Universidade Fernando Pessoa

Uma breve introdução ao R

Exploração prática e exercícios

Luis Borges Gouveia

28-12-2016


EXEMPLO
FUNÇÃO
DADOS
OBJETOS
VETOR
PODEMOS
COMANDO
COLUNAS
OBJETO
MATRIZES
PATTERN
NORMAL
USAR
GRÁFICOS
DIFERENTES
MEDIA
EXERCÍCIOS
NÚMERO
COLUNA
LINHAS
RESULTADO
DISTRIBUIÇÃO
SEQUÊNCIA
REP
CASO
VALOR
CONJUNTO
MATRIZ
SER
CADA
NÚMEROS
RESULTADOS
VALORES
GRÁFICO
FORMA
TREES
PLOT
SEQ
SEGUINTE
PODE
ARGUMENTO
OPERAÇÕES
OPERAÇÃO
POSSUI
RECURSO
NOME
APPLY
ESPAÇO
PRETENDER
VARIÂNCIA
RETORNA
ALTERNATIVA
EXISTEM
PERMITE
DISPONÍVEIS
ÚTEIS
SÉRIE
ÚTIL
MULTIPLICAÇÃO
PROGRAMAÇÃO
NESTE
CRIAR
QU
ÚLTIMO
DIM
MATRIX
BY
DESTE
TRUE
SEIS
OBTER
MEDIAN
EXISTE
VAR
PAR
TER
FUNCTION
MUITAS
PONTOS
LS
VEZ
HIST
NROW
HELP
SEMPRE
VEZES
TAL
IF
1:3
NHTEMP
MROW
DOIS
MEAN
OBTEMOS
VETORES
POS
ALTURA
POSSÍVEL
MAIOR
PODEM
MODO
TRABALHO
SUMMARY
DNORM
EFEITO
REALIZAR
UTILIZADA
PADRÃO
SESSÃO
CORRENTE
ASSIM
DENSIDADE
ARGUMENTOS
VARIÁVEIS
ELEMENTOS
LINGUAGEM
PODEMOS
OUTRO
CÁLCULO
VOLUME
DIRETÓRIO
INTERVALOS
DIMENSÕES
DATA
PUAS
SEGUINTES
OPÇÃO
ESTATÍSTICAS
CÓDIGO
HEIGHT
MÉDIA
SUBSCRIÇÃO
EXEMPLOS
SUMÁRIO
DESVIO
NAME
VERIFICAR
TAMANHO
USO
LENGTH
COMANDOS
ALL.NAMES
RESPOSTAS
BIBLIOTECAS
EDITOR

Uma breve introdução prática ao R
Versão 1.2 – Dezembro de 2016

CONTEÚDO

Uma breve introdução prática ao R
Versão 1.0 – Dezembro de 2016

1. INTRODUÇÃO

O **R** é uma linguagem de programação e também um *software* para análise de dados, que conseguem realizar análises estatísticas simples ou complexas. Inclui um conjunto de funções que permitem realizar análise de dados, resumos e exploração de dados, modelos de dados e apresentar os resultados em gráficos de qualidade profissional.

O objetivo deste texto é proporcionar uma introdução estruturada ao R, tanto mais que o seu potencial exige um investimento considerável inicial para se perceber o ambiente, a linguagem e a forma como o R trabalha, aliado ao necessário conhecimento de estatística e de como explorar e analisar os dados.

O texto está organizado para ser utilizado com o computador, para testar cada um dos exemplos, numa sessão R. Assim, os comandos apresentados devem ser introduzidos e verificada a resposta do R, deve ser entidida a operação. Para confirmar que tudo foi devidamente entendido, devem ser realizados os exercícios propostos.

Quando se está numa sessão de R, podemos criar objetos que são armazenados no espaço de trabalho corrente (correspondente à imagem digital da sessão). O espaço de trabalho corrente é definido no ambiente do R, através de funções de sistema, também embebidas na linguagem.

Para verificar qual o espaço corrente, podemos usar a função *getwd()*:

```
> getwd()
[1] "/Users/luisborgesgouveia"
```

Se se pretender mudar o diretório corrente para um diretório de trabalho diferente, é utilizada a *setwd()*.

```
> setwd("/Users/luisborgesgouveia/Documents")
```

Se verificarmos o diretório atual, agora obtemos:

```
> getwd()
[1] "/Users/luisborgesgouveia/Documents"
```

...o novo diretório

Se pretendermos listar o seu conteúdo, usamos *list.files()*

```
> list.files()
 [1] "cap5_incluir.docx"
 [2] "Filed Documents"
```

[3] "ihm_trabalho4.pptx"

[4] "images"

[5] "mcprojet.ppt"

[6] "Microsoft User Data"

[7] "OpenProj_27805.pptx"

[8] "teste.R"

Cada objeto criado permanece nessa imagem até que seja eliminado de forma explícita. Também neste caso, o **R** possui funções próprias para o efeito.

Com a função *ls()*, os objetos são listados

Por exemplo, criando dois objetos *x* e *X* (o **R** faz distinção entre maiúsculas e minúsculas):

> x <- c(2, 3, 5, 7)

> X <- c(10, 12, 14, 16)

> ls()

[1] "x" "X"

É possível eliminar um determinado objeto, com recurso à função *rm()*

Por exemplo, para eliminar o objeto *x*:

>rm(x)

O espaço de trabalho ficou agora apenas com o objeto *X*

> ls()

[1] "X"

No final da sessão, o espaço de trabalho será perdido se não for gravado. É possível gravar o espaço de trabalho a qualquer momento com a função *save.image()* ou recorrendo ao icon de gravação (disco) na consola do interface.

Os comandos escritos em R são guardados em memória durante uma sessão. Podemos sempre voltar a invocar os mesmos (com recurso às teclas do cursor) ou a visualizar com base no elevador da janela da consola. Em alternativa é também possivel copiar e colar comandos da história e dessa forma usar as diferentes facilidades de edição para não ter de repetir todos os comandos. Existe ainda a possibilidade de gravar a história (menu *File*, opção *Save History*), que permite graver uma cópia em ficheiro, da transcrição de todos os comandos utilizados e na sequência em que foram utilizados. Em alternativa é ainda possível copiar e colar os comandos num editor ou usar um ambiente que possua um editor integrado, como é o caso do *RStudio*.

Para fechar uma sessão em **R**, é utilizada a função *q()*.

Após invocar a função *q()* é solicitada a opção de se pretender ou não graver o espaço de trabalho corrente. Se não for gravado, o conteúdo da informação será perdido.

2. OBJETOS E ARITMÉTICA

O **R** armazena os dados e realiza operações com objetos. Os objetos mais simples são escalares, vetores e matrizes. Mas existem muitos outros: por exemplo, listas e data.frames. No contexto do uso avançado do **R** pode ser útil definer novos tipos de objetos, específicos para uma dada aplicação. Neste texto, apenas serão considerados os objetos em uso, mais comuns.

Uma funcionalidade importante do **R** é o de realizer coisas diferentes a objetos diferentes. Por exemplo, se for solicitada uma soma:

```
> 12 + 8
```

O resultado obtido é:

```
[1] 20
```

Desta forma, o **R** realiza aritmética escalar retornando o valor escalar 20 (na prática, o **R** retorna um vetor de tamanho 1, em que [1] denota o primeiro elemento do vetor.

Podemos atribuir valores a objetos, para uso posterior (o mesmo que variáveis). Por exemplo:

```
> x <- 12
> y <- 8
> z <- x + y
```

Realiza a mesma operação que a soma anterior (com o mesmo resultado, 20) armazenando o resultado no objeto *z*. Podemos observer os conteúdos de um objeto, escrevendo o seu nome:

```
> z
[1] 20
```

A qualquer momento, podemos listar os objetos que foram criados e que existem na imagem da sessão:

```
> ls()
[1] "x" "y" "z"
```

Note que *ls* é também, ele próprio, um objeto. Se escrevermos *ls*, obtemos listados os conteúdos desse objeto, no caso, uma função:

```
> ls

function (name, pos = -1L, envir = as.environment(pos), all.names = FALSE, pattern, sorted=TRUE) {
  if (!missing(name)) {
    pos <- tryCatch(name, error = function(e) e)
    if (inherits(pos, "error")) {
      name <- substitute(name)
      if (!is.character(name))
        name <- deparse(name)
      warning(gettextf("%s converted to character string",
        sQuote(name)), domain = NA)
      pos <- name
    }
  }
  all.names <- .Internal(ls(envir, all.names, sorted))
  if (!missing(pattern)) {
    if ((ll <- length(grep("[", pattern, fixed = TRUE))) &&
      ll != length(grep("]", pattern, fixed = TRUE))) {
      if (pattern == "[") {
        pattern <- "\\["
        warning("replaced regular expression pattern '[' by  '\\\\['")
      }
      else if (length(grep("[^\\\\]\\[<-", pattern))) {
        pattern <- sub("\\[<-", "\\\\\\[<-", pattern)
        warning("replaced '[<-' by '\\\\[<-' in regular expression pattern")
      }
    }
    grep(pattern, all.names, value = TRUE)
  }
  else all.names
}
<bytecode: 0x106010110>
<environment: namespace:base>
```

De momento, não nos devemos preocupar com entender o código desta função. Quando utilizamos os parenthesis em *ls()*, asseguramos que a função é executada (e não mostrado o seu conteúdo/código). Neste caso, *ls()* lista os objetos da sessão.

Em geral, podemos afirmar que uma função opera num objeto. Por exemplo:

```
> sqrt(2500)
[1] 50
```

Calcula a raiz quadrada de 2500, Os objetos podem ser removidos do espaço de trabalho corrente, com recurso à função rm:

```
> rm(x,y)
```

Remove os objetos *x* e *y*.

Existem muitas funções disponiveis no **R** e é posível criar novas funções. Por exemplo, podemos criar uma função em que dados três números, retorna o maior entre estes (eu sei, é apenas um exemplo...). Assim, criamos uma função de nome *maiorde3*, com 3 argumentos e com a lógica de escolha do maior valor entre estes e retorna esse valor (se forem números repetidos, não faz diferença, retorna sempre o maior).

```
> maiorde3 <- function (x, y, z) {
+   maior <- x
+   if (maior < y) maior <- y
+   if (maior < z) maior <- z
+   return(maior)
+ }
> maiorde3(12, 23, 22)
[1] 23
```

Com números repetidos...

```
> maiorde3(12, 12, 1)
[1] 12
```

Em **R**, os vetores podem ser criados de diversas formas. Os seus elementos podem ser descritos, da seguinte forma:

```
> z <- c(15, 2,  5,  4)
```

Notar que o uso da função *c* para concatenar ou relacionar os elementos individuais. Esta função pode ser utilizada de forma mais alargada, por exemplo:

```
> x <- c(15, 2)
> y <- c(5, 4)
> z <- c(x, y)
```

Leva ao mesmo resultado que o commando anterior a esta sequência de três comandos, criando um vetor com os números 15, 2, 5 e 4. Se ptretendermos gerar uma sequência de números, podemos recorrer ao uso de gamas, tal como o exemplo:

```
> x <- 1:20
```

Que gera uma sequência de números entre 1 e 20 (1, 2, 3, 4, …, 17, 18, 19, 20). Em alternativa ao uso

de gamas, podemos recorrer à função *seq*, para gerar sequências. Por exemplo:

```
> seq(1, 20, by=4)

[1]  1  5  9 13 17
```

Que gera uma sequência iniciada no valor 1 e até 20, com passos de 4. Logo o último valor é 17, pois o seguinte seria 21, maior que 20. Um outro exemplo do uso da função *seq*:

```
> seq(6,32,length=7)

[1]  6.00000 10.33333 14.66667 19.00000 23.33333 27.66667 32.00000
```

O que acontece neste caso? Foi criada uma sequência de 6 a 32, com sete intervalos de igual valor. Podemos verificar a situação, se colocarmos a sequência num vetor e fizermos a diferença entre cada um dos intervalos, isto é, entre cada um dos valores da sequência

```
> nums <- seq(6,32,length=7)
> for (i in 2:7) {
+   print(nums[i-1]-nums[i])
+ }
[1] -4.333333
[1] -4.333333
[1] -4.333333
[1] -4.333333
[1] -4.333333
[1] -4.333333
```

Ok, as diferenças entre os seis pares dos sete números são iguais e com o valor de 4.33333…

Estes exemplos ilustram as funções existentes em **R** e os seus muitos argumentos alternativos, que podem mudar de forma efetiva os resultados obtidos. Por exemplo, no caos da função *seq*, temos o argumento *by* ou o argumento *length*. Se não se especificar argumentos, o **R** toma as funções com os valores de defeito para essa função que, no caso da função *seq*, é assumir um tamanho de passo de valor 1. Assim, por exemplo:

```
>  x <- seq(1,10)
```

Gera um vetor de inteiros de 1 a 10 (1, 2, 3, 4, 5, 6, 7, 8, 9 e 10).

Agora, vale a pena mencionar a facilidade de ajuda do **R**, Se não se conhecer como usar uma dada função ou não se conhecer quais são as suas opções (argumentos) e os seus valores por defeito, então podemos usar o comando *help* ou o ponto de interrogação (*?*). É normalmente uma ajuda que descreve a função e possui normalmente exemplos para demonstração.

Por exemplo, se se pretender ajuda para saber mais sobre a função *seq*:

```
> help(seq)
```

ou

```
> ?seq
```

Uma função útil para construir vetores de valores é o comando *rep*, para repetir valores. Por exemplo:

```
> rep(0, 100)
[1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[38] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[75] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Que repete o valor zero, cem vezes, ou:

```
> rep(1:3, 6)
[1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
```

Em que a sequência 1, 2 e 3 é repetida seis vezes.

A função pode ser usada de forma mais complexa. Por exemplo:

```
> rep(1:3, c(6,6,6))
[1] 1 1 1 1 1 1 2 2 2 2 2 2 3 3 3 3 3 3
```

A sequência gerada é de 6 valores 1, seguindos por outros seis valores 2 e mais seis valores 3. Por exemplo, com o seguinte comando podemos obter diferentes repetições para cada um dos valores da gama:

```
> rep(1:3, c(6,4,2))
[1] 1 1 1 1 1 1 2 2 2 2 3 3
```

Neste caso, o número 1 é repetido seis vezes, o dois, quatro vezes e o três, duas vezes. Se tomarmos novamente o comando *rep(1:3, c(6,6,6))*, podemos obter o mesmo resultado, da seguinte forma:

```
> rep(1:3,rep(6,3))
[1] 1 1 1 1 1 1 2 2 2 2 2 2 3 3 3 3 3 3
```

Como já apresentado, o **R** tem em conta os objetos nas suas operações e funções. Por exemplo:

```
> x <- c(2, 4, 5)
> y <- c(3, 5, 7)
> x + y
[1] 5 9 12
```

e

```
> x * y
[1] 6 20 35
```

Tal demosntra que o **R** utiliza aritmética de vetores, por reconhecer como vetores, os objetos x e y. O **R** também tenta lidar com objetos de deferens tipos ou misturados, tais como:

```
> x <- c(2, 4, 6)
> x + 2
[1] 4 6 8
```

Um cuidado especial deve ser tomado de forma a assegurar que o **R** faz o pretendido em cada circunstância (de alguma forma, basta testar e observer se os resultados foram os esperados).

Duas funções em particular são úteis, pelo que vale a pena ter a sua funcionalidade presente: (1) *length*, que retorna o tamanho de um vetor (isto é, o número de elementos que o vetor contém) e (2) *sum*, que calcula o somatório dos valores dos elementos do vetor. Um exemplo:

```
> x <- c(2, 4, 5)
> sum(x)
[1] 11
> length(x)
[1] 3
```

2.1 EXERCÍCIOS

1. Defina

```
> x <- c(4, 2, 6)
> y <- c(1, 0, -1)
```

2. Qual o resultado dos comandos seguintes em **R**? Utilize o **R** para confirmar posteriormente as respostas.

   (a) length(x)
   (b) sum(x)
   (c) sum(x^2)
   (d) x+y
   (e) x*y
   (f) x-2
   (g) x^2

3. Descreva as seguintes sequências e utilize o **R** para verificar as respostas:

   (a) 7:11
   (b) seq(2, 9)
   (c) seq(4, 10, by=2)
   (d) seq(3, 30, length=10)
   (e) seq(6, -4, by=-2)

4. Descreva os resultados obtidos com as seguintes expressões em **R** e a seguir, use o **R** para verificar o que está correto:

   (a) rep(2,4)
   (b) rep(c(1,2),4)
   (c) rep(c(1,2),c(4,4))
   (d) rep(1:4,4)
   (e) rep(1:4,rep(3,4))

5. Utilize a função *rep* para definir os seguintes vetores em **R**.

   (a) 6,6,6,6,6,6
   (b) 5,8,5,8,5,8,5,8
   (c) 5,5,5,5,8,8,8,8

3. ELABORAR SUMÁRIOS DOS DADOS E SUBSCRIÇÃO

Considere a recolha de um conjunto de dados sobre o tempo que demorou a resolver um problema com recurso em **R** (em minutos), por um conjunto de alunos. Os dados foram armazenados num objeto x:

```
> x <- c(7.5,8.2,3.1,5.6,8.2,9.3,6.5,7.0,9.3,1.2,14.5,6.2)
```

Quantos alunos foram considerados?

```
> length(x)

[1] 12
```

É possível obter algumas estatísticas dos dados (estatísticas descritivas); podemos assim obter a media, com a função *mean()* e a variância, com a função *var():*

```
> mean(x)
[1] 7.216667
> var(x)
[1]    11.00879
```

Adicionalmente, a função *summary()* faz um resumo dos dados, apresentando o mínimo e o máximo, a média, a mediana e o primeiro e terceiro quartil. A execução da função para o exemplo dados:

```
> summary(x)

   Min. 1st Qu.  Median    Mean  3rd Qu.     Max.

  1.200   6.050   7.250   7.217   8.475  14.500
```

Se se considerar que os 12 valores representam dois grupos de observação, para problemas diferentes, podemos considerar a sua análise separada. Assim, cada grupo possui o mesmo número de observações, seis. Deste modo, cada um dos grupos pode ser sumarizado de forma independente de modo a se poder analisar melhor os resultados. Para o efeito, devemos extrair do vetor x, os dois subvetores de seis elementos cada. Tal pode ser realizado por subscrição, com explicitação das games de indices, no vetor, conforme exemplificado a seguir:

```
> x[1:6]

[1] 7.5 8.2 3.1 5.6 8.2 9.3
```

e

```
> x[7:12]

[1] 6.5 7.0 9.3 1.2 14.5 6.2
```

A subscrição permite obter os dados de forma separada. Deste modo, podemos aplicar a função *summary()* aplicada a cada um dos dois subvetores:

```
> summary(x[7:12])

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.

  3.100   6.075   7.850   6.983   8.200   9.300

> summary(x[7:12])

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.

  1.200   6.275   6.750   7.450   8.725  14.500
```

Outros subconjuntos podem ser criados, com recurso às funções já apresentadas. Por exemplo:

```
> x[c(2,4,9)]

[1] 8.2 5.6 9.3
```

Neste caso, foram considerados três elementos do vetor: o Segundo, o quarto e o nono. Por outro lado, o uso de números negativos permitem excluir elementos particulares. Por exemplo:

```
> x[-(1:6)]

[1]  6.5  7.0  9.3  1.2 14.5  6.2
```

Esta estratégia de subscrição é equivalente ao uso de:

```
> x[7:12]

[1]  6.5  7.0  9.3  1.2 14.5  6.2
```

3.1 EXERCÍCIOS

1. Considerando o objeto *x <- c(5,9,2,3,4,6,7,0,8,12,2,9)*, indique qual(ais) o(s) element(s) que cada um dos comandos **R** representa e a seguir , use o **R** para confirmar as respostas:

   (a) x[2]

   (b) x[2:4]

   (c) x[c(2,3,6)]

   (d) x[c(1:5,10:12)]

   (e) x[-(10:12)]

2. Tomando os dados *y <- c(33,44,29,16,25,45,33,19,54,22,21,49,11,24,56)* contendo as vendas de cerveja em barris, para os dias úteis de uma semana, em três lojas diferentes (os primeiros três valores são para as lojas 1, 2 e 3 na segunda-feira e assim sucessivamente). Produza um sumário estatístico para as vendas em cada dia da semana e também por cada loja.

4. MATRIZES

As matrizes podem ser criadas no **R** de diversas formas. A mais simples é pela criação de colunas que são posteriormente associadas com o comando *cbind*. Por exemplo:

```
> x <- c(5,7,9)
> y <- c(6,3,4)
> z <- cbind(x,y)
> z
     x  y
[1,] 5  6
[2,] 7  3
[3,] 9  4
```

As dimensões da matriz podem ser verificadas com o comando *dim*:

```
> dim(z)
[1] 3 2
```

O resultado descreve três linhas e duas colunas. Existe um comando semelhante, *rbind*, para construir matrizes pela associação de linhas. As funções *cbind* e *rbind* também podem ser aplicadas às próprias matrizes (desde que as suas dimensões sejam compatíveis), para formar matrizes maiores. Por exemplo:

```
> rbind(z,z)
     [,1] [,2]
[1,]    5    6
[2,]    7    3
[3,]    9    4
[4,]    5    6
[5,]    7    3
[6,]    9    4
```

As matrizes podem ser construídas com recurso à função *matrix*, que permite a sua definição explícita e direta. Por exemplo, o comando:

```
z <- matrix(c(5,7,9,6,3,4), nrow=3)
```

Resulta numa matriz de 3 linhas por duas colunas. Deve ser notado que o tamanho da matriz é determinado pelo tamanho pelo tamanho do vetor (seis elementos) e pelo requisito que o número de linhas deve ser 3, como especificado pelo argumento *nrow=3*.

A matriz *z* organiza os seus elementos da seguinte forma:

```
> z
     [,1] [,2]
[1,]    5    6
[2,]    7    3
[3,]    9    4
```

Uma forma alternativa para obter a mesma matriz era especificar o número de colunas como 2, com o argumento *ncol=2*. A matriz é preenchida com os valores pela ocupação sucessiva de cada coluna, todas as linhas. Se se pretender que seja primeiro populadas cada linha para todas as colunas, existe uma opção *byrow=T*. Por exemplo:

```
> z <- matrix(c(5,7,9,6,3,4), nr=3, byrow=T)
> z
     [,1] [,2]
[1,]    5    7
```

```
[2,]    9    6
[3,]    3    4
```

Note que o argumento *nrow* for abreviado para *nr*. Estas abreviações são sempre possíveis para os argumentos das funções, desde que não causem ambiguidade pelo que, em caso de dúvida, usar sempre a notação completa do argumento.

Como usualmente, o **R** tenta interpretar as operações em matrizes de um modo natural. Por examplo, com a matriz *z*, já definica e a matriz *y*, podemos fazer diversas operações:

```
> y <- matrix(c(1,3,0,9,5,-1), nrow=3, byrow=T)
> y
      [,1] [,2]
[1,]    1    3
[2,]    0    9
[3,]    5   -1
```

Se somarmos as matrizes *y* + *z*, obtemos:

```
> y + z
      [,1] [,2]
[1,]    6   10
[2,]    9   15
[3,]    8    3
```

Se multiplicarmos as matrizes *y* e *z*, obtemos:

```
> y * z
      [,1]    [,2]
[1,]    5      21
[2,]    0      54
[3,]   15      -4
```

Uma questão importante! A multiplicação realizada foi orientada aos components e não a multiplicação convencional de uma multiplicação de matrizes. De facto, a multiplicação convencional é indefinida no contexto de

y e *z*, porque as dimensões das matrizes não são compatíveis para realizar a multiplicação. Se definirmos uma matriz *x*, com as dimensões compatíveis, podemos considerar a mutliplicação de matrizes.

```
> x <- matrix(c(3, 4, -2, 6), nrow=2, byrow=T)
> x
     [,1] [,2]
[1,]    3    4
[2,]   -2    6
```

Para realizar a multiplicação matricial, o **R** impõe uma notação própria; *%*%*. Desta forma se pretendermos mutiplicar as matrizes *y* por *x*, que são compatíveis nas suas dimensões:

```
> dim(x)
[1] 2 2
> dim(y)
[1] 3 2
```

O número de colunas de *y* é igual ao número de linhas de *x*, pelo que podemos realizar a multiplicação matricial:

```
> y %*% x
     [,1] [,2]
[1,]   -3   22
[2,]  -18   54
[3,]   17   14
```

Outras funções úteis para matrizes são *t*, para cálculo da matriz transposta e *solve*, para o cálculo da matriz inversa.

Transpor a matriz *z*:

```
> t(z)
     [,1] [,2] [,3]
[1,]    5    9    3
[2,]    7    6    4
```

e, calcular a matriz inversa de *x*:

```
> solve(x)
            [,1]       [,2]
[1,]  0.23076923 -0.1538462
[2,]  0.07692308  0.1153846
```

Tal como no caso dos vetores, pode ser útil extrair subcomponentes das matrizes. Neste caso, pode ser pretendido retirar elementos em particular, linhas ou colunas, Tal como anteriormente referido, a subscrição com recurso a indices e a parentesis retos, torna o processo possível. Os exemplos seguintes mostram algumas das possibilidades de uso:

Considerando z:

```
> z
     [,1] [,2]
[1,]    5    7
[2,]    9    6
[3,]    3    4

> z[1,1]
[1] 5

>  z[c(2,3),2]
[1] 6 4

> z[,2]
[1] 7 6 4

> z[1:2,]
     [,1] [,2]
[1,]    5    7
[2,]    9    6
```

Deste modo, é necessário especificar quais as linhas e as colunas. Sempre que se omitir o número inteiro para qualquer das dimensões, implica que todos os elementos desta dimensão sejam selecionados.

4.1 EXERCÍCIOS

1. Criar em **R** as matrizes

$$x = \begin{bmatrix} 3 & 2 \\ -1 & 1 \end{bmatrix}$$

e

$$y = \begin{bmatrix} 1 & 4 & 0 \\ 0 & 1 & -1 \end{bmatrix}$$

Calcular as seguintes expresses e verificar as respostas no **R**:

(a) 2 * x

(b) x * x

(c) x %*% x

(d) x %*% y

(e) t(y)

(f) solve(x)

2. Tomando as matrizes *x* e *y* já especificadas em 1., calcule o efeito das seguintes operações de subscrição e verifique as respostas no **R**.

(a) x[1,]

(b) x[2,]

(c) x[, 2]

(d) y[1, 2]

(e) y[, 2:3]

5. ANEXAR (*ATTACHING*) OBJETOS

O **R** possui diveross conjuntos de dados que são úteis para explorar em exemplos. É possível listar os conjuntos de dados disponíveis, com uma sua descirção, com recurso à função *data()*:

```
> data()
```

Para aceder a qualquer um dos conjuntos de dados, basta invocar a função com o nome correspondente do conjunto de dados pretendido. Por exemplo:

```
> data(trees)
```

Para listar as primeiras cinco linhas, para todas as caraterísticas do conjunto de dados, usamos:

```
> trees[1:5, ]
  Girth Height Volume
1   8.3     70   10.3
2   8.6     65   10.3
3   8.8     63   10.2
4  10.5     72   16.4
5  10.7     81   18.8
```

As colunas representam as medidas de circunferência (*gith*), altura (*height*) e volume das árvores (cerejeiras - *cherry trees*). Para saber mais sobre o conjunto de dados, digitar o comando **R**: *help(trees)*.

Para saber as dimensão do conjunto de dados, podemos recorrer às dimensões da matriz:

```
> dim(trees)
[1] 31  3
```

Logo, temos 3 colunas, com as caraterísticas, registadas para 31 cerejeiras

Assim, se se pretender trabalhar com as colunas destes dados, podemos usar a técnica de subscrição. Por exemplo, *trees[,2]* fornece todas as alturas. No entanto, o **R** possui uma forma alternative para poder referir as colunas nestes casos em que representam caraterísticas ou dimensões de um conjunto de dados. Essa forma alternativa é referida como anexar (*attaching*) ao conjunto de dados. Para o caso do conjunto de dados *trees*, o comando **R** é o seguinte:

```
> attach(trees)
```

Este comando torna os conteúdos do objeto trees, um diretório e se escrever o nome de um objeto, o **R** irá verificar no interior do diretório para descobrir esse nome. Uma vez que a altura (*height*) é o nome de uma das colunas de *trees*, o **R** reconhece este objeto quando o se escreve o seu nome. Assim, o exemplo:

```
> mean(Height) [1] 76
```

Produz o mesmo resultado que:

```
> mean(trees[ ,2])
[1] 76
```

Uma vez que são sinónimos. No entanto, o código fica bem mais fácil de se entender, quando se usa a primeira opção. Para o **R**, *trees* é um objeto designado por *dataframe*; essencialmente uma matriz com colunas com nome (embora uma *dataframe*, ao contrário de uma matriz, possa incluir valores – variáveis – não numéricos, como nomes). Assim, por causa disto, existe uma forma equivalente para a sintaxe de extrair dados do objeto, como por exemplo, o vetor das alturas:

```
> trees$Height
[1] 70 65 63 72 81 83 66 75 80 75 79 76 76 69 75 74 85 86 71 64 78 80 74 72 77 81
[27] 82 80 80 80 87
```

Que permite obter os dados da altura, das 31 cerejeiras, que pode ser utilizado sem ter sido anexado previamente ao conjunto de dados.

5.1 EXERCÍCIOS

1. Anexar o conjunto de dados *quakes* e produzir um sumário estatístico das variáveis *depth* e *mag*.
2. Anexar o conjunto de dados *mtcars* e descobrir a média do peso (*weight*) e a media do consumo de combustível (*fuel consumption*) para veículos no conjunto de dados. Use o comando *help(mtcars)* para uma descrição das variáveis disponíveis.

6. A FUNÇÃO APPLY

É possível escrever ciclos no **R**, mas devem ser evitados sempre que possível, pois tem custos computacionais e aumentam a complexidade dos programas. Uma situação comum é quando se pretende aplicar a mesma função a cada linha ou coluna de uma matriz. Por exemplo, podemos querer calcular o valor médio de cada uma das variáveis (colunas) do conjunto de dados *trees*. De um modo óbvio, podemos tomar cada coluna de forma separada mas tal pode ser repetitiva, especialmente quando existem muitas colunas. A função *apply* simplifica a operação, sendo fácil de entender com um exemplo:

```
> apply(trees, 2, mean)
    Girth    Height    Volume
13.24839  76.00000  30.17097
```

Produz o efeito de calcular a media de cada coluna (dimensão 2) do conjunto de dados *trees*. Se tivesse sido utilizado o valor 1 em vez de 2, como segundo argumento, era calculadada a media de cada linha.

Qualquer função pode ser aplicada desta forma, mesmo quando existam argumentos que tenham de ser fornecidos. Ver *help(apply)* para mais detalhes.

6.1 EXERCÍCIOS

1. Repetir a análise do conjunto de dados *quakes* e *mtcars* utilizando a função *apply* para simplificar os cálculos.

2. Se

$$y = \begin{bmatrix} 1 & 4 & 1 \\ 0 & 2 & -1 \end{bmatrix}$$

qual é o resultado de usar o comando **R** *apply(y[,2:3],1,mean)*? Verificar a resposta no **R**.

7. COMPUTAÇÃO ESTATÍSTICA E SIMULAÇÃO

Muitas das computações estatísticas mais fastidiosas são as que envolvem simulacrao ou o cálculo de probabilidades e a consulta de tabelas: estas operações podem ser facilmente realizadas no **R**. Entre outros, é útil para descobrir os intervalos de confiança. Tomemos o exemplo da distribuição normal. Existem funções em **R** para avaliar a função de densidade (de probabilidade), função de distribuição acumulada e a função de quartis (a função do inverso da distribuição).

Estas funções são respetivamente, *dnorm*, *pnorm* e *qnorm* (sendo que *norm*, designa a distribuição normal). Ao contrário das tabelas, não existe necessidade de normalizer as variáveis primeiro, Por exemplo, supor $X \sim N(3,22)$ – uma variável *X*, de distribuição normal, com média 3 e variância de 22, pode ser escrita em **R** como:

```
> x
     [,1] [,2]
[1,]    3    4
[2,]   -2    6
> dnorm(x,3,sqrt(22))
                [,1]          [,2]
[1,] 0.08505478 0.08314352
[2,] 0.04818818 0.06932115
```

O comando **R** permite assim calcular a função de densidade nos pontos contidos no vetor *x*. A função *dnorm* assume a media igual a zero e um desvio padrão de uma unidade, por defeito. No exemplo, a media é 3 e o desvio padrão é 4.690416 (resultado da raiz quadrada da variância, que é 22). A função pede o desvio padrão e não a variância, o que deve ser tomado em consideração para a colação dos valores dos argumentos. Por exemplo:

```
> dnorm(5,3,2)
[1] 0.1209854
```

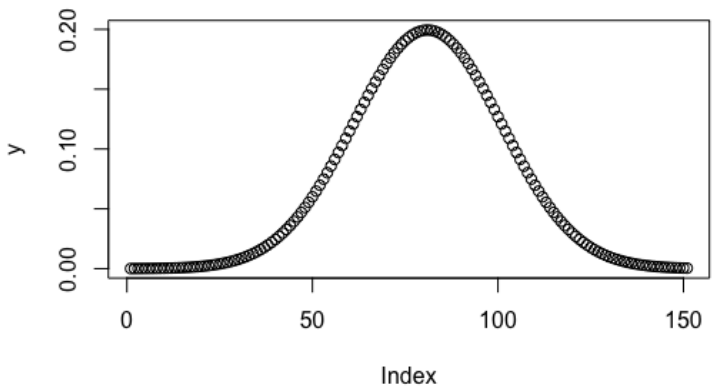
Avalia a densidade de probabilidade de uma distribuição normal com média 3 e variância 4 – *N(3,4)*; o valor de 2 é o desvio padrão: com estes parâmetros é calculada a densidade de probabilidade para o valor 5. Um outro exemplo:

```
> x <- seq(-5,10, by=.1)
> dnorm(x,3,2)
```

Calcula a função de densidade da mesma distribuição com intervalos de 0.1 no intervalo *[-5,10]*, implicando o cálculo para 151 valores. Se se criar um novo objeto *y*, com os valores gerados e se desenhar o gráfico correspondente, vai ser obtida a curva normal para o valor médio 3 e variância 4. Os comandos **R** para o efeito são os seguintes:

```
> x <- seq(-5,10, by=.1)
> y <- dnorm(x, 3, 2)
> plot(y)
```

O resultado é o seguinte gráfico:



As funções *pnorm* e *qnorm* funcionam de forma idêntica. Propõe-se o uso da função *help* para ser obtida mais informação e explorar as funções em causa.

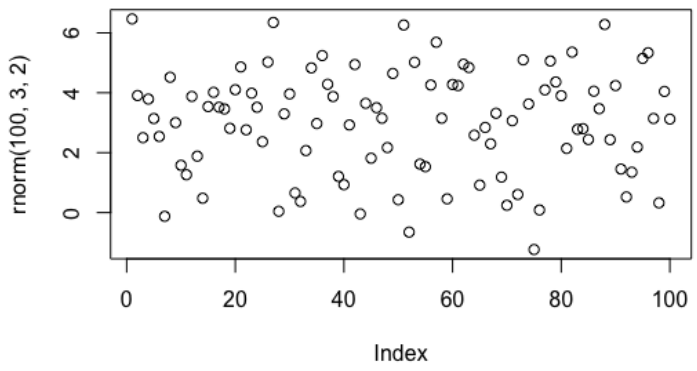
Estas funções estão associadas com a distribuição normal. No **R**, existem funções semelhantes, para outras distribuições. Por exemplo, *dt*, *pt* e *qt* para a distribuição t de *Student*, embora neste caso, os argumentos sejam os graus de liberdade em vez da média e desvio padrão. Outras distribuições também estão disponíveis, incluindo a Binomial, a Exponencial, a de *Poisson* e a Gama. É necessário conhecimento de estatística para explorar as funções e interpretar os seus resultados.

Uma técnica adicional importante para muitas aplicações estatísticas é a simulação de dados de uma distribuição de probabilidade específica. O **R** possibilita a simulação de um leque alargado de distribuições, utilizando uma sintaxe semelhante à utilizada nas funções de densidade. Por exemplo, para simular 100 observações de uma distribuição normal de media 3 e variância 4, *N(3,4)*, basta o seguinte comando **R**:

> rnorm(100, 3, 2)

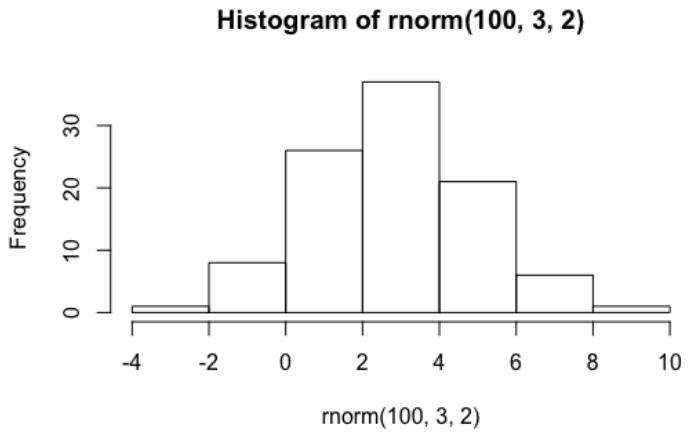
São assim gerados 100 valores de distribuição normal, com media 3 e desvio padrão 2, que por serem aleatórios tem valores, conforme mostrado no seguinte gráfico:

> plot(rnorm(100,3,2))



Na sucessão de valores, não é visivel a sua distribuição normal, mas se contarmos as ocorrências de cada um destes valores por intervalos, verificamos que o número de repetições de valores se aproximam de uma curva normal. Podemos verificar isso mesmo, através do gráfico:

> hist(rnorm(100,3,2))



Se fizermos a verificação com a sobreposição de uma curva normal com os mesmos parâmetros da geração de 100 números aleatórios, podemos obter os seguintes resultados:

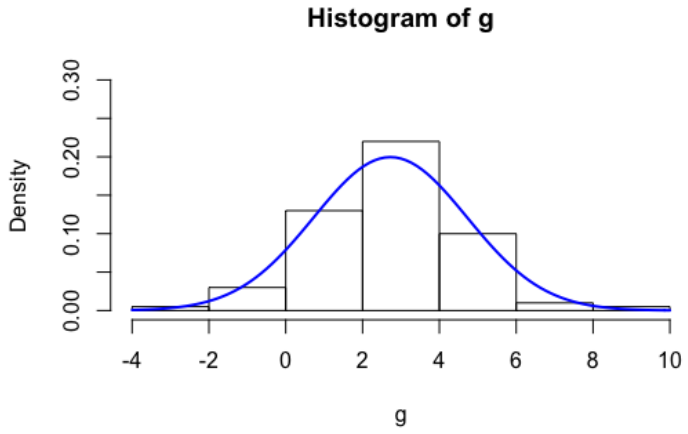
> g = rnorm(100,3,2)

> m <- mean(g)

> std <- sqrt(var(g))

> hist(g, prob=TRUE, ylim=c(0,0.3))

```
> curve(dnorm(x, mean=m, sd=std), col="blue", lwd=2, add=TRUE)
```



De igual modo, por exemplo, a função *rt* (t de *Student*) e *rpois* (*Poisson*) permitem a simulação com recurso a estas distribuições.

7.1 EXERCÍCIOS

1. Suponha $X \sim N(2, 0.25)$. Tome f e F como as funções de densidade e de distribuição de X, respetivamente. Utilize o **R** para o seu cálculo

   (a) $f(0.5)$

   (b) $F(2.5)$

   (c) $F^{-1}(0.95)$ (observação: F^{-1} é a função quartil)

   (d) $\Pr(1 \leq X \leq 3)$

2. Repetir a questão 1, mas para o caso de X possuir uma distribuição *t-Student* com 5 graus de liberdade.

3. Utilizar a função *rpois* para simular 100 valores de uma distribuição de *Poisson* com um parâmetro à sua escolha. Produzir um sumário estatístico do resultado e verificar se a média e a variância se encontram alinhadas de forma razoável com os valores expectáveis para este tipo de distribuição.

4. Repetir a questão anterior, substituindo a função *rpois* pela função *rexp*.

8. GRÁFICOS

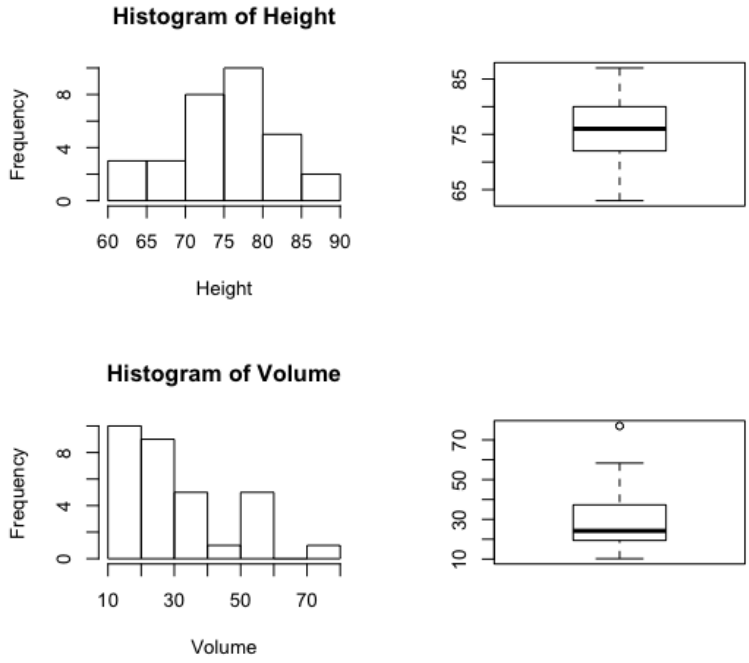
O **R** possui imensas funcionalidades associadas com a produção de gráficos de alta qualidade. Alguns pequenos exemplos foram já apresentados. Uma funcionalidade útil antes de começar a produzir os gráficos, é dividir a página em partes menores de forma a permitir que mais do que uma figura seja desenhada. Por exemplo:

```
> par(mfrow=c(2,2))
```

Este comando cria uma janela de gráficos com 2 linhas e duas colunas. Com esta escolha, os gráficos são colocados sucessivamente por linha, Se se usar mfcol, os gráficos serão colocados sucessivamente por coluna. A função par é uma função gerral para atribuição dos parâmetros gráficos. Existem muitas opções possíveis; ver *help(par)*.

Tomemos um exemplo, baseado nos dados do conjunto de dados *trees*:

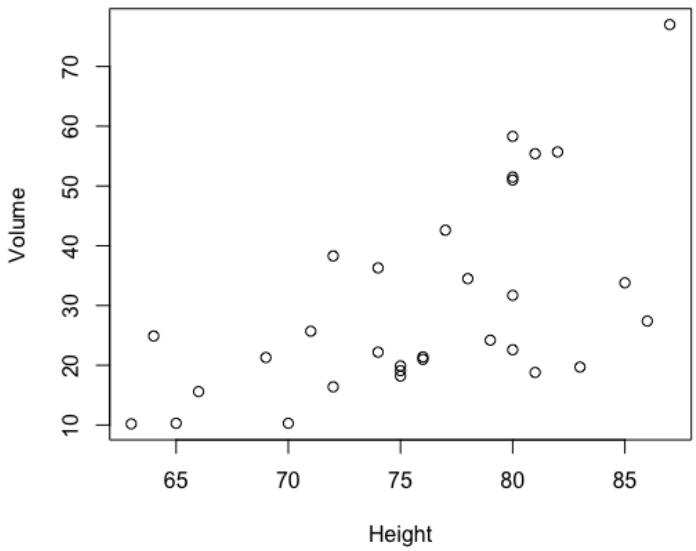
```
> par(mfrow=c(2,2))
> hist(Height)
> boxplot(Height)
> hist(Volume)
> boxplot(Volume)
> par(mfrow=c(1,1))
```



A figura criada apresenta as alturas e os volumes das árvores, para o conjunto de dados *trees*. Notar que o último comando, com a função *par* se destina a repor a janela dos gráficos no tamanho normal (um só gráfico).

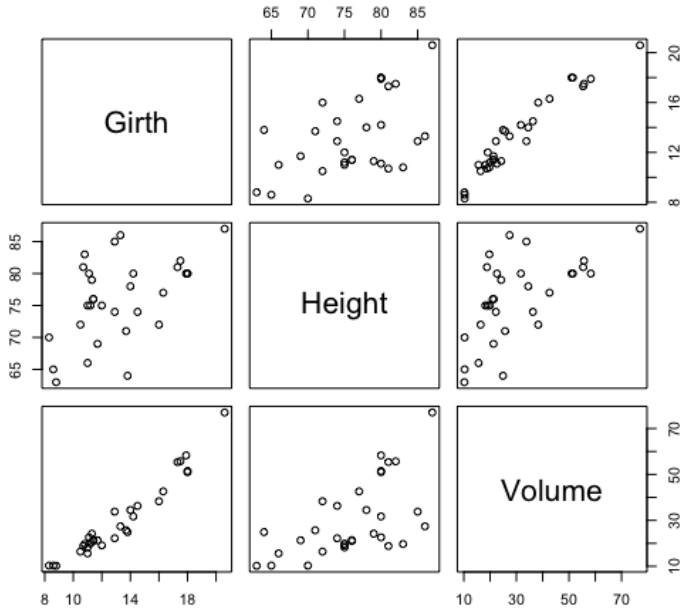
Um outro tipo de gráfico é o obtido com a função *plot*. Neste caso, obtemos um gráfico com base na informação e mapeamento gráfico dos valores de *x* e *y*. Por exemplo, mapear os valores de altura e volume para cada uma das 31 árvores (cerejeiras):

```
> plot(Height, Volume)
```



O **R** pode também produzir um diagrama de dispersão (*scatterplot*) para a matriz do conjunto de dados *trees*. Uma matriz de diagramas de dispersão apresenta um gráfico para cada par de variáveis, usando a função *pairs*. No conjunto de dados *trees*, temos três variáveis: circunferência, altura e volume, pelo que existe uma matriz de três por três, logo com 9 posições a que se desconta o número de variáveis, resultando seis pares de comparação das variáveis:

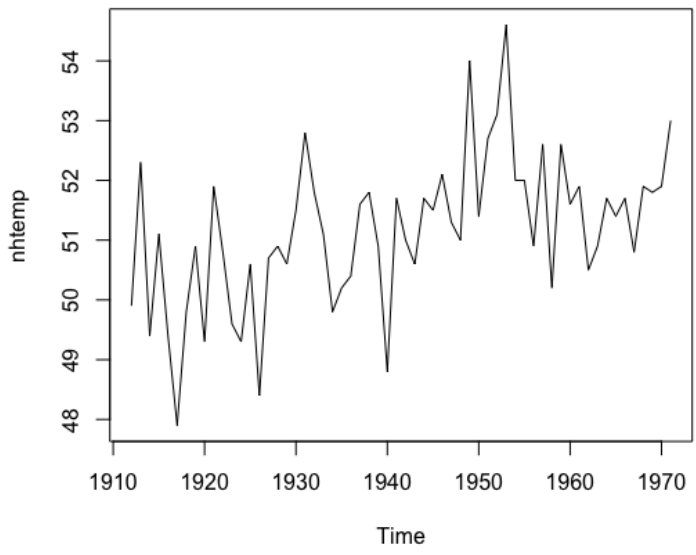
```
> pairs(trees)
```



Tal como em muitas outras funções, a função *plot* tem em conta os objetos que lhe servem de argumento e o seu comportamento depende do objeto em que é aplicada. Por exemplo, se o objeto é uma matriz, a função *plot* é idêntica a *pairs*: *plot(trees)* produz o mesmo resultado (gráfico) que *pairs(trees)*.

Outros exemplos, também com recurso a conjuntos de dados fornecidos com o **R**:

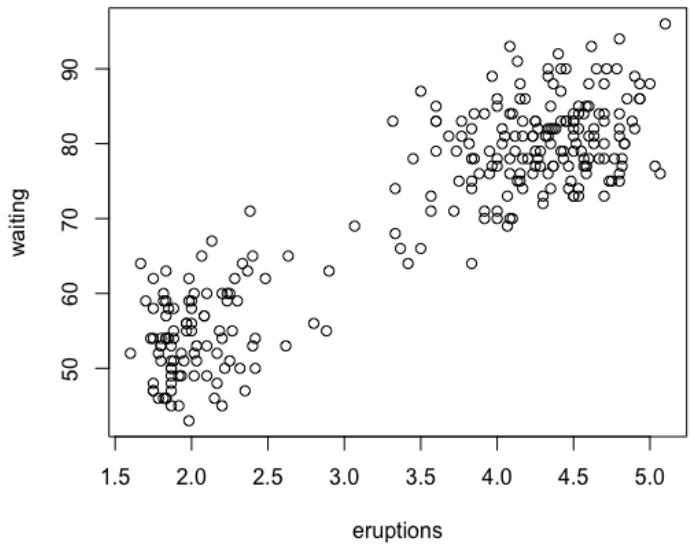
```
> data(nhtemp)
> plot(nhtemp)
```



O conjunto de dados nhtemp compila as temperaturas médias anuais em New Haven, série temporal de 1912 a 1971. Um outro exemplo:

> data(faithful)

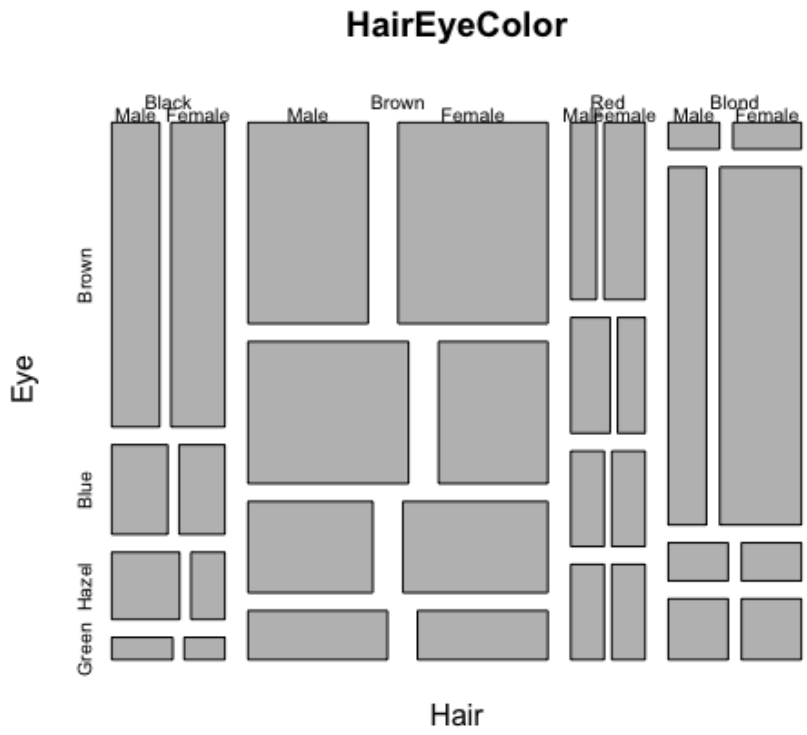
> plot(faithful)



Neste caso, o conjunto de dados *faithful* apresenta 272 observações da erupção e do intervalo de espera de um Geyser. Ainda um outro exemplo:

> data(HairEyeColor)

> plot(HairEyeColor)



O conjunto de dados *HairEyeColor* é um conjunto estatístico de dois quadros de valores de estudantes (um por género), que compara a cor dosseus cabelos e dos seus olhos:

```
> HairEyeColor
, , Sex = Male

        Eye
Hair    Brown Blue Hazel Green
  Black    32   11    10     3
  Brown    53   50    25    15
  Red      10   10     7     7
  Blond     3   30     5     8

, , Sex = Female

        Eye
Hair    Brown Blue Hazel Green
  Black    36    9     5     2
```

Brown	66	34	29	14
Red	16	7	7	7
Blond	4	64	5	8

Existem muitos argumentos opcionais para a maior parte das funções gráficas, que podem ser utilizadas para controlar cores, legendas, simbolos de dados, eixos, títulos, etc. As funções points and lines são úteis para adicionar pontos e linhas, respetivamente, ao gráfico corrente. A função abline é útil para adicionar uma linha com um ponto de interceção e um declive específico.

Para imprimir um gráfico, selecionar a janela do gráfico e usar o botão da direita do rato para escolher a opção de impressão, de gravação ou de cópia, de modo a reutilizar a imagem obtida.

8.1 EXERCÍCIOS

1. Utilize

```
> x <- rnorm(100)
```

Para gerar dados aleatórios. Produza uma figura que mostre um histrograma e um diagram de caixas (*boxplot*) com os dados gerados. Modifique os nomes dos eixos e o título do gráfico, de modo a realizar a sua personalização.

2. Execute os seguintes comandos **R**:

```
> x <- (-10):10
> n <- length(x)
> y <- rnorm(n, x, 4)
> plot(x, y)
> abline(0, 1)
```

Tente entender o efeito de cada comando no gráfico produzido.

Modifique o código da seguinte forma a verifique quais as alterações produzidas ao gráfico

```
x <- (-10):10
n <- length(x)
y <- rnorm(n, x, 4)
plot(x, y, pch=16, cex=1.4, col = "blue")
abline(lm(y~x))
```

3. Execute o seguinte código em **R**:

```
> data(nhtemp)
> plot(nhtemp)
```

É produzido um gráfico de uma série temporal com as medidas das temperaturas médias anuais para New Hampshire, nos Estados Unidos.

4. O exemplo anterior ilustra que a função *plot* atua de forma diferente, em função dos objetos serem de diferentes tipos. O objeto *nhtemp* possui uma classe especial de uma série de tempo. Mais genericamente, podemos ter os dados das observações anuais num vetor, mas precisamos de construer o gráfico da série de tempo. A operação pode ser realizada com o comando em **R**:

```
> temp <- as.vector(nhtemp)
```

Que cria um vetor temp que contém as temperaturas anuais. Uma forma de produzir um gráfico semelhante à série de tempo é com o comando em **R**:

```
> plot(1912:1971, temp)
```

No entanto, este gráfico coloca pontos em vez de linhas. Para unir os dados com linhas, podemos utilizar:

```
> plot(1912:1971, temp, type='l')
```

O argumento *type*, também permite a colocação simultânea de pontos e linhas, com *type='b'*.

9. ESCREVER FUNÇÕES

Uma importante funcionalidade do **R** é a facilidade de extender os comandos da linguagem de programação, escrevendo novas funções. Por exemplo, embora exista uma função *var*, que calcula a variância de um conjunto de dados num vetor, não existe uma função para o cálculo do desvio padrão. Podemos definir uma função para o efeito:

```
> sd <- function(x) sqrt(var(x))
```

Assim, por exemplo, com base nos seguintes valores:

```
> r <- c(3, 5, 6, 8, 9, 12, 3, 8)
> sd(r)
```

```
[1] 3.105295
```

Se se pretender modificar a função, o comando *fix(sd)* irá abrir a função num editor que se pode usar para a modificar. Também se poderia usar este modo, para definir a função. Se a função não existe ainda, o efeito é abrir o editor com a estrutura base de uma função:

```
function () {
}
```

Esta estrutura pode ser utilizada para inserir o corpo da função. O editor torna mais fácil inserir e corrigir o código da função. Note que as chavetas iniciam e finalizam o código, possibilitando a possibilidade de várias linhas de comandos serem incluídas no código da função. Por exemplo, podemos criar uma função escrevendo

```
> fix(several.plots)
```

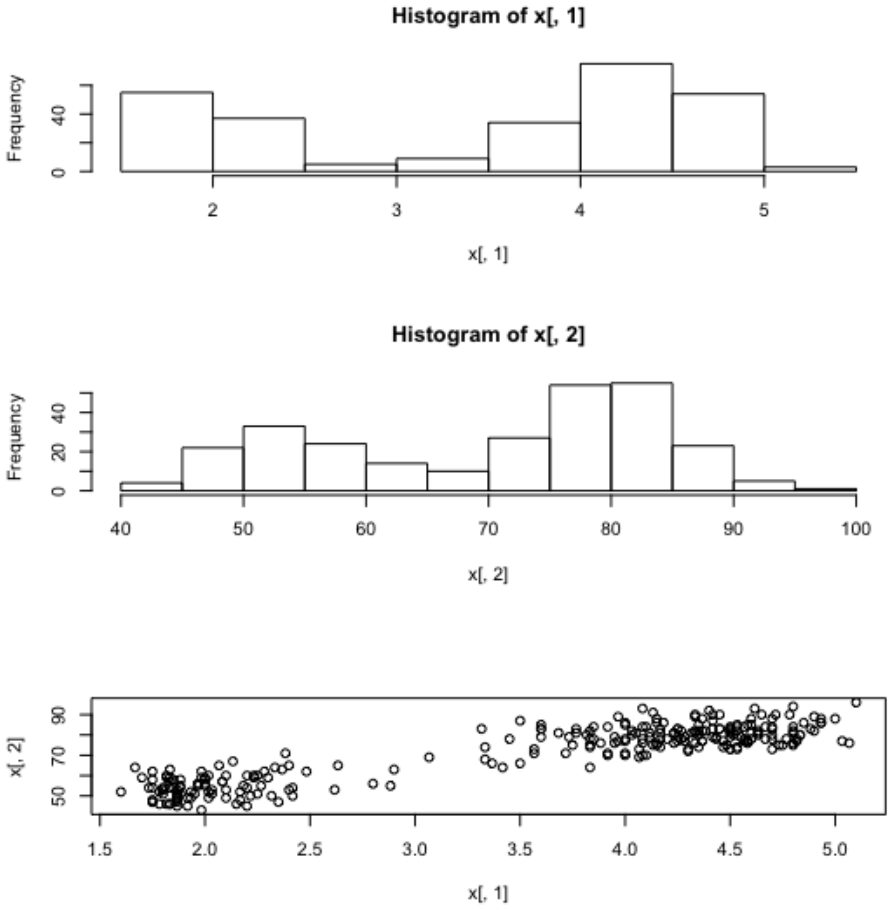
E usar o editor para definir a seguinte função:

```
multi.graf <- function(x) {
    par(mfrow = c(3, 1))
    hist(x[, 1]) hist(x[, 2])
    plot(x[,1], x[,2])
    par(mfrow = c(1, 1))
    apply(x, 2, summary) }
```

O resultado da função está relatado na linha final. As linhas anteriores são tomadas como intermediárias, embora as saídas gráficas também são enviadas para o ecran, se forem incluídas. Esta função função em particular apenas fará sentido para objetos *x* que sejam matrizes com duas colunas – produz um histrograma de cada coluna e um gráfico de pontos de uma coluna com outra. Por último, é retornado um sumário, por cada coluna do objeto de dados.

Deste modo, a função *multi.graf()* produz o seguinte resultado:

```
> multi.graf (faithful)
```



	eruptions	waiting
Min.	1.600	43.0
1st Qu.	2.163	58.0
Median	4.000	76.0
Mean	3.488	70.9
3rd Qu.	4.454	82.0
Max.	5.100	96.0

9.1 EXERCÍCIOS

1. Escreva uma função em **R** que tome como argumentos dois vetors *x* e *y* e produz um gráfico do tipo *scatterplot* e calcula o ceficiente de correlação (utilizando a função *cor(x,y)*).

2. Escreva uma função em **R** que tome um vetor *(x1, x2, … ,xn)* e calcule a soma do valor de *xi* com a posição *i*). Por exemplo, um vetor com três valores: *(2, 4, 1)*, fica com o cálculo exemplificado de *(1+2, 2+4, 3+1)* e com o resultado final de *(3, 6, 4)*.

3. Escreva uma função em **R** para criar um vetor que fornece os n primeiros números de Fibonacci. Os números de Fibonacci são gerados em sequência, sendo os seus dois primeiros valores, a unidade e os seguintes, a soma dos dois números anteriores. A sequência de números segue a sequência: *1, 1, 2, 3, 5, 8, 13, 21, …*

4. Escreva uma função em **R** para listar os fatores que dividem um dado número com resto zero. A função deve no final, indicar o número de fatores que esse número possui.

5. Escreva uma função em **R** que solicite um número inteiro do utilizador e verifique se se trata de um número primo.

10. USAR TABELAS ESTATÍSTICAS E DESENHO DE DISTRIBUIÇÕES DE PROBABILIDADE

O **R** proporciona uma forma simples para usar as tabelas estatísticas e para o cálculo de probabilidades com recurso às distribuições mais comuns (contínuas ou discretas). Permite ainda o desenho das curvas associadas com essas distribuições, de forma rápida e simples. A seguir são partilhados um conjunto de comandos **R** que permitem demonstrar este potencial. Chama-se à atenção, mais uma vez, que o **R** é sensível ao uso de minúsculas e maiúsculas, pelo que *tmp*, *Tmp* e *tmP*, constituem três objetos em **R** diferentes.

A sintaxe geral para as funções de distribuição em **R** possui funções próprias e tomas elas tem associados quatro tipos de comandos. Assim, Existem quatro comandos básicos em **R** que se aplicam a diversas funções de distribuição, pela inclusão de uma letra inicial que define esse comando, a acrescentar à função de distribuição. Considerando a função de distribuição *fdist* e os *parametros* que especificam a distribuição, a sintaxe genérica é a seguinte:

- Iniciada por **d** – *dfdist(x, parametros)*: função de densidade de probabilidade de *fdist*, calculada em *x*.
- Iniciada por **q** – *qfdist(p, parametros)*: retorna o valor *x* que satisfaz a probabilidade para a função de distribuição *fdist(parametros)* ≤ *x)* = *p*
- Iniciada por **p** – *pfdist(x, parametros)*: retorna a probabilidade da função de distribuição *fdist(parametros)* ≤ *x)*
- Iniciada por **r** – *rfdist(n, parametros)*: gera um número *n* de variáveis aleatórias da função de distribuição *fdist(parametros)*

As funções contínuas de probabilidade mais comuns estão também disponíveis em **R**. No caso de determinados parâmetros não sejam especificados, são tomados os seus valores por defeito. Por exemplo, no caso da distribuição normal, será a curva normal padrão, caso a média e a variância não forem especificadas.

Alguns exemplos de distribuições de probabilidade contínuas:

Distribuição	Sintaxe
Normal	*norm()*: por defeito, a curva unitária normal; *norm(μ,σ²)*, distribuição normal com média μ e variância σ²
t de Student	*t(df)*: por defeito, a distribuição central com *df* graus de liberdade; *t(df,ncp)*, distribuição não central *t* com parâmetro não central *ncp*
Qui quadrado χ2	*chisq(df)*: por defeito, distribuiçao central χ2 com *df* graus de liberdade; *chisq(df,ncp)*, distribuição não central χ2 com parâmetro não central *ncp*

F	*f(df1,df2)*: por defeito distribuição central F com *df1* e *df2* graus de Liberdade; *f(df1,df2,ncp)*, distribuição não central F com parâmetro não central *ncp*

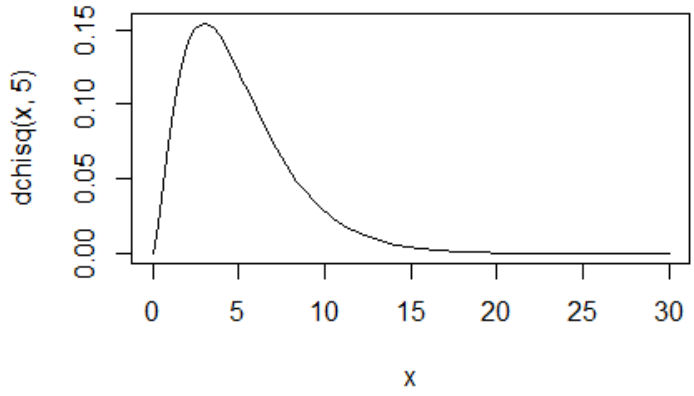
Por sua vez, a representação gráfica das funções de distribuição de probabilidade é um dos aspetos mais diferenciadores do **R**. De fato, o **R** possui uma excelente capacidade de gerar gráficos com elevada qualidade com base nas suas funções. Um dos comandos para a criação destes gráficos é a função *curve*:

- *curve(função, xlow, xhigh)*: gera um gráfico da função, entre o valor *xlow* e *xhigh*
- *curve(função, xlow, xhigh, n=500)*: gera um gráfico da função, com recurso a 500 pontos distribuídos de forma uniforme, entre *xlow* e *xhigh*

Tomemos um exemplo de uma função qui-quadrado com 5 graus de liberdade, entre os pontos 0 e 30. O comando **R** para o fazer, com a função *curve*, é:

```
> curve( dchisq(x, 5), 0, 30)
```

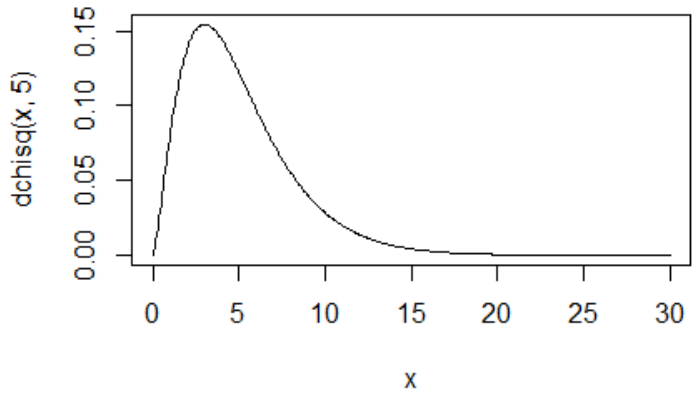
E o resultado é:



Se o comando tiver ainda o argumento de pontos de espaçamento, obtemos:

```
> curve( dchisq(x, 5), 0, 30, n = 500)
```

E o resultado é:



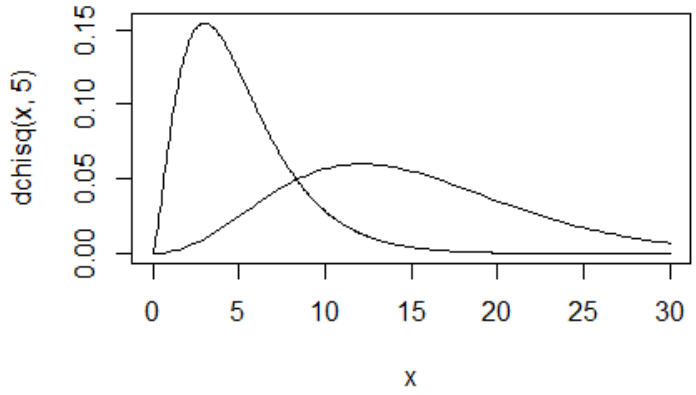
Neste caso, são utilizados 500 pontos (igualmente separados) para desenhar uma curva. As duas curvas não apresentam grandes diferenças, neste caso.

Um outro exemplo do recurso a gráficos é o de verificar a alteração das funções de distribuição de probabilidade em função dos seus parâmetros. Considere que se pretendia examinar o efeito de passar de uma distribuição qui quadrado central para uma distribuição qui quadrado não central. A distribuição não central qui quadrado em **R** é dada pela função *dchisq(x, df,ncp)*, em que *df* são os graus de liberdade e *ncp* o parâmetro não central. Para adicionar uma curva a um gráfico já existente, podemos utilizer o parâmetro *add = TRUE* na função *curve*.

Por exemplo, para adicionar uma curva a uma distribuição qui quadrado com um parâmetro de não centralidade de valor 10 e tomando igualmente, a curva inicial:

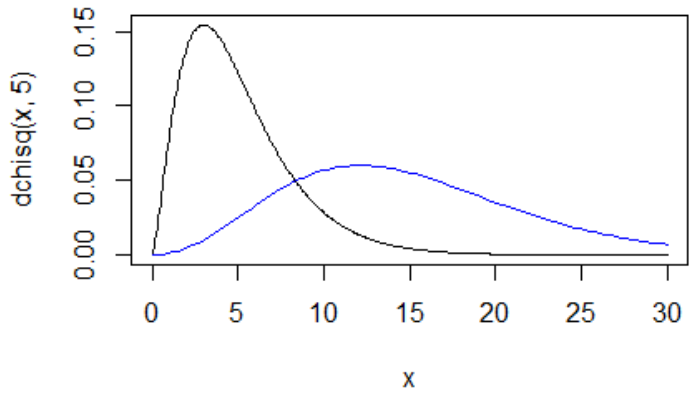
```
> curve( dchisq(x, 5), 0, 30)

> curve( dchisq(x, 5,10), 0, 30, add = TRUE)
```

O resultado é o seguinte, onde são representadas as duas curvas, no mesmo gráfico:



O **R** permite também especificar a cor utilizada nos gráficos, usando o argumento col = "cor", como por exemplo, no comando seguinte que apresenta uma curva em azul (*blue*). Os códigos de cor podem ser consultados no manual do **R**:

```
> curve( dchisq(x, 5,10), 0, 30, add = TRUE, col = "blue")
```



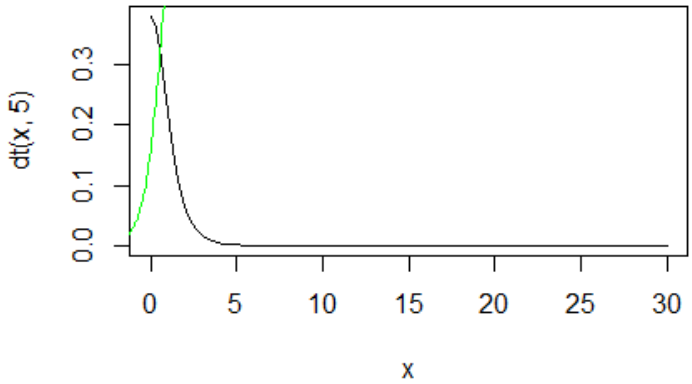
Um outro exemplo é o uso do **R** para desenhar as densidades de probabilidade acumuladas:

```
> curve( dt(x, 5), 0, 30)
```

Neste caso, são desenhadas as probabilidades acumuladas para uma distribuição t de *Student* com 5 graus de liberdade para valores de *x* no intervalo entre -3 e 3.

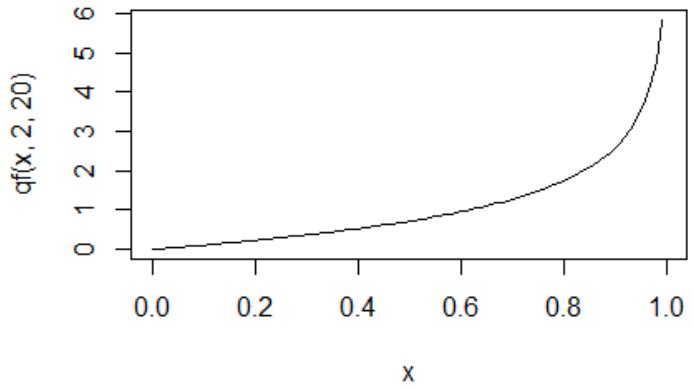
```
> curve( pt(x,5,1), -3, 3, add = TRUE, col = "green")
```

Este comando sobrepõe uma curva verde para a probabilidade acumulada para uma distribuição não central T de Student, com um parâmetro 1, conforme visível no seguinte gráfico:



Um gráfico interessante é o obtido mapeando os valores *x* para um dado valor de probabilidade *p*. Por exemplo, para uma distribuição F central, com 2 e 20 graus de liberdade, uma vez que agora *x* é a probabilidade que varia entre zero e um, obtemos com o seguinte comando em **R**, o respetivo gráfico:

```
> curve( qf(x,2,20), 0, 1)
```



Obter o valor *p* e os valores críticos

O recurso ao comando em **R**, *pfdist(x, parametros)*, retorna o valor *p* associado com o valor *x* que é resultado da distribuição. Isto é, retorna *Pr(fdist(parameteros) ≤ x).*

De igual forma que *qfdis(p, parametros)* retorna o valor *x* para obter um valor *p* em particular.

Por exemplo, qual é o valor *x* de uma distribuição t de *Student* com 12 graus de liberdade, garantindo que a probabilidade de 99% de um valor aleatório é abaixo de *x*? A resposta pode ser obtida pelo comando em **R**:

```
> qt(0.99,12)
```

```
[1] 2.680998
```

De igual forma, se se considerar uma distribuição t de *Student*, com um parâmetro de não centralidade de 1,75 e se observar um valor de 3,5, a probabilidade de ser este valor ou um valor menor, é dada por:

```
> pt(3.5,12,1.75)
```

```
[1] 0.915665
```

O valor *p* é um menos o valor obtido pois a *Pr(fdist > x) = 1 - Pr(fdist ≤ x)*:

```
> 1 - pt(3.5, 12, 1.75)
```

```
[1] 0.08433496
```

Ainda mais um exemplo é o cálculo dos intervalos do nível de confiança α, dados por *xmin* e *xmax*, em que *Pr(fdist ≤ (xmin) = (1 – α/2)* e *Pr(fdist ≤ (xmax) = 1 – (1 – α/2)*.

Considerando um intervalo de confiiança de 99.9% para uma distribuição normal, tomamos assim um valor alfa de $\alpha = 0.999$, e assim $(1 - \alpha) / 2 = 0.0005$. Podemos obter os valores em **R**, para os intervalos superior e inferior, escrevendo os comandos *qnorm(0.0005)* e *qnorm(0.9995)*:

```
> qnorm(0.0005)

[1] -3.290527

> qnorm(0.9995)

[1] 3.290527
```

Como exemplo de uso de outras funcionalidades do **R**, podemos calcular estes valores especificando primeiramente o valor de alfa (α) pretendido, escrevendo o comando:

```
alpha <- 0.999
```

Este comando indica ao **R** o valor de *alfa*. Os valores de limite superior e inferior são obtidos do cálculo de *qnorm((1 - alpha) / 2)* e *qnorm(1 - (1 - alpha) / 2)*

```
> alpha <- 0.999

> qnorm((1-alpha)/2)

[1] -3.290527

> qnorm(1-(1-alpha)/2)

[1] 3.290527
```

Outra funcionalidade útil do **R** é a possibilidade de introduzir um vetor de valores e, para muitas das funções de distribuição em **R**, é obtido como resultado um vetor de valores. É possível definir um vetor de máximas e mínimas probabilidades (para as quais se podem estabeler os valores críticos associados) pelo comando em **R**:

```
> xvals <- c((1-alpha)/2, 1-(1-alpha)/2)
```

Deste modo, o comando *c()* significa criar um vetor a partir (neste caso dois) elementos de uma lista e atribuir ao objeto *xvals*, o valor deste vetor. Usando o comando **R**, *qnorm(xvals)*, o **R** obtêm o valor para os intervalos de acordo com o especificado para o objeto *xvals*.

```
> xvals <- c((1-alpha)/2, 1-(1-alpha)/2)

> qnorm(xvals)

[1] -3.290527  3.290527
```

Enquanto os intervalos de confiança para as distribuições, normal e t de *Student*, são simétricos. Por outro lado, os intervalos de confiança associados com o qui quadrado e F, não são simétricos. Por exemplo, o intervalo de confiança de 95% para uma distribuição F, com 2 e 20 graus de liberdade é obtida, para os seus valores mínimo e máximo, respetivamente pelos comandos **R**:

```
> alpha <- 0.95
```

> qf((1-alpha)/2,2,20)

[1] 0.025349 88

> qf(1-(1-alpha)/2,2,20)

[1] 4.461255

10.1 OUTRAS DISTRIBUIÇÕES DE PROBABILIDADE CONTÍNUAS

Existem outras distribuições de probabilidade em **R**, como é o caso das contínuas. Caso certos parâmetros não sejam especificados, os seus valores por defeito são assumidos. Algumas destas distribuições são as seguintes:

Distribuição	Sintaxe
Exponential	*exp()*: exponencial com grau 1 (por defeito); *exp(grau)*: exponencial com o grau definido pelo utilizador
Uniforme	*unif()*: uniforme em (0,1) (por defeito); *unif(min, max)*: uniforme com valores de mínimo e máximo, definidos pelo utilizador
Logaritmo Normal	*lnorm()*: logaritmo normal, com média de logaritmo igual a zero e desvio padrão de logaritmo igual a 1 (por defeito); *lnorm(medlog, dplog)*: logaritmo normal com média e desvio padrão de logaritmo definidos pelo utilizador
Beta	*beta(forma1, forma2)*: distribuição beta com os parâmetros associados, especificados pelo utilizador
Gamma	*gamma(forma)*: distribuição gama com parâmetro especificado e escala igual a 1 (por defeito); *gamma(forma, escala)*: com os parâmetros especificados pelo utilizador
Weibull	*weibull(forma)*: Weibull com forma especificada e escala igual a um (por defeito); *weibull(forma,escala)*: com os parâmetros especificados pelo utilizador
Wilcox	*wilcox(m,n)*: distribuição para a estatística de categorias de somas, Wilcoxon, com um tamanho de amostra *m* e *n*.

10.2 OUTRAS DISTRIBUIÇÕES DE PROBABILIDADE DISCRETAS

Tal como para o caso das distribuições contínuas, o **R** possui inúmeras distribuições de probabilidade discretas, como os exemplos a seguir apresentados:

Distribuição	Sintaxe
Binomial	*binom(n,p)*: distribuição binomial com tamanho da amostra *n* e sucesso de probabilidade *p*
Geométrica	*geom(p)*: distribuição geométrica, com sucesso de probabilidade *p*
Poisson	*poism(λ)*: distribuição de *Poisson*, com media *λ*
Binomial negativa	*nbinom(tar,p,mu)*: distribuição binomial negativa, com alvo *tar*, probabilidade de sucesso *p* e média *μ*

Um exemplo de utilização para o desenho de uma função discreta exige que primeiro sejam gerada a sequência de inteiros. O **R** pode gerar essa sequência:

```
x <- seq(0, 25, 1)
```

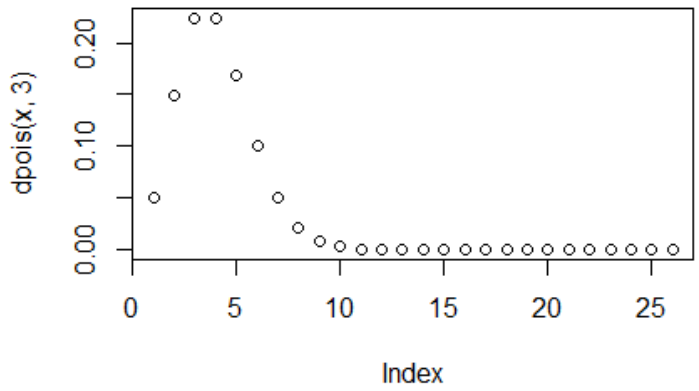
Este comando gera uma sequência de 26 números inteiros, de 0 a 25 em passos de uma unidade e guarda essa sequência no objeto *x* (o comando **R**, *seq(a, b, c),* permite gerar sequências de valores, de *a* a *b*, em passos de *c*.

```
> x
 [1]  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
[20] 19 20 21 22 23 24 25
```

Para desenhar os 26 valores (de 0 a 25) para uma distribuição de *Poisson*, com um parâmetro *λ = 3*, em **R**, utilizando esta sequência, tomamos o seguinte comando:

```
> plot( dpois(x, 3))
```

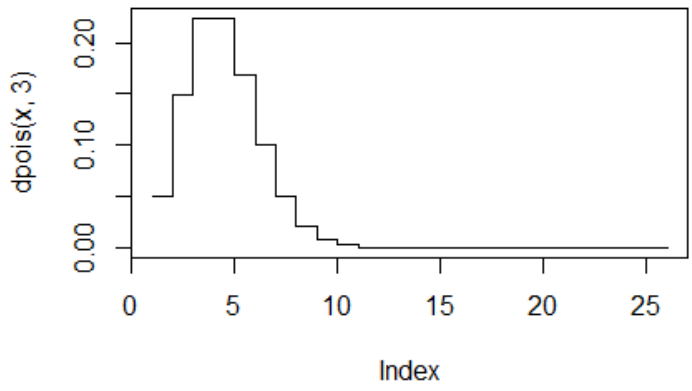
Com o seguinte gráfico obtido:



Considerando o argumento *type* ="" podemos especificar o tipo de gráfico, com um gráfico por escadas, usando *type* = "s" ou um histrograma, usando *type* ="h". Assim, o comando **R**

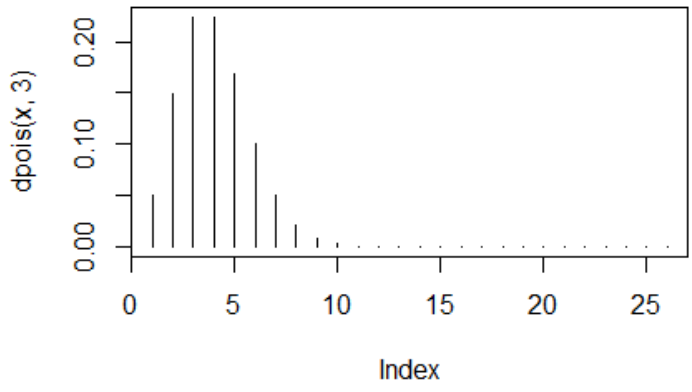
```
> plot( dpois(x, 3), type = "s")
```

Permite obter o seguinte gráfico de escada:




```
> plot( dpois(x, 3), type = "h")
```

Permite obter o seguinte histograma:

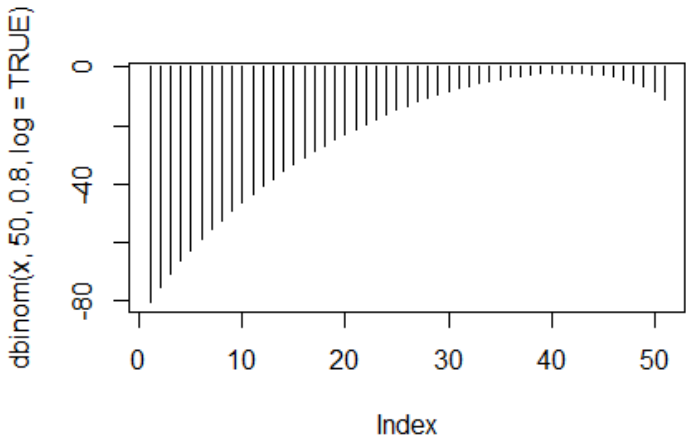


Deve ser registado que os valores desenhados para *x*, maiores que 10, são essencialmente zero. Se se pretender uma maior discriminação, podemos desenhar o logaritmo das probabilidades. Esta operação pode ser realizada com o comando *dDIST*, usando a opção *log* = *TRUE*, que retorna o logaritmo das probabilidades (*log=FALSE* é o valor por defeito que é utilizado a não ser que o contrário seja especificado pelo utilizador). Desta forma, o gráfico das probabilidades (como um histograma) para 51 elementos de uma distribuição binomial com $n = 50$ e uma probabilidade de sucesso $p = 0.8$, é obtida pelos seguintes comandos em **R**:

> x <- seq(0,50,1)

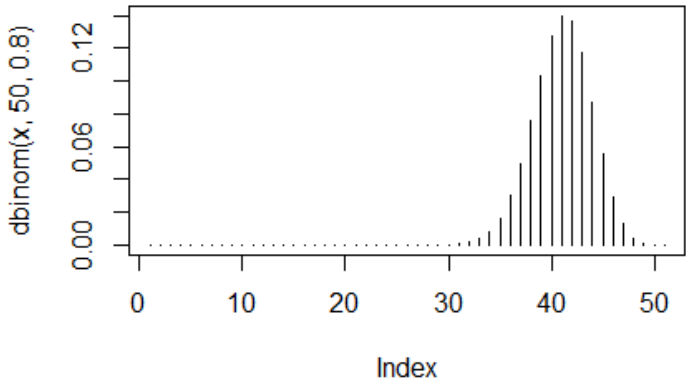
> plot(dbinom(x,50,0.8,log=TRUE),type="h")

Obtendo o seguinte gráfico:



Por sua vez, se se pretender o gráfico das probabilidades numa escala linear usamos o seguinte comando em **R**:

> plot(dbinom(x,50,0.8),type="h")



11. OUTROS TEMAS

Existem inúmeras funcionalidades associadas com o **R** e com as diferentes bibliotecas de funções (pacotes, em Inglês, *packages*) que foram construídas nesta linguagem de programação. Algumas destas funcionalidades incluem:

1. Funções para lidar com os modelos estatísticos como os modelos lineares
2. Funções para lidar com regressão não linear
3. Funções para optimização e resolução de equações
4. Facilidades para programas utilizando ciclos e estruturas condicionais
5. Rotinas de produção de gráficos para visualizar dados tridimensionais

O **R** possui ainda a possibilidade de incluir bibliotecas de funções desenvolvidas por terceiros. Uma vez que existe uma comunidade global, com muitos milhares de utilizadores, permitindo milhares de bibliotecas de funções com as mais diferentes finalidades.

Com a função *library()* podemos ter acesso às bibliotecas disponíveis. Deste modo, o seguinte comando **R**:

```
> library()
```

Fornece uma lista com uam pequena descrição das bibliotecas que já estão instaladas. Com o comando **R**:

```
> library(nome)
```

Em que o nome é a designação da biblioteca pretendida, que dará acesso às funções que nela constam.

12. OBTER MAIS AJUDA E EXPLORAR O R

Este texto apresenta uma breve introdução ao **R** e às suas funcionalidades. Para saber mais ou aprofundar os tópicos aqui apresentados existem diversas alternativas:

1. O sistema de ajuda (*help*) incluído na linguagem é uma excelente referência;
2. Os manuais no *site* official também constituem excelentes recursos, nomeadamente a carta de referência e o indice da linguagem. Mas existem diversos manuais que auxiliam com maior ou menor profundidade, a operação com a linguagem de programação **R**;
3. Existem inúmeros livros de caráter geral ou mais especializado que podem ajudar ao aprofundamento do **R**;
4. Por último, os recursos disponíveis em linha e as comunidades de utilizadores constituem um auxiliar a não deprezar para resolver problemas ou simplesmente tirar dúvidas sobre a linguagem de programação **R**.

13. EXERCÍCIOS RESOLVIDOS

Nesta seção são resolvidos de forma sucessiva os exercícios propostos nos diferentes pontos do texto. Em muitos casos, as soluções são únicas e resultam da aplicação de fórmulas ou constituem resultados obtidos por via dos próprios conceitos em causa. No entanto, existem situações em que a solução oferecida é uma das possíveis e, nestes casos, o critério tomado, foi o de usar uma solução que possa ser mais simples e de fácil compreensão.

EXERCÍCIOS RESOLVIDOS 2.1

1. Defina

```
> x <- c(4, 2, 6)
> y <- c(1, 0, -1)
```

Resposta: x e y são vetores com três elementos cada

```
> x
[1] 4 2 6
> y
[1]  1  0 -1
```

2. Qual o resultado dos comandos seguintes em **R**? Utilize o **R** para confirmar posteriormente as respostas.
   a) length(x)
   b) sum(x)
   c) sum(x^2)
   d) x+y

e) x*y
f) x-2
g) x^2

Resposta: a) retorna o número de elementos do objeto x; b) soma os valores dos elementos do objeto x; c) calcula a potência de cada um dos elementos do objet x e faz a sua soma; d) soma os elementos dos objetos x e y, um a um; multiplica os elementos dos objetos x e y, um a um; f) subtrai dois a cada um dos elementos do objeto x; g) eleva ao quadrado, cada um dos elementos do objeto x

As respostas no **R:**

```
> length(x)
[1] 3
> sum(x)
[1] 12
> sum(x^2)
[1] 56
> x+y
[1] 5 2 5
> x*y
[1]  4  0 -6
> x-2
[1] 2 0 4
> x^2
[1] 16  4 36
```

3. Descreva as seguintes sequências e utilize o **R** para verificar as respostas:
   a) 7:11
   b) seq(2, 9)
   c) seq(4, 10, by=2)
   d) seq(3, 30, length=10)
   e) seq(6, -4, by=-2)

   Resposta: a) gama de valores entre 7 e 11; b) sequência de valores entre 2 e 9; c) sequência de valores entre 4 e 10, copm saltos de duas unidades; d) sequência de valores entre 3 e 30, com saltos de 10 intervalos; e) sequência de valores entre 6 e -4, com saltos de menos duas unidades

   As respostas no **R:**

```
> 7:11
[1]  7  8  9 10 11
> seq(2, 9)
[1] 2 3 4 5 6 7 8 9
> seq(4, 10, by=2)
[1]  4  6  8 10
> seq(3, 30, length=10)
 [1]  3  6  9 12 15 18 21 24 27 30
> seq(6, -4, by=-2)
[1]  6  4  2  0 -2 -4
```

4. Descreva os resultados obtidos com as seguintes expressões em **R** e a seguir, use o **R** para verificar o que está correto:

   a) rep(2,4)
   b) rep(c(1,2),4)
   c) rep(c(1,2),c(4,4))
   d) rep(1:4,4)
   e) rep(1:4,rep(3,4))

Resposta: a) repetir o valor 2, quarto vezes; b) repetir 1 e 2, quatro vezes; c)repetir o valor 1, quarto vezes e o valor 2, quatro vezes; d) repetir a sequência de 1 a 4, quatro vezes; e) repetir a sequência de 1 a 4, cada um dos valores, três vezes
As respostas no **R**:

```
> rep(2,4)
[1] 2 2 2 2
> rep(c(1,2),4)
[1] 1 2 1 2 1 2 1 2
> rep(c(1,2),c(4,4))
[1] 1 1 1 1 2 2 2 2
> rep(1:4,4)
 [1] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4
> rep(1:4,rep(3,4))
 [1] 1 1 1 2 2 2 3 3 3 4 4 4
```

5. Utilize a função *rep* para definir os seguintes vetores em **R**.

   a) 6,6,6,6,6,6
   b) 5,8,5,8,5,8,5,8
   c) 5,5,5,5,8,8,8,8

   Resposta: a) rep(6, 6); b) rep(c(5,8), 4); c) rep(c(5,8), rep(4,2)
   Respostas no **R**:

```
> rep(6,6)
[1] 6 6 6 6 6 6
> rep(c(5,8),4)
[1] 5 8 5 8 5 8 5 8
> rep(c(5,8), rep(4, 2))
[1] 5 5 5 5 8 8 8 8
```

EXERCÍCIOS RESOLVIDOS 3.1

1. Considerando o objeto *x <- c(5,9,2,3,4,6,7,0,8,12,2,9)*, indique qual(ais) o(s) element(s) que cada um dos comandos **R** representa e a seguir , use o **R** para confirmar as respostas:

   a) x[2]
   b) x[2:4]
   c) x[c(2,3,6)]
   d) x[c(1:5,10:12)]
   e) x[-(10:12)]

   Resposta: a) retorno o valor 9; b) retorna os valores 9, 2 e 3; c) retorna os valores 9, 2 e 6; d) retorna os valores 5, 9, 2, 3, 4, 12, 2 e 9; e) retorna os valores 5, 9, 2, 3, 4, 6, 7, 0 e 8.
   Respostas no **R**:

```
> x[2]
[1] 9
> x[2:4]
[1] 9 2 3
> x[c(2,3,6)]
```

```
[1] 9 2 6
> x[c(1:5,10:12)]
[1]  5  9  2  3  4 12  2  9
> x[-(10:12)]
[1] 5 9 2 3 4 6 7 0 8
```

2. Tomando os dados *y <- c(33,44,29,16,25,45,33,19,54,22,21,49,11,24,56)* contendo as vendas de cerveja em barris, para os dias úteis de uma semana, em três lojas diferentes (os primeiros três valores são para as lojas 1, 2 e 3 na segunda-feira e assim sucessivamente). Produza um sumário estatístico para as vendas em cada dia da semana e também por cada loja.
   Resposta: os dados devem ser repartidos através de subscrição, em três grupos: y[c(1,4,7,10,13)]; y[c(2,5,8,11,14)] e y[c(3,6,9,12,15)].
   Respostas no **R**:

```
> y <- c(33,44,29,16,25,45,33,19,54,22,21,49,11,24,56)
> y
 [1] 33 44 29 16 25 45 33 19 54 22 21 49 11 24 56
> y[c(1,4,7,10,13)]
[1] 33 16 33 22 11
> y[c(2,5,8,11,14)]
[1] 44 25 19 21 24
> y[c(3,6,9,12,15)]
[1] 29 45 54 49 56
> summary(y[c(1,4,7,10,13)])
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    11      16      22      23      33      33
> summary(y[c(2,5,8,11,14)])
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   19.0    21.0    24.0    26.6    25.0    44.0
> summary(y[c(3,6,9,12,15)])
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   29.0    45.0    49.0    46.6    54.0    56.0
```

EXERCÍCIOS RESOLVIDOS 4.1

1. Criar em **R** as matrizes

$$x = \begin{bmatrix} 3 & 2 \\ -1 & 1 \end{bmatrix}$$

   e

$$y = \begin{bmatrix} 1 & 4 & 0 \\ 0 & 1 & -1 \end{bmatrix}$$

   Calcular as seguintes expresses e verificar as respostas no **R**:

a) 2 * x
b) x * x
c) x %*% x
d) x %*% y
e) t(y)
f) solve(x)

Resposta: primeiro devem ser criadas as matrizes com recurso ao comando matrix e depois realizadas as operações com os objetos criados

```
> x <- matrix(c(3,-1, 2, 1), nrow=2)
> x
     [,1] [,2]
[1,]    3    2
[2,]   -1    1
> y <- matrix(c(1, 0, 4, 1, 9, -1), nrow=2)
> y
     [,1] [,2] [,3]
[1,]    1    4    9
[2,]    0    1   -1
> 2 * x
     [,1] [,2]
[1,]    6    4
[2,]   -2    2
> x * x
     [,1] [,2]
[1,]    9    4
[2,]    1    1
> x %*% x
     [,1] [,2]
[1,]    7    8
[2,]   -4   -1
> x %*% y
     [,1] [,2] [,3]
[1,]    3   14   25
[2,]   -1   -3  -10
> t(y)
     [,1] [,2]
[1,]    1    0
[2,]    4    1
[3,]    9   -1
> solve(x)
     [,1] [,2]
[1,]  0.2 -0.4
[2,]  0.2  0.6
```

2. Tomando as matrizes *x* e *y* já especificadas em 1., calcule o efeito das seguintes operações de subscrição e verifique as respostas no **R**.

a) x[1,]
b) x[2,]
c) x[, 2]
d) y[1, 2]
e) y[, 2:3]

Resposta: a) todas as colunas da primeira linha de x; b) todas as colunas da segunda linha de x; c) todas as linhas, da segunda coluna de x; d) o elemento da primeira linha e segunda coluna, de y; e) todas as linhas da segunda e terceira coluna de y
Respostas no **R**:

```
> x[1, ]
[1] 3 2
> x[2, ]
[1] -1  1
> x[, 2]
[1] 2 1
> y[1, 2]
[1] 4
> y[, 2:3]
     [,1] [,2]
[1,]    4    9
[2,]    1   -1
```

EXERCÍCIOS RESOLVIDOS 5.1

1. Anexar o conjunto de dados *quakes* e produzir um sumário estatístico das variáveis *depth* e *mag*.

   Resposta: usar o conjunto de dados quakes, com 1000 elementos e 5 dimemsões
   Resposta em **R:**

```
> data(quakes)
> summary(quakes$depth)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   40.0   99.0  247.0  311.4  543.0  680.0
> summary(quakes$mag)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   4.00   4.30   4.60   4.62   4.90   6.40
```

2. Anexar o conjunto de dados *mtcars* e descobrir a média do peso (*weight*) e a media do consumo de combustível (*fuel consumption*) para veículos no conjunto de dados. Use o comando *help(mtcars)* para uma descrição das variáveis disponíveis.

   Resposta: com o recurso ao comando *help(mtcars)* obtemos informação sobre o conjunto de dados mtcars. Com base na descrição, a variável do peso é wt e a do consumo é mpg

   mtcars {datasets} R Documentation
   Motor Trend Car Road Tests
   Description
   The data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models).
   Usage

mtcars

Format

A data frame with 32 observations on 11 variables.

[, 1]	mpg	Miles/(US) gallon
[, 2]	cyl	Number of cylinders
[, 3]	disp	Displacement (cu.in.)
[, 4]	hp	Gross horsepower
[, 5]	drat	Rear axle ratio
[, 6]	wt	Weight (1000 lbs)
[, 7]	qsec	1/4 mile time
[, 8]	vs	V/S
[, 9]	am	Transmission (0 = automatic, 1 = manual)
[,10]	gear	Number of forward gears
[,11]	carb	Number of carburetors

Source

Henderson and Velleman (1981), Building multiple regression models interactively. Biometrics, 37, 391–411.

Respostas em **R**:

```
> data(mtcars)
> help(mtcars)
> mean(mtcars$wt)
[1] 3.21725
> mean(mtcars$mpg)
[1] 20.09062
```

EXERCÍCIOS RESOLVIDOS 6.1

1. Repetir a análise do conjunto de dados *quakes* e *mtcars* utilizando a função *apply* para simplificar os cálculos.

   Resposta: é utilizada a função apply(), com o valor 2 para aplicar a função (mean) às colunas.

   Resposta em **R**:

```
> data(quakes)
> apply(quakes, 2, mean)
        lat        long       depth         mag    stations
-20.64275  179.46202  311.37100    4.62040   33.41800
>
> data(mtcars)
> apply(mtcars, 2, mean)
       mpg         cyl        disp          hp        drat
 20.090625   6.187500 230.721875 146.687500   3.596563
        wt        qsec          vs          am        gear
  3.217250  17.848750   0.437500   0.406250   3.687500
      carb
  2.812500
```

2. Se

$$y = \begin{bmatrix} 1 & 4 & 1 \\ 0 & 2 & -1 \end{bmatrix}$$

qual é o resultado de usar o comando **R** *apply(y[,2:3],1,mean)*? Verificar a resposta no **R**.

Resposta: o comando permite o cálculo da média, dos elementos para uma parte da matriz que compreende todas as linhas da segunda e terceira coluna (a média por linha, resulta da indicação do valor 1) – como são duas colunas, são obtidos dois valores.
Resposta em **R**:

```
> y <- matrix(c(1,0, 4, 2, 1, -1), nrow=2)
> y
   [,1] [,2] [,3]
[1,]   1   4   1
[2,]   0   2  -1
> apply(y[,2:3],1, mean)
[1] 2.5 0.5
```

EXERCÍCIOS RESOLVIDOS 7.1

1. Suponha $X \sim N(2, 0.25)$. Tome f e F como as funções de densidade e de distribuição de X, respetivamente. Utilize o **R** para o seu cálculo

   a) f(0.5)
   b) F(2.5)
   c) F−1(0.95) (observação: F−1 é a função quartil)
   d) Pr(1 ≤ X ≤ 3)

   Resposta:considerando uma distribuição normal com média 2 e variância de 0.25 e aplicamas a função de densidade (dnorm); a função de distribuição (pnorm) e a função de quartis (qnorm)
   Respostas em **R**:

```
> dnorm(0.5, 2, sqrt(0.25))
[1] 0.008863697
> pnorm(2.5, 2, sqrt(0.25))
[1] 0.8413447
> qnorm(0.95, 2, sqrt(0.25))
[1] 2.822427
> pnorm(3, 2, sqrt(0.25)) - pnorm(1, 2, sqrt(0.25))
[1] 0.9544997
```

2. Repetir a questão 1, mas para o caso de X possuir uma distribuição *t-Student* com 5 graus de liberdade.

Resposta: a estrutura das funções é semelhante. Neste caso, mas com recurso à distribuição t-Student que, em R, é a função t() e nos seus parâmetros que são os graus de libertadade e não a média e o desvio padrão.
Respostas em **R**:

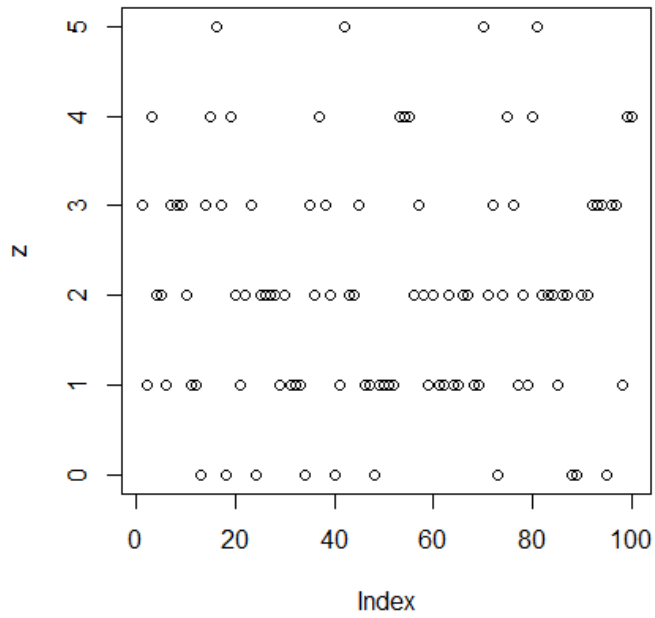
```
> dt(0.5, 5)
[1] 0.3279185
> pt(2.5, 5)
[1] 0.972755
> qt(0.95, 5)
[1] 2.015048
> pt(3, 5) - pnorm(1, 5)
[1] 0.9849187
```

3. Utilizar a função *rpois* para simular 100 valores de uma distribuição de *Poisson* com um parâmetro à sua escolha. Produzir um sumário estatístico do resultado e verificar se a média e a variância se encontram alinhadas de forma razoável com os valores expectáveis para este tipo de distribuição.

   Resposta: a função em R, para a distribuição de Poisson é pois(), pelo que podemos gerar os 100 valores pretendidos com a função rpois() em R. Optamos por um valor lambda (média) de 2 e verificamos os valores obtidos pelo recurso à função summary e a um gráfico.
   Resposta no **R**:

```
> z<-rpois(100, 2)
> print(z)
 [1] 3 1 4 2 2 1 3 3 3 2 1 1 0 3 4 5 3 0 4 2 1 2 3 0 2 2 2 2
[29] 1 2 1 1 1 0 3 2 4 3 2 0 1 5 2 2 3 1 1 0 1 1 1 1 4 4 4 2
[57] 3 2 1 2 1 1 2 1 1 2 2 1 1 5 2 3 0 2 4 3 1 2 1 4 5 2 2 2
[85] 1 2 2 0 0 2 2 3 3 3 0 3 3 1 4 4
> summary(z)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00    1.00    2.00    2.05    3.00    5.00
> plot(z)
```

   Com o seguinte gráfico…



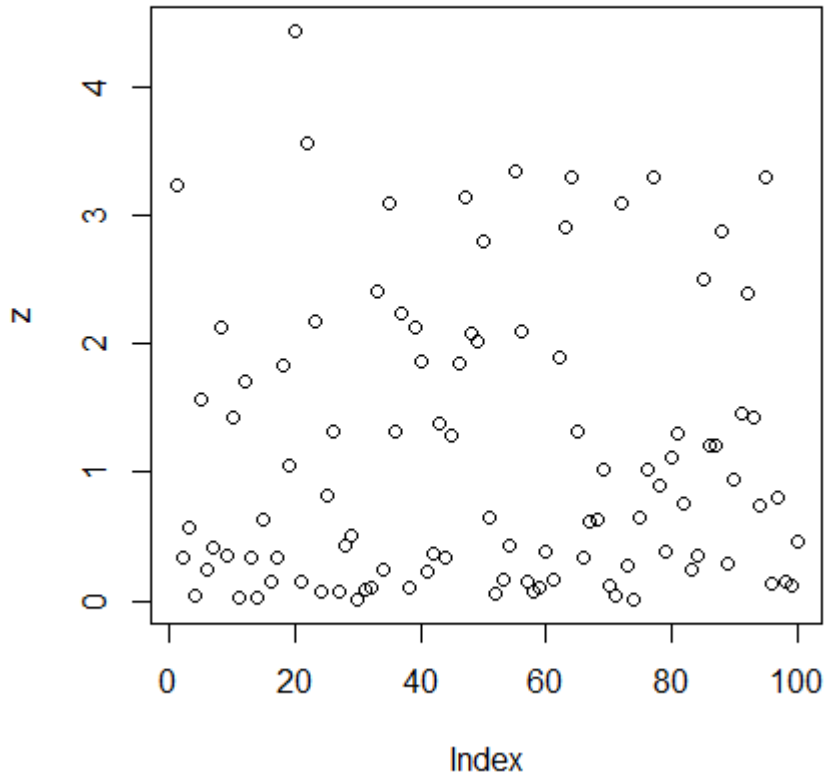
4. Repetir a questão anterior, substituindo a função *rpois* pela função *rexp*.

Resposta: a função exp() em R, corresponde à distribuição exponencial.
Resposta em **R**:

```
> z<-rexp(100)
> print(z)
  [1] 3.243288502 0.334403874 0.565296485 0.038088761
  [5] 1.570319107 0.243206803 0.416041951 2.124290550
  [9] 0.359639727 1.434833836 0.029404410 1.715524647
 [13] 0.340645791 0.028709122 0.633585621 0.146124342
 [17] 0.339303151 1.836162031 1.053370211 4.444685776
 [21] 0.155877044 3.559720969 2.180055863 0.075768514
 [25] 0.825705528 1.321309703 0.070849720 0.433318702
 [29] 0.501042530 0.005925449 0.094108249 0.102410426
 [33] 2.411345610 0.238463433 3.096472069 1.326701608
 [37] 2.247087410 0.098240762 2.136931630 1.868768170
 [41] 0.234124697 0.372599990 1.387738513 0.336196721
 [45] 1.291198949 1.854786437 3.144629963 2.078206321
 [49] 2.015610678 2.802463261 0.649139516 0.056547353
 [53] 0.167901265 0.432195359 3.344218500 2.098952111
 [57] 0.154293520 0.070339032 0.105106640 0.387316244
 [61] 0.162418266 1.896728661 2.903008975 3.299591608
 [65] 1.316174636 0.339452593 0.611313831 0.627381224
 [69] 1.016115189 0.124585460 0.045230954 3.098827720
 [73] 0.275839694 0.016694816 0.649524164 1.017719676
```

```
 [77] 3.299543582 0.896699076 0.383314801 1.109418282
 [81] 1.303325308 0.754214881 0.247183943 0.345469115
 [85] 2.499414935 1.212014548 1.207664099 2.880381770
 [89] 0.284900721 0.943669620 1.460672555 2.394440211
 [93] 1.420871649 0.741623335 3.292838573 0.130901496
 [97] 0.799506467 0.151379642 0.122082958 0.468276059
> summary(z)
   Min.  1st Qu.   Median    Mean  3rd Qu.    Max.
0.005925 0.242000 0.695600 1.124000 1.858000 4.445000
> plot(z)
```

Com o seguinte gráfico…



EXERCÍCIOS RESOLVIDOS 8.1

1. Utilize

```
> x <- rnorm(100)
```

Para gerar dados aleatórios. Produza uma figura que mostre um histrograma e um diagram de caixas (*boxplot*) com os dados gerados. Modifique os nomes dos eixos e o título do gráfico, de modo a realizar a sua personalização.

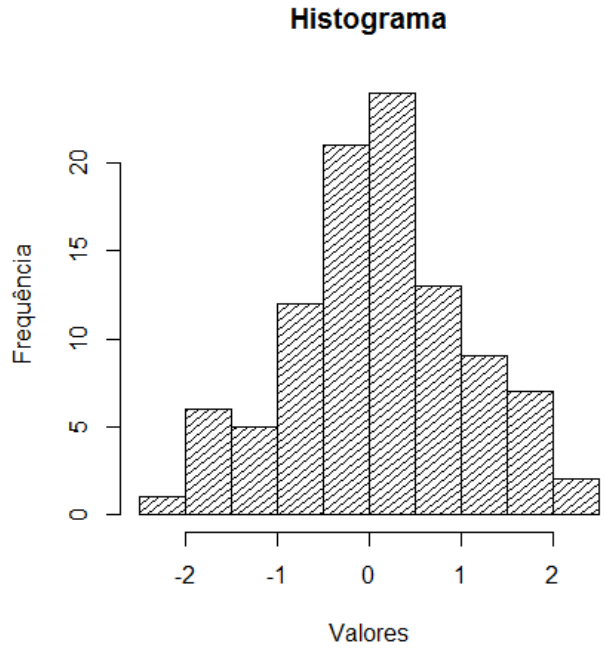
Resposta: o R possui um conjunto de funções poderoso para a geração de gráficos. Para obter histogramas, usamos a função hist() e para obter gráficos de caixas, a função boxplot(). Em ambas existem inúmeros argumentos para personalizar os gráficos.

Resposta em **R**:

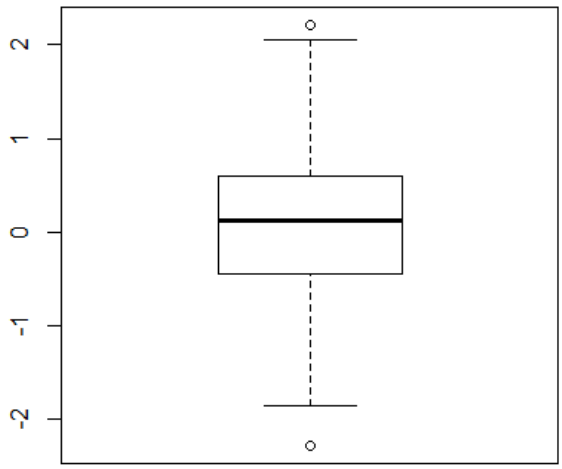
```
> hist(x, xlab="Valores", ylab="Frequência",main="Histograma", density=20)
> boxplot(x)
```

Com os seguintes gráficos...

Do hist(x, ...)



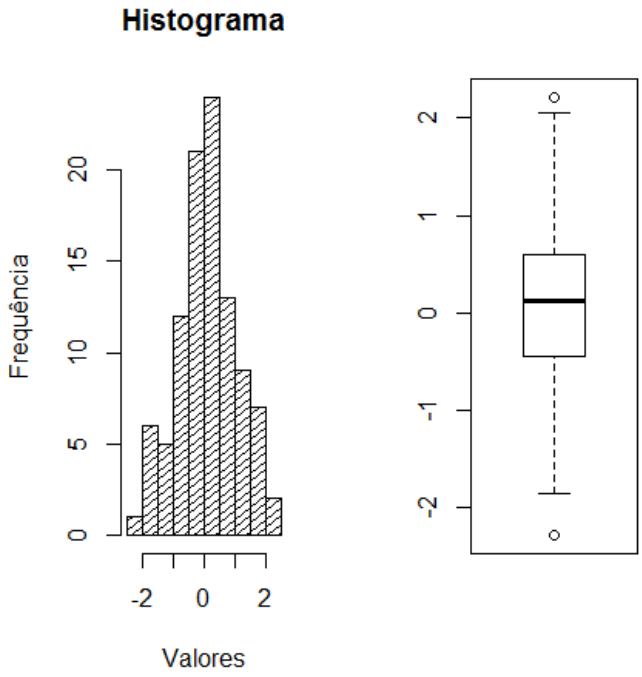
Do boxplot(x)



Podemos ainda considerar gerar os dois gráficos, lado a lado, com recurso ao comando R, par, com o argumeto mfrow:

```
par(mfrow=c(1,2))
hist(x, xlab="Valores", ylab="Frequência",main="Histograma", density=20)
boxplot(x)
par(mfrow=c(1,1))
```

Com o seguinte gráfico



2. Execute os seguintes comandos **R**:

```
> x <- (-10):10
> n <- length(x)
> y <- rnorm(n, x, 4)
> plot(x, y)
> abline(0, 1)
```

Tente entender o efeito de cada comando no gráfico produzido. Modifique o código da seguinte forma a verifique quais as alterações produzidas ao gráfico

```
x <- (-10):10
n <- length(x)
y <- rnorm(n, x, 4)
plot(x, y, pch=16, cex=1.4, col = "blue")
abline(lm(y~x))
```

Resposta: as primeiras três linhas são iguais. Nestas é criado com vetor com uma sequência de 21 números de -10 a 10, tomada a dimensão do vetor (21) e criado um vetor também com 21 números aleatórios com distribuição normal e com média do valor do elemento de x correspondente e desvio padrão de 4. Depois é criado um gráfico de pontos dos valores de x e y e desenhada uma linha no gráfico com interceção zero e declive um. No segundo grupo de código, o gráfico possui argumentos que permitem a sua personalização e adição de cor aos pontos e a linha no gráfico, agora é o resultado da regressão entre as variáveis
Resposta em **R**:
Gráfico do primeiro bloco de código

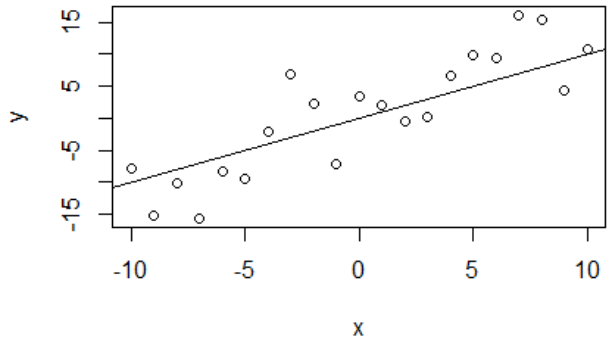
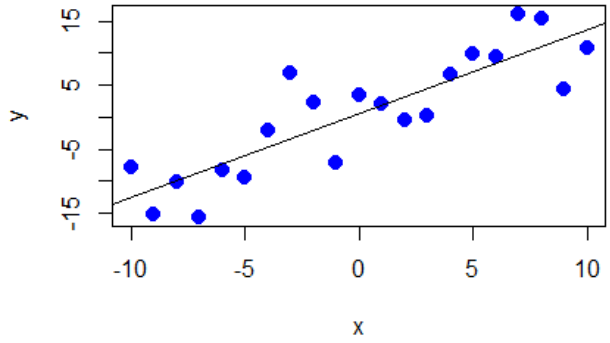


Gráfico do segundo bloco de código

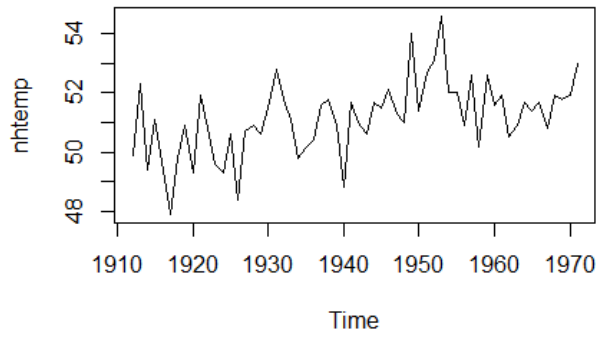


5. Execute o seguinte código em **R**:

```
> data(nhtemp)
> plot(nhtemp)
```

É produzido um gráfico de uma série temporal com as medidas das temperaturas médias anuais para New Hampshire, nos Estados Unidos.

Resposta e resultado dos comandos em **R**: Neste caso, basta executar os comandos, que é gerado o seguinte gráfico:



6. O exemplo anterior ilustra que a função *plot* atua de forma diferente, em função dos objetos serem de diferentes tipos. O objeto *nhtemp* possui uma classe especial de uma série de tempo. Mais genericamente, podemos ter os dados das observações anuais num vetor, mas precisamos de construir o gráfico da série de tempo. A operação pode ser realizada com o comando em **R**:

   > temp <- as.vector(nhtemp)

   Que cria um vetor temp que contém as temperaturas anuais. Uma forma de produzir um gráfico semelhante à série de tempo é com o comando em **R**:

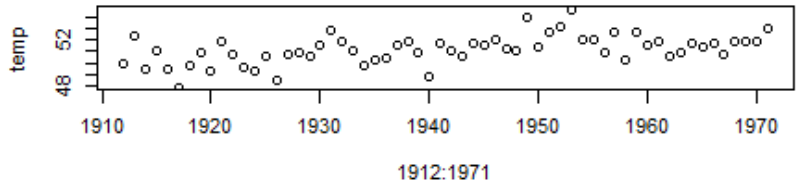
   > plot(1912:1971, temp)

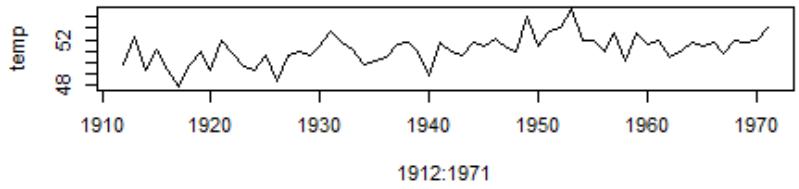
   No entanto, este gráfico coloca pontos em vez de linhas. Para unir os dados com linhas, podemos utilizar:

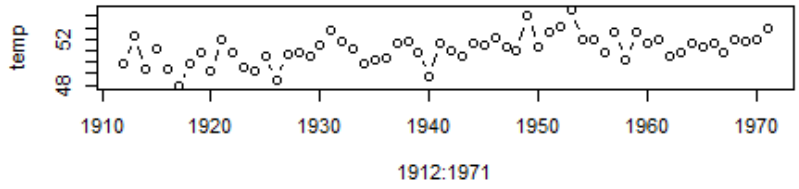
   > plot(1912:1971, temp, type='l')

   O argumento *type*, também permite a colocação simultânea de pontos e linhas, com *type='b'*.

   Resposta e resultado dos comandos em **R**: são apresentados três gráficos, para os dois comandos plot referidos e para um terceiro comando com o argumento type='b':







EXERCÍCIOS RESOLVIDOS 9.1

1. Escreva uma função em **R** que tome como argumentos dois vetors *x* e *y* e produz um gráfico do tipo *scatterplot* e calcula o coeficiente de correlação (utilizando a função *cor(x,y)*).

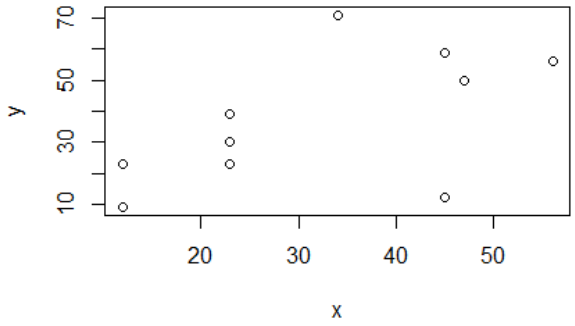
   Resposta:

```
fplotcor <- function(x, y) {
  plot(x,y)
  print("Correlação entre as variáveis X e Y")
  cor(x,y)
}
```

   Comandos em **R**:

```
> x <- c(12, 23, 34, 23, 45, 23, 56, 12, 45, 47)
> y <- c(9, 30, 71, 39, 59, 23, 56, 23, 12, 50)
> fplotcor(x, y)
[1] "Correlação entre as variáveis X e Y"
[1] 0.5701789
```

O gráfico criado:



2. Escreva uma função em **R** que tome um vetor *(x1, x2, … ,xn)* e calcule a soma do valor de *xi* com a posição *i*). Por exemplo, um vetor com três valores: *(2, 4, 1)*, fica com o cálculo exemplificado de *(1+2, 2+4, 3+1)* e com o resultado final de *(3, 6, 4)*.

Resposta:

```
transfvet <- function(x) {
 tam <- length(x)
 for (i in 1:tam) {
  x[i]<-x[i] + i
 }
 print(x)
}
```

Comandos em **R**:

```
> y <- c(1, 1, 1, 1, 1, 1)
> transfvet(y)
[1] 2 3 4 5 6 7
```

3. Escreva uma função em **R** para criar um vetor que fornece os n primeiros números de Fibonacci. Os números de Fibonacci são gerados em sequência, sendo os seus dois primeiros valores, a unidade e os seguintes, a soma dos dois números anteriores. A sequência de números segue a sequência: *1, 1, 2, 3, 5, 8, 13, 21, …*

Resposta:

```
fibonacci <- function (n) {
 if(n < 3) {
  print("Primeiros dois valores iguais a um")
 } else {
  fib <- rep(1,n)
  for(i in 3:n) {
   fib[i] <- fib[i-1] + fib[i-2]
  }
  print(fib)
```

```
 }
}
```

Comandos em **R**:

```
> fibonacci(1)
[1] "Primeiros dois valores iguais a um"
> fibonacci(2)
[1] "Primeiros dois valores iguais a um"
> fibonacci(3)
[1] 1 1 2
> fibonacci(5)
[1] 1 1 2 3 5
> fibonacci(10)
 [1]  1  1  2  3  5  8 13 21 34 55
```

4. Escreva uma função em **R** para listar os fatores que dividem um dado número com resto zero. A função deve no final, indicar o número de fatores que esse número possui

Resposta:

```
fatores <- function(x) {
 print(paste("Número:",x, "divisível por:"))
 cont <- 0
 for(i in 1:x) {
  if((x %% i) == 0) {
   print(i)
   cont <- cont + 1
  }
 }
 print(paste("Número de fatores:", cont))
}
```

Comandos em **R**:

```
> fatores(3)
[1] "Número: 3 divisível por:"
[1] 1
[1] 3
[1] "Número de fatores: 2"
> fatores(24)
[1] "Número: 24 divisível por:"
[1] 1
[1] 2
[1] 3
[1] 4
[1] 6
[1] 8
```

```
[1] 12
[1] 24
[1] "Número de fatores: 8"
> fatores(48)
[1] "Número: 48 divisível por:"
[1] 1
[1] 2
[1] 3
[1] 4
[1] 6
[1] 8
[1] 12
[1] 16
[1] 24
[1] 48
[1] "Número de fatores: 10"
```

5. Escreva uma função em R que solicite um número inteiro do utilizador e verifique se se trata de um número primo.

Resposta:

```
primo <- function() {
  num = as.integer(readline(prompt="Introduza um número inteiro: "))
  flag = 0
  if(num > 1) {
    flag = 1
    for(i in 2:(num/2)) {
      if ((num %% i) == 0) {
        flag = 0
        break
      }
    }
  }
  if(num == 2)   flag = 1
  if(flag == 1) {
    print(paste(num,"é um número primo"))
  } else {
    print(paste(num,"não é número primo"))
  }
}
```

Comandos em **R**:

```
> primo()
Introduza um número inteiro: 0
[1] "0 não é número primo"
> primo()
Introduza um número inteiro: 1
```

```
[1] "1 não é número primo"
> primo()
Introduza um número inteiro: 12
[1] "12 não é número primo"
> primo()
Introduza um número inteiro: 7
[1] "7 é um número primo"
> primo()
Introduza um número inteiro: 487
[1] "487 é um número primo"
>
```