- function(cel) {
   far <- 9/5*cel + 32
   print(far)
}
```

Assim,

```
CemF(32.511)
 [1] 90.5198
```

A seguinte função calcula o fatorial do número x,

```
fatorial <- function(x){
   f <- 1
   for (i in 1:x) {f <- f*i}
   return(f)
}
```

(mesmo que no R isto possa ser calculado simplesmente como `x!`)

6 Estatística descritiva

Sugiro utilizar o sistema de ajuda do R para aprender a utilizar as funções[1]: `max`, `min`, `mean`, `median`, `quantile`, `var`, `sd`, `cov`, `cor`, `hist`, `pie`, e `boxplot`. Uma função bastante util para obter um resumo dos resultados de diversas análises é `summary`.

A teoria mínima necessária para entender o output destas funções pode ser encontrada no primeiro Capítulo do livro de Magalhães (EdUSP).

7 Gráficos

Existem muitas formas de criarmos gráficos em R. Tal vez a mais utilizada considera a função `plot`. A sintaxe mais simples é `plot(x, y)` onde x e y são dois vetores da mesma dimensão (`x` contém os valores a serem graficados nas abissas e `y` nas ordenadas). Por exemplo, para os dados carregados na Seção 2,

```
plot(dia, Nov, xlab="dia", ylab="número de infestados por dia")
```

Para salvarmos o gráfico em formato pdf, utilizando vários argumentos de `plot` para ajustar a forma em como o grafico é apresentado, podemos fazer

```
pdf("covid.pdf")
plot(dia, Nov, xlab="dia", ylab="número de infestados por dia",
     cex=1.3, pch=19, col="black", cex.axis=1.3, cex.lab=1.3)
dev.off()
```

O resultado disto ultimo é apresentado na Figura 1.

Existem outros formatos aos quais podemos exportar gráficos: png, tiff, eps, postscript, metafile (Windows), tiKz (LaTeX/TeX), ... As funções para algumas destas opções são : `png`, `tiff`, `cairo_ps`, `metafile`. O formato `tikz` requere carregar previamente `tikzDevice`.

[1] as vezes Google pode ser bem mais ilustrativo e fácil do que os manuais do R!

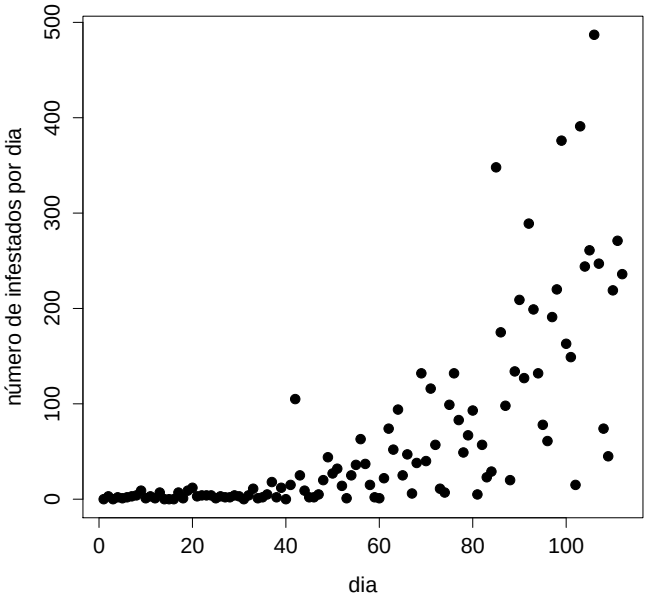
número de infestados por dia
0
100
200
300
400
500
0
20
40
60
80
100
dia

Figura 1: