CE-223: Estatística Computacional

Primeiro Semestre de 2008

Paulo Justiniano Ribeiro Junior

Última atualização: 17 de junho de 2008

1 Usando o LINUX no LABEST e LM

Nesta aula é feita uma introdução ao sistema operacional *LINUX* que vem sendo adotado no LABEST. É ainda mostrado como rodar o programa `R` neste sistema.

1.1 Comandos básicos do *LINUX*

Aqui estão alguns comandos básicos do *LINUX*:

Todos os comandos são documentados com `man` e possuem diversas outras opções.
Por exemplo para ver a documentação e opções do comando `tail` digite:
`man tail`
Para sair da tela de ajuda co comando basta digitar a tecla `q`

1.2 Praticando alguns comandos

Entre em sua conta, abra um terminal (clique no botão `xterm`) e faça o seguinte, utilizando os comandos da tabela acima.

1. inspecione o conteúdo do diretório com o comando `ls`
2. use o editor `nano` para criar um arquivo chamando `arquivo.txt`. Para abrir o editor digite no prompt do Linux:

   ```
   nano
   ```

   Digite o texto abaixo no editor:

   ```
   Este é um texto digitado no Linux usando o editor nano.
   ```

3. grave o arquivo e saia do editor. Para isto veja as opções na parte de baixo da tela do *nano*. Note que o caracter `^` corresponde à tecla `CTRL`. Portanto para gravar o arquivo você vai precisar teclar `CTRL-O` (tecla "control" mais o caracter "O")
4. inspecione novamente o conteúdo do diretório com o comando `ls`
5. troque o nome do arquivo de `arquivo.txt` para `arq1.txt`
6. use o comando `more` para visualizar o conteúdo do arquivo

Tabela 1: Alguns comandos básicos do LINUX

who	mostra os usuários logados no sistema
w	também mostra os usuários logados no sistema
quota -v	mostra informações sobre cotas na área do usuário
du -hs *	mostra o espaço usado por cada arquivo/diretório de usuário
ls ls -l ls -a	lista conteúdo do diretório local mostra conteúdo detalhado mostra arquivos escondidos
mkdir	cria diretório
cp cp -r	copia arquivo copia recursivamente (para copiar diretórios)
mv	mover ou renomear arquivo/diretório
rm rm -r rm -rf	apaga arquivo apaga recursivamente apaga recursivamente sem confirmação (use com cuidado!)
cd	muda de diretório
pwd	mostra o diretório atual
cat, more ou less	mostram conteúdos de arquivo
tail head	mostra final de arquivo mostra começo de arquivo
zip e unzip	comprime/descomprime arquivos .zip
gzip e gunzip	comprime/descomprime arquivos .gz
gv xpdf	mostra arquivos postscript (.ps) mostra arquivos em ¨portable document format¨ (.pdf)
ssh scp	acessa outra máquina Linux via protocolo seguro SSH copia arquivos entre máquinas Linux via protocolo seguro
grep	procura por palavra ou expressão em um ou mais arquivos
rgrep	procura por palavra ou expressão recursivamente
chmod	muda permissão de arquivos e diretórios
locate	procura por um nome de arquivo/diretório
passwd	troca a senha
nano	abre o editor *nano*
emacs	abre o editor *emacs*
kile	abre o editor *kile* adequado para edição de textos em LaTeX
mozilla opera	abre o browser *Mozilla* abre o browser *Ópera*
ooffice	abre o *OpenOfficce*
R	abre o programa R
disquete*	abre programa para transferência de arquivos da área do usuário para disquete inserido em drive local

O símbolo * indica comando exclusivo para uso nos terminais do LABEST.

7. crie um diretório chamando `aula1`
8. copie o arquivo `arq1.txt` para dentro deste diretório
9. digite `pwd` e veja (e entenda) o que sai na tela
10. entre no diretório `aula1`

11. digite novamente `pwd` e veja o que sai na tela
12. volte para o seu diretório "raiz" usando o comando `cd`
13. digite `pwd` de novo e veja "onde você está agora" (em qual diretório)
14. digite o comando `ls` e veja o resultado
15. apague o arquivo `arq1.txt`
16. digite novamente o comando `ls` e veja o resultado
17. entre no diretório `aula1`
18. use o comando `pwd` para ver se você está no diretório correto
19. abra agora um novo arquivo chamando `arq2.R` usando o `emacs`
20. digite neste arquivo as seguinte linhas:

```
x <- rnorm(100)
summary(x)
hist(x)
sum(x > 0)
```

21. grave o arquivo e feche o editor `emacs`
22. veja o conteúdo do diretório com o comando `ls`
23. abra o editor `openoffice` e digite o seguinte texto

```
Este é um texto digitado no Linux usando o editor OpenOffice.
O Openoffice é uma alternativa ao MS-Office.
```

24. grave o texto num arquivo com o nome `arq3` no formato do openoffice
25. grave o texto num arquivo com o nome `arq3` no formato do MS-Word (extensão .doc)
26. feche o editor e retorne à linha de comando
27. liste os arquivos agora existentes em seu diretório `aula1`
28. use o Openoffice para criar uma planilha com os seguinte dados

```
A   12
A   13
A   11
A   10
B   14
B   15
B   12
B   13
```

29. salve esta planilha num arquivo com o nome `arq4` no formato openoffice

30. salve esta planilha num arquivo com o nome `arq4` no formato do MS-Excel
31. feche o programa openoffice
32. liste os arquivos nos seu diretório
33. volte ao seu diretório raiz.

1.3 Alguns links

Alguns links com material introdutório sobre o *LINUX*:

- Apostila preparada por Stonebank é um excelente material introdutório.
- A Apostila preparada pelo PET-Informática é um excelente material introdutório.
- O Linux é um sítio com muitas dicas e tutoriais.

Links para algumas distribuições *LINUX*:

- Debian-Linux é a distribuição usada no LABEST/C3SL.
- Ubuntu-Linux é uma distribuição muito amigével, fácil de instalar e de usar a qual recomendamos o uso para instalação em computadores individuais.

1.4 Rodando o programa R no *LINUX*

O programa R pode ser rodado no *LINUX* de duas formas:

1. na linha do comando do LINUX (console) – basta digitar `R` na linha de comando do Linux.
2. dentro do editor *Xemacs (ou emacs)*, assim como é feito no Windows. Para isto inicie o editor com o comando `emacs &` e depois inicie o Rcom a combinação de teclas `ESC SHIFT-X SHIFT-R`.

Neste curso será dada preferência à segunda forma, i.e. rodar o R dentro do *Emacs*. Maiores detalhes sobre este mecanismo são fornecidos no Tutorial de Introdução ao R.

2 Instalando o R

Há várias formas de se instalar o R que basicamente pode ser reunidas em duas formas: (i) instalação usando arquivos binários ou (ii) instalação compilando os arquivos fonte.

1. **A partir de arquivos compilados**

   Para isto é necessário baixar o arquivo de instalação adequado a seu sistema operacional e rodar a instalação. Nas áreas de *download* do R, como por exemplo em http://cran.br.r-project.org voce irá encontrar arquivos de instalação para os sistemas operacionais Linux, Windows e Macintosh.

   No caso do Windows siga os *links*:
   `Windows (95 and later) --> base`
   e copie o arquivo de instalação `.exe` que deve ser rodado para efetuar a instalação.

   Além disto o R está disponível como pacote de diversas distribuições LINUX tais como Ubuntu, Debian, RedHat (Fedora), Suse, entre outras. Por exemplo, para instalar no Debian ou Ubuntu LINUX pode-se fazer (com privilágios de **root**):

   (a) No arquivo `/etc/apt/sources.list` adicione a seguinte entrada:
   - Ubuntu:
     `deb http://cran.R-project.org/bin/linux/ubuntu dapper/`
   - Debian:
     `deb http://cran.R-project.org/bin/linux/debian stable/`

   (b) atualize a lista de pacotes com:
   `apt-get update`

   (c) A seguir rode na linha de comando do LINUX:
   ```
   apt-get install r-base r-base-core r-recommended
   apt-get install r-base-html r-base-latex r-doc-html r-doc-info r-doc-pdf
   ```

   Além destes há diversos outros pacotes Debian para instalação dos pacotes adicionais do R e outros recursos.

2. **Compilando a partir da fonte**

   Neste caso pode-se baixar o arquivo fonte do R (`.tar.gz`) que deve ser descompactado e instruções para compilação devem ser seguidas.

   Eu pessoalmente prefiro rodar os comandos disponíveis neste link.

   Maiores informações podem ser obtidas o manual R Instalation and Administration

3 Introdução

O programa computational R é gratuito, de código aberto e livremente distribuído e proporciona um ambiente para análises estatísticas. Seguem algumas informações básicas sobre este sistema.

3.1 O projeto R

O programa R é gratuito e de código aberto que propicia excelente ambiente para análises estatísticas e com recursos gráficos de alta qualidade. Detalhes sobre o projeto, colaboradores, documentação e diversas outras informações podem ser encontradas na página oficial do projeto em: http://www.r-project.org.

O programa pode ser copiado livremente pela internet. Há alguns espelhos (*mirrors*) brasileiros da área de *downloads* do programa chamada de CRAN (Compreensive R Arquive Network), entre eles um situado no C3SL/UFPR que pode ser acessado em `http://cran.br-r-project.org`

Será feita uma apresentação rápida da página do R durante o curso onde os principais recursos serão comentados assim como as idéias principais que governam o projeto e suas direções futuras.

3.2 Um tutorial sobre o R

Além dos materiais disponíveis na página do programa há também um *Tutorial de Introdução ao* R disponível em `http://www.est.ufpr.br/Rtutorial`.

Sugerimos aos participantes deste curso que percorram todo o conteúdo deste tutorial e retornem a ele sempre que necessário no decorrer do curso.

3.3 Utilizando o R

Siga os seguintes passos.

1. Inicie o R em seu computador. Para iniciar o Rno LINUX basta digitar R na linha de comando.
2. Você verá o símbolo > indicando onde você irá digitar comandos.
   Este é o *prompt* do R indicando que o programa está pronto para receber seus comandos.
3. A seguir digite (ou "recorte e cole") os comandos mostrados neste material.
   No restante deste texto vamos seguir as seguintes convenções:
   - comandos do R são sempre mostrados em fontes do tipo `typewriter como esta`;
   - linhas iniciadas pelo símbolo # são comentários e são ignoradas pelo R.

3.4 Cartão de referência

Para operar o R é necessário conhecer e digitar comandos. Isto pode trazer alguma dificuldade no inicio até que o usuário se familiarize com os comandos mais comuns. Uma boa forma de aprender e memorizar os comandos básicos é utilizar um Cartão de Referência que é um documento que você pode imprimir e ter sempre com você e que contém os comandos mais frequentemente utilizados. Aqui vão três opções:

- Cartão de Referência em formato HTML e traduzido para português.
- Cartão de Referência em formato PDF preparado por Jonathan Baron.
- Cartão de Referência em formato PDF preparado por Tom Short.

3.5 Rcmdr - "The R commander" — "menus" para o R

Para operar o R, na forma usual, é necessário conhecer e digitar comandos. Alguns usuários acostumados com outros programas notarão de início a falta de "menus". Na medida que utilizam o programa, os usuários (ou boa parte deles) tendem a preferir o mecanismo de comandos pois é mais flexível e com mais recursos.

Entretanto, alguns iniciantes ou usuários esporádicos poderão ainda preferir algum tipo de "menu".

O pacote **Rcmdr** foi desenvolvido por John Fox visando atender a esta demanda. Para utilizar este pacote basta instalá-lo e carregar com o comando `require(Rcmdr)` e o menu se abrirá automaticamente.

Atençao: Note que o **Rcmdr** não provê acesso a toda funcionalidade do R mas simplesmente a alguns procedimentos estatísticos mais usuais.

Maiores informações sobre este pacote podem ser encontradas na página do **Rcmdr**.

4 Aritmética e Objetos

4.1 Operações aritméticas

Você pode usar o R para avaliar algumas expressões aritméticas simples. Por exemplo:

```
> 1+2+3         # somando estes números ...
[1] 6
> 2+3*4         # um pouquinho mais complexo
[1] 14
> 3/2+1
[1] 2.5
> 4*3**3        # potências são indicadas por  ** ou ^
[1] 108
```

Nos exemplos acima mostramos uma operação simples de soma. Note no segundo e terceiro comandos a prioridade entre operações. No último vimos que a operação de potência é indicada por `**`. Note que alternativamente pode-se usar o símbolo `^`, por exemplo `4*3^3` produziria o mesmo resultado mostrado acima.

O símbolo `[1]` pode parecer estranho e será explicado mais adiante. O R também disponibiliza funções usuais como as que são encontradas em uma calculadora:

```
> sqrt(2)
[1] 1.414214
> sin(3.14159)       # seno de (Pi radianos) é zero
[1] 2.65359e-06
```

Note que o ângulo acima é interpretado como sendo em radianos. O valor `Pi` está disponível como uma constante. Tente isto:

```
> sin(pi)
[1] 1.224606e-16
```

Aqui está uma lista resumida de algumas funções aritméticas no R:

Estas expressões podem ser agrupadas e combinadas em expressões mais complexas:

```
> sqrt(sin(45 * pi/180))
[1] 0.8408964
```

4.2 Valores faltantes e especiais

Vimos nos exemplos anteriores que `pi` é um valor especial, que armazena o valor desta constante matemática. Existem ainda alguns outro valores especiais usados pelo R:

- `NA` ***N****ot* ***A****vailable*, denota dados faltantes. Note que deve utilizar maiúsculas.
- `NaN` ***N****ot* ***a*** ***N****umber*, denota um valor que não é representável por um número.
- `Inf` e `-Inf` mais ou menos infinito.

Vejamos no exemplo abaixo alguns resultados que geram estes valores especiais. No final desta sessão revisitamos o uso destes valores.

```
> c(-1, 0, 1)/0
[1] -Inf  NaN  Inf
```

sqrt()	raiz quadrada
abs()	valor absoluto (positivo)
sin() cos() tan()	funções trigonométricas
asin() acos() atan()	funções trigonométricas inversas
sinh() cosh() tanh()	funções hiperbólicas
asinh() acosh() atanh()	funções hiperbólicas inversas
exp() log()	exponencial e logarítmo natural
log10() log2()	logarítmo base-10 e base-2
gamma()	função Gamma de Euler
factorial	fatorial ($n!$)
choose()	número de combinações ($\frac{n!}{x!(n-x)!}$)
combn()	todas conjuntos gerados pela combinações de certo número de elementos

4.3 Objetos

O R é uma linguagem orientada à objetos: variáveis, dados, matrizes, funções, etc são armazenados na memória ativa do computador na forma de objetos. Por exemplo, se um objeto `x` tem o valor 10, ao digitarmos o seu nome, o programa exibe o valor do objeto:

```
> x <- 10
> x
[1] 10
```

O dígito 1 entre colchetes indica que o conteúdo exibido inicia-se com o primeiro elemento do objeto `x`. Você pode armazenar um valor em um objeto com certo nome usando o símbolo `<-`. Exemplos:

```
> x <- sqrt(2)      # armazena a raiz quadrada de 2 em x
> x                 # digite o nome do objeto para ver seu conteúdo
[1] 1.414214
```

Neste caso lê-se: *x "recebe"a raiz quadrada de 2.* Alternativamente ao símbolo `<-` usualmente utilizado para atribuir valores a objetos, pode-se ainda usar os símbolos `->` ou `=` (este apenas em versões mais recentes do R). O símbolo `_` que podia ser usado em versões mais antigas no R tornou-se inválido para atribuir valores a objetos em versões mais recentes e passou a ser permitido nos nomes dos objetos. As linhas a seguir produzem o mesmo resultado.

```
> x <- sin(pi)
> x
[1] 1.224606e-16
> x <- sin(pi)
> x
[1] 1.224606e-16
> x = sin(pi)
> x
[1] 1.224606e-16
```

Neste material será dada preferência ao primeiro símbolo. Usuários pronunciam o comando dizendo que o objeto "recebe" (em inglês "gets") um certo valor. Por exemplo em `x <- sqrt(2)` dizemos que "x recebe a raiz quadrada de 2". Como pode ser esperado você pode fazer operações aritméticas com os objetos.

```
> y <- sqrt(5)      # uma nova variável chamada y
> y+x               # somando valores de x e y
[1] 2.236068
```

Note que ao atribuir um valor a um objeto o programa não imprime nada na tela. Digitando o nome do objeto o programa imprime seu conteúdo na tela. Digitando uma operação aritmética, sem atribuir o resultado a um objeto, faz com que o programa imprima o resultado na tela. Nomes de variáveis devem começar com uma letra e podem conter letras, números e pontos. Um fato importante é que o R distingue letras maiúsculas e minúsculas nos nomes dos objetos, por exemplo `dados`, `Dados` e `DADOS` serão interpretados como nomes de três objetos diferentes pela linguagem. *DICA*: tente atribuir nomes que tenham um significado lógico, relacionado ao trabalho e dados em questão. Isto facilita lidar com um grande número de objetos. Ter nomes como `a1` até `a20` pode causar confusão ... A seguir estão alguns exemplos válidos ...

```
> x <- 25
> x * sqrt(x) -> x1
> x2.1 <- sin(x1)
> xsq <- x2.1**2 + x2.2**2
```

... e alguns que NÃO são válidos:

```
> 99a <- 10
> a1 <- sqrt 10
> a-1 <- 99
> sqrt(x) <- 10
```

No primeiro caso o nome não começa com uma letra, o que é obrigatório, `a99` é um nome válido, mas `99a` não é. No segundo faltou um parentesis na função `sqrt`, o correto seria `sqrt(10)`. NO terceiro caso o hífen não é permitido, por ser o mesmo sinal usado em operações de subtração. O últio caso é um comando sem sentido.

É ainda desejável, e as vez crucial evitar ainda outros nomes que sejam de objetos do sistema (em geral funções, ou constantes tais como o número π) como, por exemplo:

```
 c q  s  t  C  D  F  I  T  diff  exp  log  mean  pi  range  rank  var
```

Nomes reservados: O R, como qualquer outra linguagem, possui nomes reservados, isto nomes que não podem ser utilizados para objetos por terem um significado especial na linguagem. São eles:

```
FALSE  Inf  NA  NaN  NULL TRUE
break  else  for  function  if  in  next  repeat  while
```

Valores especiais revisitados: Vimos anteriormente os valores especiais `NA`, `NaN` e `Inf`. Estes valores podem ser atribuídos a objetos ou elementos de um objeto e pode-se ainda testar a presença destes valores em objetos ou seus elementos.

No exemplo a seguir definimos um vetor de valores e verificamos que o objeto criado não contém nenhum destes valores especiais. Note neste exemplo o uso do caracter `!` que indica negação. As funções do tipo `is.*()` testam cada valor do vetor individualmente enquanto que `any()` verifica a presença de *algum* valor que satisfaça a condição e `all()` verifica se `todos` os valores satisfazem a condição.

```
> x <- c(23, 34, 12, 11, 34)
> is.na(x)
[1] FALSE FALSE FALSE FALSE FALSE
> !is.na(x)
[1] TRUE TRUE TRUE TRUE TRUE
```

```
> is.nan(x)
[1] FALSE FALSE FALSE FALSE FALSE
> is.finite(x)
[1] TRUE TRUE TRUE TRUE TRUE
> !is.finite(x)
[1] FALSE FALSE FALSE FALSE FALSE
> any(!is.finite(x))
[1] FALSE
> all(is.finite(x))
[1] TRUE
```

A seguir vamos substituir o terceiro dado `12` pelo código de dado faltante. Note ainda que operações envolvendo `NA` tipicamente retornam valor `NA` o que faz sentido uma vez que o valor não pode ser determinado, não está disponível.

```
> x[3] <- NA
> x
[1] 23 34 NA 11 34
> is.na(x)
[1] FALSE FALSE  TRUE FALSE FALSE
> any(is.na(x))
[1] TRUE
> all(is.na(x))
[1] FALSE
> x + 5
[1] 28 39 NA 16 39
> x/10
[1] 2.3 3.4  NA 1.1 3.4
> mean(x)
[1] NA
```

Agora vamos ver outros valores especiais.

```
> x1 <- (x - 34)/0
> x1
[1] -Inf  NaN   NA -Inf  NaN
> is.finite(x1)
[1] FALSE FALSE FALSE FALSE FALSE
> !is.finite(x1)
[1] TRUE TRUE TRUE TRUE TRUE
> is.nan(x1)
[1] FALSE  TRUE FALSE FALSE  TRUE
```

5 Miscelânia de funcionalidades do R

5.1 O R como calculadora

Podemos fazer algumas operações matemáticas simples utilizando o R. Vejamos alguns exemplos calculando as seguintes somas:

(a) $10^2 + 11^2 + \ldots + 20^2$
Para obter a resposta devemos

- criar uma sequência de números de 10 a 20
- elevar ao quadrado cada valor deste vetor
- somar os elementos do vetor

E estes passos correspondem aos seguintes comandos

```
> (10:20)
 [1] 10 11 12 13 14 15 16 17 18 19 20
> (10:20)^2
 [1] 100 121 144 169 196 225 256 289 324 361 400
> sum((10:20)^2)
[1] 2585
```

Note que só precisamos do último comando para obter a resposta, mas é sempre útil entender os comandos passo a passo!

(b) $\sqrt{\log(1)} + \sqrt{\log(10)} + \sqrt{\log(100)} + \ldots + \sqrt{\log(1000000)}$,
onde log é o logarítmo neperiano. Agora vamos resolver com apenas um comando:

```
> sum(sqrt(log(10^(0:6))))
[1] 16.4365
```

5.2 Gráficos de funções

Para ilustrar como podemos fazer gráficos de funções vamos considerar cada uma das funções a seguir cujos gráficos são mostrados nas Figuras 5.2 e 5.2.

(a) $f(x) = 1 - \frac{1}{x} sin(x)$ para $0 \leq x \leq 50$

(b) $f(x) = \frac{1}{\sqrt{50\pi}} \exp[-\frac{1}{50}(x - 100)^2]$ para $85 \leq x \leq 115$

A idéia básica é criar um vetor com valores das abscissas (valores de x) e calcular o valor da função (valores de $f(x)$) para cada elemento da função e depois fazer o gráfico unindo os pares de pontos. Vejamos os comandos para o primeiro exemplo.

```
> x1 <- seq(0, 50, l = 101)
> y1 <- 1 - (1/x1) * sin(x1)
> plot(x1, y1, type = "l")
```

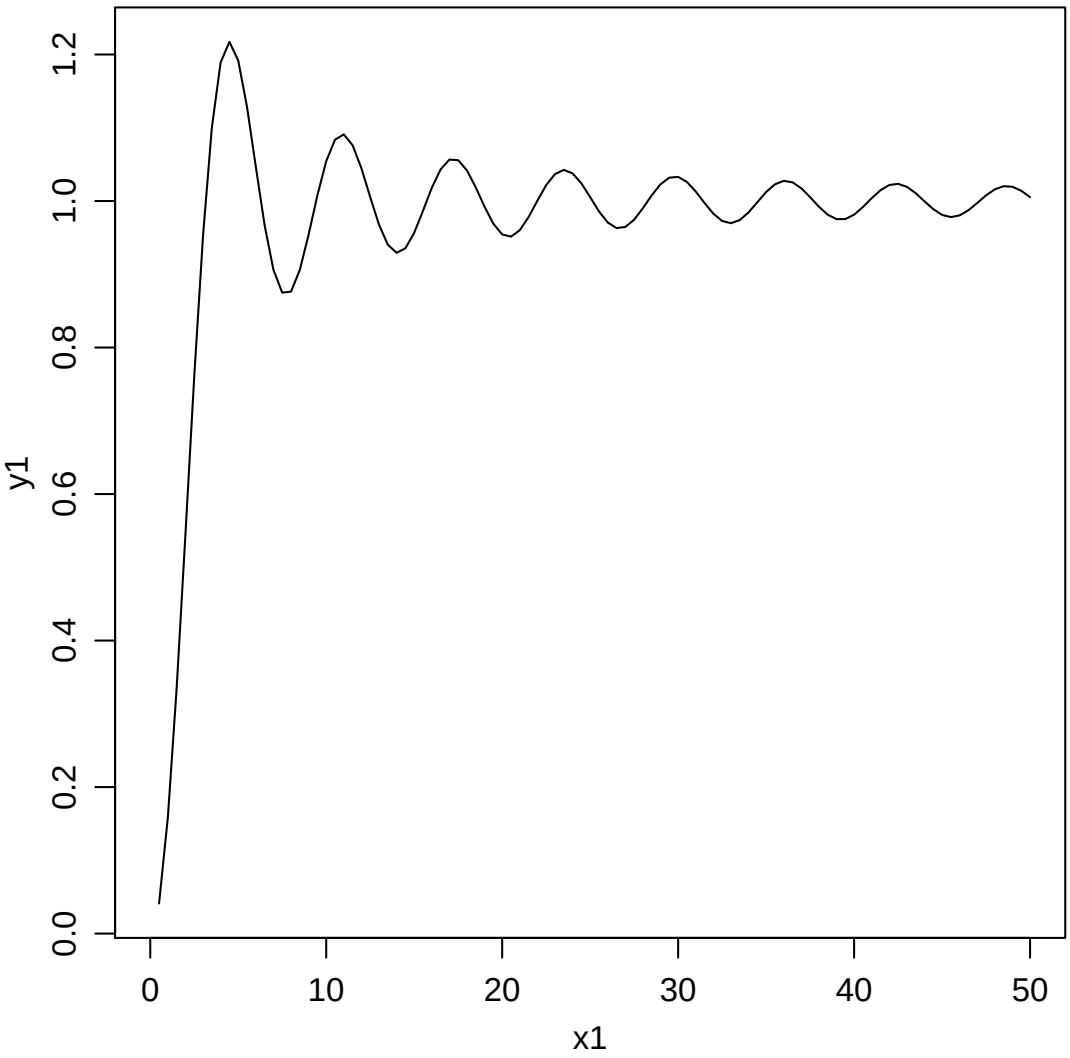


Figura 1: Gráfico da função dada em (a).

Note que este procedimento é o mesmo que aprendemos para fazer esboços de gráficos a mão em uma folha de papel!

Há ainda uma outra maneira de fazer isto no R utilizando `plot.function()` conforme pode ser visto no comando abaixo que nada mais faz que combinar os três comandos acima em apenas um.

```
> plot(function(x) 1 - (1/x) * sin(x), 0, 50)
```

Vejamos agora como obter o gráfico para a segunda função.

```
> x2 <- seq(80, 120, l = 101)
> y2 <- (1/sqrt(50 * pi)) * exp(-0.02 * (x2 - 100)^2)
> plot(x2, y2, type = "l")
```

Note ainda que esta função é a densidade da distribuição normal e o gráfico também poderia ser obtido com:

```
> y2 <- dnorm(x2, 100, 5)
> plot(x2, y2, type = "l")
```

ou ainda:

```
> plot(function(x) dnorm(x, 100, 5), 85, 115)
```

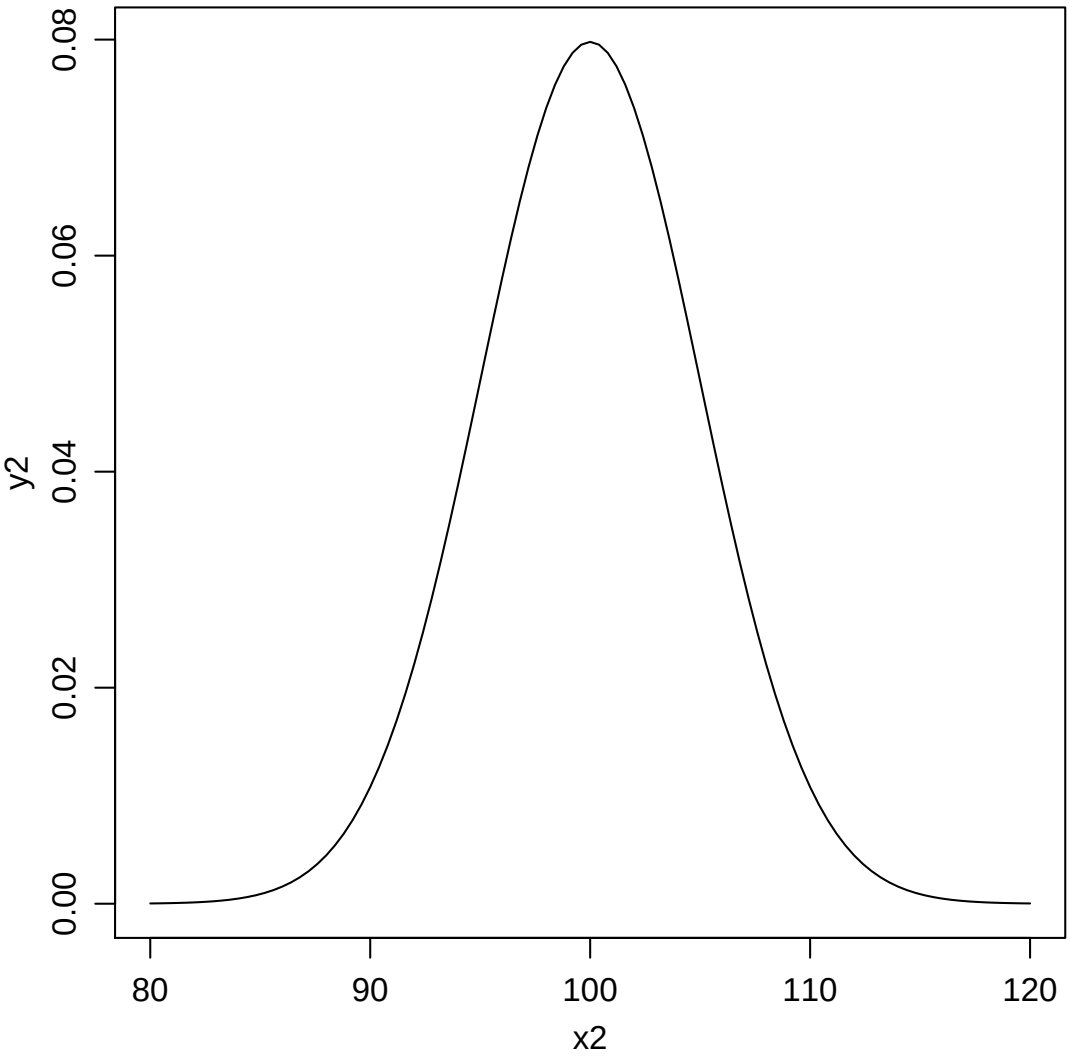


Figura 2: Gráfico da função dada em (b).

5.3 Integração numérica

A função `integrate()` é usada para integração numérica em uma dimensão. Como exemplo vamos considerar resolver a seguinte integral:

$$I = \int_{-3}^{3} x^2 dx. \tag{1}$$

Para resolver a integral devemos criar uma *função* no R com a expressão da função que vamos integrar e esta deve ser passada para `integrate()` conforme este exemplo:

```
> fx <- function(x) x^2
> integrate(fx, -3, 3)
18 with absolute error < 2e-13
```

A integral acima corresponde à área mostrada no gráfico da Figura 5.3. Esta figura é obtida com os seguinte comandos:

```
> x <- seq(-4, 4, l = 100)
> x2 <- x^2
```

```
> plot(x, x^2, ty = "l")
> x <- seq(-3, 3, l = 100)
> x2 <- x^2
> polygon(rbind(cbind(rev(x), 0), cbind(x, x2)), col = "gray")
```

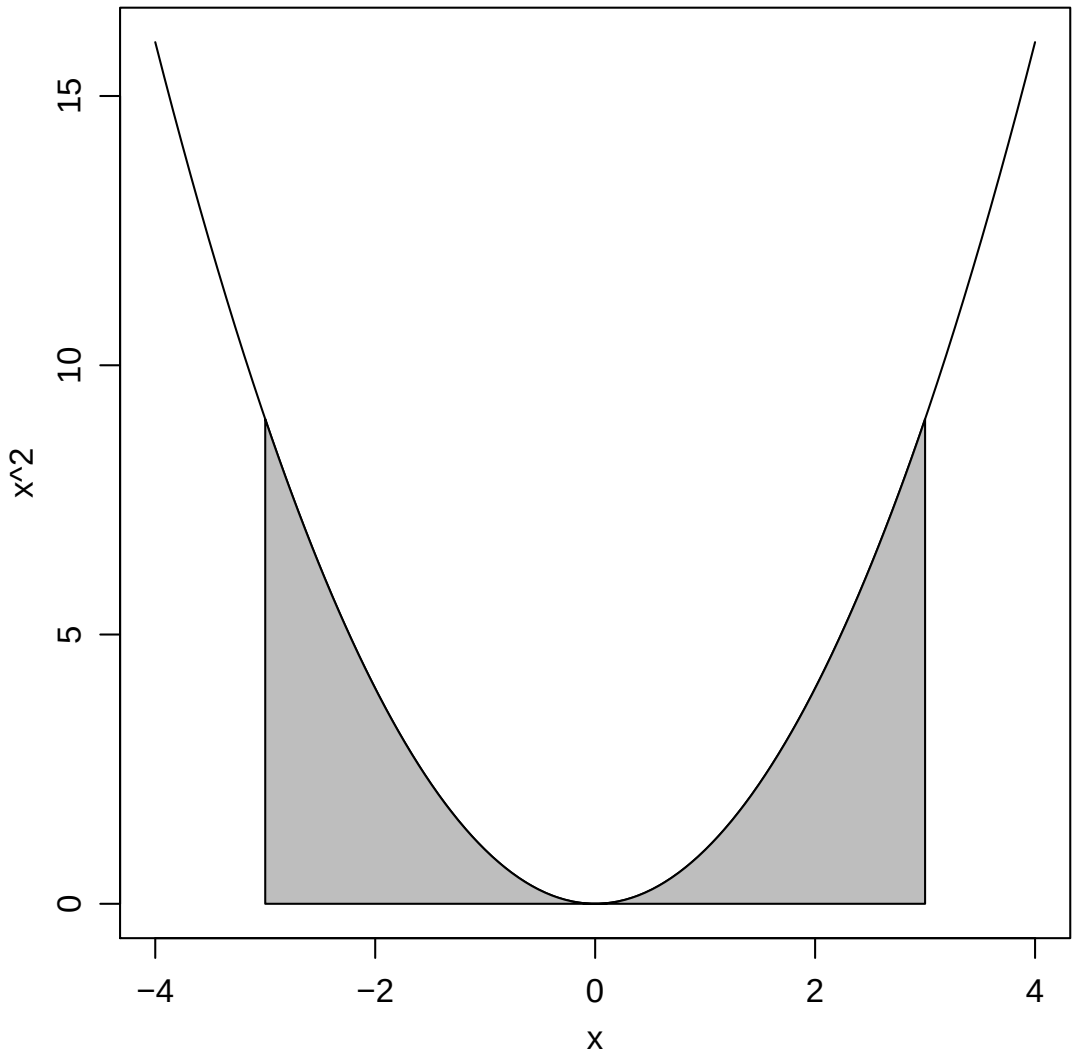


Figura 3: Gráfico onde a área indicada corresponde à integral definida na equação Equation 1.

Vejamos mais um exemplo. Sabemos que para distribuições contínuas de probabilidades a integral está associada a probabilidade em um intervalo. Seja $f(x)$ uma f.d.p. de uma variável contínua, então $P(a < X < b) = \int_a^b f(x)dx$. Por exemplo, seja X v.a. com distribuição $N(100, 81)$ e portanto $f(x) = \frac{1}{9\sqrt{2\pi}} \exp\{-\frac{1}{162}(x - 100)^2\}$. A probabilidade $P(85 < X < 105)$ pode ser calculada das três formas diferentes que irão retornar os memos resultados conforma mostrado a seguir.

```
> fx <- function(x) {
+     (1/(9 * sqrt(2 * pi))) * exp(-(1/162) * (x - 100)^2)
+ }
> integrate(fx, 85, 105)
0.6629523 with absolute error < 7.4e-15
> integrate(function(x) dnorm(x, 100, 9), 85, 105)
```

```
0.6629523 with absolute error < 7.4e-15
> pnorm(105, 100, 9) - pnorm(85, 100, 9)
[1] 0.6629523
```

5.4 Matemática simbólica no R

Embora o R seja um programa predominantemente para operações numéricas, é possivel obter alguns resultados simbólicos, em particular para expressões de derivadas que podem ser informadas para algorítimos de otimização numérica. A forma básica de utilização consiste em: (i) defina a expressão desejada dentro de `quote()`, (ii) use `D()` para obter a expressão da derivada desejada informando a expressão e o termo em relação ao qual deseja-se derivar a expressão, (iii) use `eval()` caso queira obter o valor numérico de uma determinada expressão. A documentação `help(D)` fornece mais detalhes. Vejamos um exemplo.

```
> f <- quote(sin(x^2 + log(y + z)))
> f
sin(x^2 + log(y + z))
> df.dx <- D(f, "x")
> df.dx
cos(x^2 + log(y + z)) * (2 * x)
> df.dy <- D(f, "y")
> df.dy
cos(x^2 + log(y + z)) * (1/(y + z))
> eval(f, list(x = 1, y = 2, z = 3))
[1] 0.5073913
> eval(df.dx, list(x = 1, y = 2, z = 3))
[1] -1.723432
```

Existem programas computacionais especializados em matemática simbólica dentre os quais destacam-se os projetos axiom e maxima.

- o programa axiom está disponível em
- o programa maxima está disponível em

5.5 Exercícios

1. Calcule o valor das expressões abaixo

   (a) Seja $x = (12, 11, 14, 15, 10, 11, 14, 11)$.
   Calcule $E = -n\lambda + (\sum_1^n x_i)\log(\lambda) - \sum_1^n \log(x_i!)$, onde n é o número de elementos do vetor x e $\lambda = 10$.
   Dica: o fatorial de um número pode ser obtido utilizando a função `prod`. Por exemplo o valor de 5! é obtido com o comando `prod(1:5)`.
   Há ainda uma outra forma usando a função Gama e lembrando que para a inteiro, $\Gamma(a+1) = a!$. Portanto podemos obter o valor de 5! com o comando `gamma(6)`.

   (b) $E = (\pi)^2 + (2\pi)^2 + (3\pi)^2 + ... + (10\pi)^2$

   (c) $E = \log(x+1) + \log(\frac{x+2}{2}) + \log(\frac{x+3}{3}) + \ldots + \log(\frac{x+20}{20})$, para $x = 10$

2. Obtenha o gráfico das seguintes funções:

   (a) $f(x) = x^{12}(1-x)^8$ para $0 < x < 1$

   (b) Para $\phi = 4$,

$$\rho(h) = \begin{cases} 1 - 1.5\frac{h}{\phi} + 0.5(\frac{h}{\phi})^3 \text{ , se } h < \phi \\ 0 \text{ , caso contrário} \end{cases}$$

3. Considerando as funções acima calcule as integrais a seguir e indique a área correspondente nos gráficos das funções.

   (a) $I_1 = \int_{0.2}^{0.6} f(x)dx$

   (b) $I_2 = \int_{1.5}^{3.5} \rho(h)dh$

4. Mostre os comandos para obter as seguintes sequências de números

   (a) 1 11 21 31 41 51 61 71 81 91

   (b) 1 1 2 2 2 2 2 3 3 3

   (c) 1.5 2.0 2.5 3.0 3.5 1.5 2.0 2.5 3.0 3.5 1.5 2.0 2.5 3.0 3.5

5. Escreva a sequência de comandos para obter um gráfico x versus y, onde x é um vetor com 100 valores igualmente espaçados no intervalo $[-1, 1]$ e $y = sin(x) * \exp(-x)$.

6. Escreva uma sequência de comandos no R para calcular a soma dos 80 primeiros termos das séries:

   (a) 1 + 1/32 + 1/52 + 1/72 + 1/92 + ...

   (b) 1 - 1/22 + 1/32 - 1/42 + 1/52 - 1/62 + ...

6 Tipos de objetos

Os tipos básicos de objetos do Rsão:

- vetores
- matrizes e arrays
- data-frames
- listas
- funções

Os quatro primeiros tipos são objetos que armazenam dados e que diferem entre si na forma da armazenar e operar com os dados. O último (função) é um tipo objeto especial que recebe algum "input"e produz um "output".

Experimente os comandos listados para se familiarizar com estas estruturas. Note que usamos as funções do tipo `is.*()` para testar se um objeto é de um determinado tipo. Estas funções são `is.vector()`, `is.matrix()`, `is.array()`, `is.data.frame()`, `is.list()`, `is.function()`.

6.1 Vetores

Vetores são o tipo básico e mais simples de objeto para armazenar dados no R. O R é uma linguagem vetorial, e portanto capaz de operar vetores e matrizes diretamente sem a necessidade de "loops", como por exemplo em códigos C e/ou Fortran.

Nos exemplo a seguir mostramos algumas operações com vetores. A função `c()` ("c" de concatenar) é usada para criar um vetor. Os colchetes `[ ]` são usados para indicar seleção de elementos. As funções `rep()`, `seq()` e o símbolo `":"` são usadas para facilitar a criação de vetores que tenham alguma lei de formação.

```
> x1 <- 10
> x1
[1] 10
> x2 <- c(1, 3, 6)
> x2
[1] 1 3 6
> x2[1]
[1] 1
> x2[2]
[1] 3
> length(x2)
[1] 3
> is.vector(x2)
[1] TRUE
> is.matrix(x2)
[1] FALSE
> is.numeric(x2)
[1] TRUE
```

```
> is.character(x2)
[1] FALSE
> x3 <- 1:10
> x3
 [1]  1  2  3  4  5  6  7  8  9 10
> x4 <- seq(0, 1, by = 0.1)
> x4
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> x4[x4 > 0.5]
[1] 0.6 0.7 0.8 0.9 1.0
> x4 > 0.5
 [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
> x5 <- seq(0, 1, len = 11)
> x5
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> x6 <- rep(1, 5)
> x6
[1] 1 1 1 1 1
> x7 <- rep(c(1, 2), c(3, 5))
> x7
[1] 1 1 1 2 2 2 2 2
> x8 <- rep(1:3, rep(5, 3))
> x8
 [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
```

Um escalar é um vetor de comprimento igual a 1. Os vetores podem ser compostos de números e caracteres ou apenas de um destes tipos. Portanto, adicionando um caracter a um vetor numérico este é transformado em um vetor alfanumérico.

```
> x2
[1] 1 3 6
> c("a", x2)
[1] "a" "1" "3" "6"
> c(x2, "a")
[1] "1" "3" "6" "a"
```

Diversas operações numéricas podem ser feitas sobre vetores. Uma característica importante da linguagem é a ”lei da reciclagem” que permite operações sobre vetores de tamanhos diferentes.

```
> x2
[1] 1 3 6
> x2 + 3
[1] 4 6 9
> x2 + 1:3
[1] 2 5 9
> x2 + 1:6
```

```
[1]  2  5  9  5  8 12
> (1:3) * x2
[1]  1  6 18
> x2/(1:6)
[1] 1.00 1.50 2.00 0.25 0.60 1.00
> x2^(1:3)
[1]   1   9 216
```

Vetores são uma estrutura de dados sobre a qual podemos aplicar funções como por exemplo as que fornecem medidas estatísticas.

```
> x9 <- round(rnorm(10, mean = 70, sd = 10))
> x9
 [1] 73 91 57 77 65 75 59 96 91 70
> sum(x9)
[1] 754
> mean(x9)
[1] 75.4
> var(x9)
[1] 184.9333
> min(x9)
[1] 57
> max(x9)
[1] 96
> summary(x9)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  57.00   66.25   74.00   75.40   87.50   96.00
> fivenum(x9)
[1] 57 65 74 91 96
```

Criando vetores com elementos repetidos As funções `rep()` e `seq()` do R são úteis para criar vetores de dados que seguem um certo padrão.
Clique aqui para ver um arquivo de dados.
vamos ver os comandos que podem ser usados para criar vetores para cada uma das três colunas iniciais deste arquivo.

A primeira coluna pode ser obtida com um dos dois comandos mostrados inicialmente, a seguir. Os demais reproduzem a segunda e terceira coluna do arquivo de dados.

```
> rep(1:4, each = 12)
> rep(1:4, rep(12, 4))
> rep(rep(1:3, each = 4), 4)
> rep(1:4, 12)
```

Vetores lógicos e seleção de elementos Como dito anteriormente os colchetes `[]` são usados para selecionar elementos de um vetor. No exemplo abaixo vemos como selecionar os 3 primeiros elementos do vetor `x9` criado anteriormente e depois os elementos em posição par no vetor (segundo, quarto, sexto, oitavo e décimo)

```
> x9[1:3]
[1] 73 91 57
> x9[2 * (1:5)]
[1] 91 77 75 96 70
```

Entretanto, a seleção de elementos é mais geral podendo atender a critérios definidos pelo usuário. A seguir mostramos que podemos criar um vetor lógico `ind.72` que indica se cada valor de `x9` é ou não maior que 72. O vetor pode ser ainda convertido para o formato de uma variável indicadora ("dummy").

```
> ind.72 <- x9 > 72
> ind.72
 [1]  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE  TRUE FALSE
> as.numeric(ind.72)
 [1] 1 1 0 1 0 1 0 1 1 0
> x10 <- x9[ind.72]
> x10
[1] 73 91 77 75 96 91
```

Vetores de caracteres Vetores de caracteres também são criados por `c()` com elementos entre aspas. Há também algumas funções para criação automática.

```
> nomes <- c("fulano", "beltrano", "cicrano")
> nomes
[1] "fulano"   "beltrano" "cicrano"
> let5 <- letters[1:5]
> let5
[1] "a" "b" "c" "d" "e"
> let10 <- LETTERS[11:20]
> let10
 [1] "K" "L" "M" "N" "O" "P" "Q" "R" "S" "T"
```

Uma função particularmente útil para criar vetores de caracteres é `paste()`. Examine os seguintes comandos.

```
> paste(nomes, 1:3)
[1] "fulano 1"   "beltrano 2" "cicrano 3"
> paste("fulano", 2)
[1] "fulano 2"
> paste("fulano", 2, sep = "")
[1] "fulano2"
> paste(letters[1:8], 2, sep = "")
```

```
[1] "a2" "b2" "c2" "d2" "e2" "f2" "g2" "h2"
```

Vejamos ainda mais um exemplo. Considere criar um vetor com elementos:

```
T1   T1   T1   T1   T2   T2   T2   T2   T3   T3   T3
```

```
> rep(paste("T", 1:3, sep = ""), c(4, 4, 3))
 [1] "T1" "T1" "T1" "T1" "T2" "T2" "T2" "T2" "T3" "T3" "T3"
```

Fatores Comentamos anteriormente que os vetores podem ser numéricos ou de caracteres. Entretanto há mais um tipo importante de objeto: os *fatores*. Por exemplo, ao criar um vetor de indicadores de "tratamentos" em uma análise de experimentos devemos declarar este vetor como um "fator". Portanto revisitando o exemplo visto anteriormente temos que uma forma mais adequada de usar o vetor como variável indicadora de tratamentos é defini-lo como um fator. Note que neste caso, diferentemente do anterior, são registrados os "níveis" (levels) do fator.

```
> factor(rep(paste("T", 1:3, sep = ""), c(4, 4, 3)))
 [1] T1 T1 T1 T1 T2 T2 T2 T2 T3 T3 T3
Levels: T1 T2 T3
```

É importante notar a diferença entre um *vetor de caracteres* e um vetor que seja *um fator* que são objetos de classes diferentes. O primeiro simplesmente guarda os seus elementos enquanto o segundo possui *atributos* que nesta caso incluem os níveis do fator. Nos comandos abaixo esta distinção fica mais clara onde um vetor é criado inicialmente como *numérico* e depois convertido para *fator*.

```
> estados <- c("PR", "SC", "RS")
> estados
[1] "PR" "SC" "RS"
> class(estados)
[1] "character"
> attributes(estados)
NULL
> estados <- factor(estados)
> estados
[1] PR SC RS
Levels: PR RS SC
> class(estados)
[1] "factor"
> attributes(estados)
$levels
[1] "PR" "RS" "SC"

$class
[1] "factor"
```

Um fato relevante a respeito da manipulação de fator é que uma seleção de parte dele que exclua um certo valor não exclui este valor dos atributos do vetor como no caso abaixo.

```
> estados.sel <- estados[-3]
> estados.sel
[1] PR SC
Levels: PR RS SC
```

Da mesma forma pode-se criar um vetor e definir para eles níveis, mesmos que estes níveis não estejam entre os elementos atualmente existentes no vetor. Note no exemplo abixo o que acontece com o valor `"MG"` em cada caso.

```
> est <- c("SC", "PR", "SC", "PR", "RS", "SP", "RS", "SP", "ES", "PR",
+     "RJ", "ES")
> est
 [1] "SC" "PR" "SC" "PR" "RS" "SP" "RS" "SP" "ES" "PR" "RJ" "ES"
> table(est)
est
ES PR RJ RS SC SP
 2  3  1  2  2  2
> sesul <- factor(est, levels = c("PR", "SC", "RS", "MG", "SP", "RJ",
+     "ES"))
> sesul
 [1] SC PR SC PR RS SP RS SP ES PR RJ ES
Levels: PR SC RS MG SP RJ ES
> table(sesul)
sesul
PR SC RS MG SP RJ ES
 3  2  2  0  2  1  2
```

Fatores Ordenados Um tipo especial de fator é dado pelos *fatores ordenados* que são fatores para os quais preserva-se a ordenação natural dos níveis. No próximo exemplo vemos um vetor inicialmente definido como de caracteres e a diferença entre defini-lo como não-ordenado ou ordenado. A ordenação segue a ordem alfabética a menos que uma ordenação diferente seja definida pelo usuário no argumento `levels`. Note ainda é pode-se usar duas funções diferentes para definir fatores ordenados: `factor(..., ord=T)` ou `ordered()`.

```
> grau <- c("medio", "baixo", "medio", "alto", "baixo", "baixo", "alto",
+     "medio", "alto", "medio")
> factor(grau)
 [1] medio baixo medio alto  baixo baixo alto  medio alto  medio
Levels: alto baixo medio
> factor(grau, ord = T)
 [1] medio baixo medio alto  baixo baixo alto  medio alto  medio
Levels: alto < baixo < medio
> ordered(grau)
 [1] medio baixo medio alto  baixo baixo alto  medio alto  medio
Levels: alto < baixo < medio
> factor(grau, ord = T, levels = c("baixo", "medio", "alto"))
```

```
 [1] medio baixo medio alto  baixo baixo alto  medio alto  medio
Levels: baixo < medio < alto
> ordered(grau, levels = c("baixo", "medio", "alto"))
 [1] medio baixo medio alto  baixo baixo alto  medio alto  medio
Levels: baixo < medio < alto
```

Mais algumas operações com vetores Considere o vetor `vec` obtido como se segue. As funções abaixo mostram como inverter a ordem dos elementos do vetor (`rev()`), ordenar os elementos ((`sort()`) e a posição de cada elemento no vetor ordenado e encontrar o "rank"dos elementos (`rank()`). As operações `%%` e `%%` fornecem, respectivamente, o resto e a parte inteira de uma divisão.

```
> vec <- round(rnorm(7, m = 70, sd = 10))
> vec
[1] 83 66 83 73 81 63 71
> rev(vec)
[1] 71 63 81 73 83 66 83
> sort(vec)
[1] 63 66 71 73 81 83 83
> order(vec)
[1] 6 2 7 4 5 1 3
> vec[order(vec)]
[1] 63 66 71 73 81 83 83
> rank(vec)
[1] 6.5 2.0 6.5 4.0 5.0 1.0 3.0
> vec%%5
[1] 3 1 3 3 1 3 1
> vec%/%5
[1] 16 13 16 14 16 12 14
```

A função which retorna a posição do(s) elemento(s) que obedece a certo critério.

```
> which(vec > 70)
[1] 1 3 4 5 7
> which.max(vec)
[1] 1
> which.min(vec)
[1] 6
```

Outra operação é a remoção de elementos de vetores através de índices negativos.

```
> vec
[1] 83 66 83 73 81 63 71
> vec[-5]
[1] 83 66 83 73 63 71
```

```
> vec[-(2:4)]
[1] 83 81 63 71
```

Para mais detalhes sobre vetores você pode consultar ainda as seguinte páginas:

- Vetores: `http://www.leg.ufpr.br/Rtutorial/vectors.html`
- Aritmética de vetores: `http://www.leg.ufpr.br/Rtutorial/vecarit.html`
- Caracteres e fatores: `http://www.leg.ufpr.br/Rtutorial/charfacs.html`
- Vetores Lógicos: `http://www.leg.ufpr.br/Rtutorial/logicals.html`
- Índices `http://www.leg.ufpr.br/Rtutorial/subscrip.html`

6.2 Matrizes

Matrizes são montadas a partir da reorganização de elementos de um vetor em linhas e colunas. Por "default" a matrix é preenchida por colunas e o argumento opcional `byrow=T` inverte este padrão. A seleção de elementos ou submatrizes é feita usando `[,]` sendo que antes da vírgula indica-se a(s) linha(s) e depois a(s) coluna(s) a serem selecionadas. Opcionalmente matrizes podem ter nomes associados às linhas e colunas ("rownames"e "colnames"). Cada um destes componentes da matrix é um vetor de nomes. Os comandos a seguir ilustram todas estas funcionalidades.

```
> m1 <- matrix(1:12, ncol = 3)
> m1
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
> matrix(1:12, ncol = 3, byrow = T)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
> length(m1)
[1] 12
> dim(m1)
[1] 4 3
> nrow(m1)
[1] 4
> ncol(m1)
[1] 3
> m1[1, 2]
[1] 5
> m1[2, 2]
[1] 6
```

```
> m1[, 2]
[1] 5 6 7 8
> m1[3, ]
[1]  3  7 11
> m1[1:2, 2:3]
     [,1] [,2]
[1,]    5    9
[2,]    6   10
> dimnames(m1)
NULL
> dimnames(m1) <- list(c("L1", "L2", "L3", "L4"), c("C1", "C2", "C3"))
> dimnames(m1)
[[1]]
[1] "L1" "L2" "L3" "L4"

[[2]]
[1] "C1" "C2" "C3"
> m1[c("L1", "L3"), ]
   C1 C2 C3
L1  1  5  9
L3  3  7 11
> m1[c(1, 3), ]
   C1 C2 C3
L1  1  5  9
L3  3  7 11
> m2 <- cbind(1:5, 6:10)
> m2
     [,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10
> m3 <- cbind(1:5, 6)
> m3
     [,1] [,2]
[1,]    1    6
[2,]    2    6
[3,]    3    6
[4,]    4    6
[5,]    5    6
```

Matrizes são muitas vezes utilizadas para armazenar frequências de cruzamentos entre variáveis. Desta forma é comum surgir a necessidade de obter os *totais marginais*, isto é a soma dos elementos das linhas e/ou colunas das matrizes, o que pode ser diretamente obtido com `margin.table()`. No caso de matrizes estas operação produz o mesmo resultado que outras funções conforme mostramos a seguir.

```
> margin.table(m1, margin = 1)
L1 L2 L3 L4
15 18 21 24
> apply(m1, 1, sum)
L1 L2 L3 L4
15 18 21 24
> rowSums(m1)
L1 L2 L3 L4
15 18 21 24
> margin.table(m1, margin = 2)
C1 C2 C3
10 26 42
> apply(m1, 2, sum)
C1 C2 C3
10 26 42
> colSums(m1)
C1 C2 C3
10 26 42
```

Operações com matrizes Operações com matrizes são feitas diretamente assim como no caso de vetores. A "lei da reciclagem"permanece válida. Existem diversas operações sobre matrizes e vamos apresentar apenas algumas aqui. Note que as operações abaixo são todas realizadas elemento a elemento.

```
> m4 <- matrix(1:6, nc = 3)
> m5 <- matrix(10 * (1:6), nc = 3)
> m4
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> m5
     [,1] [,2] [,3]
[1,]   10   30   50
[2,]   20   40   60
> m4 + m5
     [,1] [,2] [,3]
[1,]   11   33   55
[2,]   22   44   66
> m4 * m5
     [,1] [,2] [,3]
[1,]   10   90  250
[2,]   40  160  360
> m5 - m4
     [,1] [,2] [,3]
[1,]    9   27   45
[2,]   18   36   54
```

```
> m5/m4
     [,1] [,2] [,3]
[1,]   10   10   10
[2,]   10   10   10
```

A multiplicação de matrizes é feita usando o operador `%*%`. A função `t()` faz transposição e a inversão é obtida com `solve()`. O pacote **MASS** fornece `ginv()` para obtenção de inversa generalizada (inversa de Moore-Penrose)

```
> t(m4) %*% m5
     [,1] [,2] [,3]
[1,]   50  110  170
[2,]  110  250  390
[3,]  170  390  610
```

A função `solve()` na verdade é mais geral e fornece a solução de um sistema de equações lineares. Por exemplo, a solução do sistema:

$$\begin{cases} x + 3y - z & = 10 \\ 5x - 2y + z & = 15 \\ 2x + y - z & = 7 \end{cases}$$

pode ser obtida com:

```
> mat <- matrix(c(1, 5, 2, 3, -2, 1, -1, 1, -1), nc = 3)
> vec <- c(10, 15, 7)
> solve(mat, vec)
[1] 3.615385 3.307692 3.538462
```

Uma outra função muito útil para cálculos matriciais é `crossprod()` para produtos cruzados: `crossprod(X)` retorna $X\prime X$ `crossprod(X,Y)` retorna $X\prime Y$. Deve ser dada preferência a esta função sempre que possível pois é mais precisa e rápida do que o correspondente produto matricial com transposição do objeto do primeiro argumento.

Como exemplo vamos considerar as variáveis preditora e resposta com valores fornecidos na Tabela 6.2 e considere obter os coeficientes da regressão linear dados por:

$$\hat{\beta} = (X\prime X)^{-1} X\prime y \ , \tag{2}$$

onde X é a matrix com os valores da variável X acrescida de uma coluna de 1's e y são os valores da variável resposta.

Tabela 2: Valores da variável preditora e resposta para uma regressão linear simples.

1	2	3	4	5	6	7	8	9	10
13.4	16.6	15.8	17.3	18.5	22.1	23.2	35.9	31.3	39.4

Nos comandos abaixo mostramos como entrar com os dados e como obter os resultados de duas formas: (i) usando operações de matrizes de forma "ineficiente"e usando uma forma computacionalmente mais adequada de obter o mesmo resultado.

```
> X <- cbind(1, 1:10)
> y <- c(13.4, 16.6, 15.8, 17.3, 18.5, 22.1, 23.2, 35.9, 31.3, 39.4)
> solve(t(X) %*% X) %*% t(X) %*% y
     [,1]
[1,] 8.06
[2,] 2.78
> solve(crossprod(X), crossprod(X, y))
     [,1]
[1,] 8.06
[2,] 2.78
```

Notas:

1. existem formas ainda mais computacionalmente eficientes de obter o resultado acima no R, como por exemplo usando a decomposição **QR**, mas isto não será discutido neste ponto.

2. na prática para ajustar regressões no R o usuário não precisa fazer operações como a indicada pois já existem funções no R (neste caso `lm()`) que efetuam o ajuste.

Para mais detalhes sobre matrizes consulte a página:

- Matrizes

6.3 Arrays

O conceito de *array* generaliza a idéia de matrix. Enquanto em uma matrix os elementos são organizados em duas dimensões (linhas e colunas), em um array os elementos podem ser organizados em um número arbitrário de dimensões.

No R um *array* é definido utilizando a função array(). Defina um array com o comando a seguir e inspecione o objeto certificando-se que você entendeu como *arrays* são criados.

```
> ar1 <- array(1:24, dim = c(3, 4, 2))
> ar1
, , 1

     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

, , 2

     [,1] [,2] [,3] [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
[3,]   15   18   21   24
```

Examine agora os resultados dos seguintes comandos para selecionar e operar elementos do "array".

```
> ar1[, 2:3, ]
```

```
, , 1

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

, , 2

     [,1] [,2]
[1,]   16   19
[2,]   17   20
[3,]   18   21
> ar1[2, , 1]
[1]  2  5  8 11
> sum(ar1[, , 1])
[1] 78
> sum(ar1[1:2, , 1])
[1] 48
```

Podemos atribuir nomes às dimensões de um `array`.

```
> dimnames(ar1)
NULL
> dimnames(ar1) <- list(c("Baixo", "Médio", "Alto"), paste("col",
+     1:4, sep = ""), c("Masculino", "Feminino"))
```

Inspecione o "help" da função array (digite `help(array)`), rode e inspecione os exemplos contidos na documentação.

Veja agora um exemplo de dados já incluído no R no formato de array. Para "carregar" e visualizar os dados digite:

```
> data(Titanic)
> Titanic
, , Age = Child, Survived = No

      Sex
Class  Male Female
  1st     0      0
  2nd     0      0
  3rd    35     17
  Crew    0      0

, , Age = Adult, Survived = No

      Sex
Class  Male Female
  1st   118      4
  2nd   154     13
```

```
  3rd   387     89
  Crew  670      3

, , Age = Child, Survived = Yes

      Sex
Class  Male Female
  1st     5      1
  2nd    11     13
  3rd    13     14
  Crew    0      0

, , Age = Adult, Survived = Yes

      Sex
Class  Male Female
  1st    57    140
  2nd    14     80
  3rd    75     76
  Crew  192     20
```

Para obter maiores informações sobre estes dados digite:
help(Titanic)

Agora vamos responder às seguintes perguntas, mostrando os comandos do R utilizados sobre o *array* de dados.

1. quantas pessoas havia no total?

   ```
   > sum(Titanic)
   [1] 2201
   ```

2. quantas pessoas havia na tripulação (crew)?

   ```
   > sum(Titanic[4, , , ])
   [1] 885
   ```

3. quantas pessoas sobreviveram e quantas morreram?

   ```
   > apply(Titanic, 4, sum)
     No  Yes
   1490  711
   ```

4. quantas crianças sobreviveram?

   ```
   > sum(Titanic[, , 1, 2])
   [1] 57
   ```

5. quais as proporções de sobreviventes entre homens e mulheres?
   Vamos fazer por partes obtendo primeiro o número de homens e mulheres, depois dentre estes os que sobreviveram e depois obter as percentagens pedidas.

```
> apply(Titanic, 2, sum)
  Male Female
  1731    470
> apply(Titanic[, , , 2], 2, sum)
  Male Female
   367    344
> 100 * apply(Titanic[, , , 2], 2, sum)/apply(Titanic, 2, sum)
    Male   Female
21.20162 73.19149
```

Note-se ainda que assim como em matrizes, `margin.table()` poderia ser utilizada para obter os totais marginais para cada dimensão do *array* de dados, fornecendo uma maneira alternativa à alguns dos comandos mostrados acima.

```
> margin.table(Titanic, margin = 1)
Class
 1st  2nd  3rd Crew
 325  285  706  885
> margin.table(Titanic, margin = 2)
Sex
  Male Female
  1731    470
> margin.table(Titanic, margin = 3)
Age
Child Adult
  109  2092
> margin.table(Titanic, margin = 4)
Survived
  No  Yes
1490  711
```

Esta função admite ainda índices múltiplos que permitem outros resumos da tabela de dados. Por exemplo mostramos a seguir como obter o total de sobreviventes e não sobreviventes, separados por sexo e depois as porcentagens de sobreviventes para cada sexo.

```
> margin.table(Titanic, margin = c(2, 4))
        Survived
Sex        No  Yes
  Male   1364  367
  Female  126  344
> prop.table(margin.table(Titanic, margin = c(2, 4)), margin = 1)
        Survived
Sex             No       Yes
  Male   0.7879838 0.2120162
  Female 0.2680851 0.7319149
```

6.4 Data-frames

Vetores, matrizes e arrays forçam todos os elementos a serem do mesmo "tipo"i.e., ou numérico ou caracter. O "data-frame"é uma estrutura semelhante à uma matriz porém com cada coluna sendo tratada separadamente. Desta forma podemos ter colunas de valores numéricos e colunas de caracteres no mesmo objeto. Note entretanto que dentro de uma mesma coluna todos elementos ainda serão forçados a serem do mesmo tipo.

```
> d1 <- data.frame(X = 1:10, Y = c(51, 54, 61, 67, 68, 75, 77, 75,
+     80, 82))
> d1
    X  Y
1   1 51
2   2 54
3   3 61
4   4 67
5   5 68
6   6 75
7   7 77
8   8 75
9   9 80
10 10 82
> names(d1)
[1] "X" "Y"
> d1$X
 [1]  1  2  3  4  5  6  7  8  9 10
> d1$Y
 [1] 51 54 61 67 68 75 77 75 80 82
> plot(d1)
> plot(d1$X, d1$Y)
> d2 <- data.frame(Y = c(10 + rnorm(5, sd = 2), 16 + rnorm(5, sd = 2),
+     14 + rnorm(5, sd = 2)))
> d2$lev <- gl(3, 5)
> d2
          Y lev
1  14.35972   1
2  10.57823   1
3  11.19100   1
4  10.44532   1
5  12.50729   1
6  14.57557   2
7  16.36979   2
8  18.51120   2
9  14.20450   2
10 20.36193   2
11 14.59282   3
12 13.32687   3
13 14.12779   3
14 13.13010   3
15 13.35594   3
```

```
> by(d2$Y, d2$lev, summary)
INDICES: 1
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  10.45   10.58   11.19   11.82   12.51   14.36
------------------------------------------------------------
INDICES: 2
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  14.20   14.58   16.37   16.80   18.51   20.36
------------------------------------------------------------
INDICES: 3
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  13.13   13.33   13.36   13.71   14.13   14.59
> d3 <- expand.grid(1:3, 4:5)
> d3
  Var1 Var2
1    1    4
2    2    4
3    3    4
4    1    5
5    2    5
6    3    5
```

Na criação de data-frame `expand.grid()` pode ser muito útil gerando autimaticamente combinações de valores.

```
> expand.grid(1:3, 1:2)
  Var1 Var2
1    1    1
2    2    1
3    3    1
4    1    2
5    2    2
6    3    2
```

Para mais detalhes sobre data-frame consulte a página:

- Data-frames

6.5 Listas

Listas são estruturas genéricas e flexíveis que permitem armazenar diversos formatos em um único objeto.

```
> lis1 <- list(A = 1:10, B = "THIS IS A MESSAGE", C = matrix(1:9,
+     ncol = 3))
> lis1
$A
 [1]  1  2  3  4  5  6  7  8  9 10

$B
```

```
[1] "THIS IS A MESSAGE"

$C
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> lis2 <- lm(Y ~ X, data = d1)
> lis2

Call:
lm(formula = Y ~ X, data = d1)

Coefficients:
(Intercept)            X
     50.067        3.442

> is.list(lis2)
[1] TRUE
> class(lis2)
[1] "lm"
> summary(lis2)

Call:
lm(formula = Y ~ X, data = d1)

Residuals:
    Min      1Q  Median      3Q     Max
-2.9515 -2.5045 -0.2212  2.3076  4.2788

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  50.0667     1.9674   25.45 6.09e-09 ***
X             3.4424     0.3171   10.86 4.58e-06 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.88 on 8 degrees of freedom
Multiple R-squared: 0.9364,        Adjusted R-squared: 0.9285
F-statistic: 117.9 on 1 and 8 DF,  p-value: 4.579e-06

> anova(lis2)
Analysis of Variance Table

Response: Y
          Df Sum Sq Mean Sq F value    Pr(>F)
X          1 977.65  977.65  117.88 4.579e-06 ***
Residuals  8  66.35    8.29
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
> names(lis2)
```

```
 [1] "coefficients"  "residuals"     "effects"       "rank"          "fitted.values"
 [6] "assign"        "qr"            "df.residual"   "xlevels"       "call"
[11] "terms"         "model"
```

```
> lis2$pred
```

```
NULL
```

```
> lis2$res
```

```
         1          2          3          4          5          6          7
-2.5090909 -2.9515152  0.6060606  3.1636364  0.7212121  4.2787879  2.8363636
         8          9         10
-2.6060606 -1.0484848 -2.4909091
```

```
> plot(lis2)
> lis3 <- aov(Y ~ lev, data = d2)
> lis3
```

```
Call:
   aov(formula = Y ~ lev, data = d2)

Terms:
                     lev Residuals
Sum of Squares  63.42251  39.79740
Deg. of Freedom        2        12

Residual standard error: 1.821112
Estimated effects may be unbalanced
```

```
> summary(lis3)
```

```
            Df Sum Sq Mean Sq F value   Pr(>F)
lev          2 63.423  31.711  9.5618 0.003285 **
Residuals   12 39.797   3.316
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Uma lista é portanto uma coleção de objetos. Para listas há duas opções para se selecionar elementos: colchetes `[ ]` ou colchetes duplos `[[ ]]`. Entretanto os resultados retornados por cada um destes é diferente. Ou seja, o colchete simples (`[ ]`) retorna uma parte da lista, ou seja, retorna um objeto que ainda é uma lista. Já o colchete duplo (`[[ ]]`) retorna o objeto que está na posição indicada da lista. Examine o exemplo a seguir.

```
> lis1 <- list(nomes = c("Pedro", "Joao", "Maria"), mat = matrix(1:6,
+     nc = 2))
> lis1
```

```
$nomes
[1] "Pedro" "Joao"  "Maria"

$mat
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

```
> lis1[1]
```

```
$nomes
[1] "Pedro" "Joao"  "Maria"
> lis1[2]
$mat
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> lis1[[2]]
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

6.6 Funções

O conteúdo das funções podem ser vistos digitando o nome da função (sem os parênteses).

```
lm
glm
plot
plot.default
```

Entretanto isto não é disponível desta forma para todas as funções como por exemplo em `min`, `max`, `rnorm` e `lines` Nestes casos as funções não são escritas em linguagem R (em geral estão escritas em C) e para visualizar o conteúdo das funções você tem que examinar os arquivos do código fonte do R.

6.7 Que tipo de objeto eu tenho?

As funções do tipo `is.*()` mencionadas no início dasta sessão podem ser usadas para obter informações sobre a natureza de um objeto, o que pode sem muito útil quando se escreve funções em R. Entretanto são pouco práticas para determinar qual o tipo de um objeto e retornam apenas um valor lógico `TRUE` ou `FALSE`.

Uma função mais rica em detalhes é `str()` retorna informações sobre a *estrutura* do objeto. Nos exemplos a seguir vemos que a função informa sobre objecots que criamos anteriormente: `x1` é um vetor numérico, `estado` é um fator om três níveis, `ar1` é um *array*, `d1` é um *data.frame* com duas variáveis sendo uma delas de valores inteiros e a outra de valores numéricos e `lis1` é uma lista de dois elementos sendo o primeiro um vetor de caracteres e o segundo uma matrix de seis elementos e de dimensão 3×2.

```
> str(x1)
 num 10
> str(estados)
 Factor w/ 3 levels "PR","RS","SC": 1 3 2
> str(ar1)
 int [1:3, 1:4, 1:2] 1 2 3 4 5 6 7 8 9 10 ...
 - attr(*, "dimnames")=List of 3
  ..$ : chr [1:3] "Baixo" "Médio" "Alto"
  ..$ : chr [1:4] "col1" "col2" "col3" "col4"
  ..$ : chr [1:2] "Masculino" "Feminino"
```

```
> str(d1)
'data.frame':        10 obs. of  2 variables:
 $ X: int  1 2 3 4 5 6 7 8 9 10
 $ Y: num  51 54 61 67 68 75 77 75 80 82
> str(lis1)
List of 2
 $ nomes: chr [1:3] "Pedro" "Joao" "Maria"
 $ mat  : int [1:3, 1:2] 1 2 3 4 5 6
```

6.8 Exercícios

1. Mostrar comandos que podem ser usados para criar os objetos ou executar as instruções a seguir

   (a) o vetor

   ```
   [1] 4 8 2
   ```

   (b) selecionar o primeiro e terceiro elemento do vetor acima

   (c) 10

   (d) o vetor com a seqüência de valores

   ```
   [1] -3 -2 -1  0  1  2  3
   ```

   (e) o vetor com a seqüência de valores

   ```
   [1]  2.4  3.4  4.4  5.4  6.4  7.4  8.4  9.4 10.4
   ```

   (f) o vetor

   ```
    [1]  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39
   ```

   (g) o vetor

   ```
   [1]  1  3  5  7  9 11 14 17 20
   ```

   (h) o vetor de seqüência repetida

   ```
    [1] 1 1 1 2 2 2 3 3 3 4 4 4
   ```

   (i) o vetor de seqüência repetida

   ```
    [1] 4 4 4 3 3 3 2 2 2 1 1 1
   ```

   (j) o vetor de elementos repetidos

   ```
    [1] 1 2 3 1 2 3 1 2 3 1 2 3
   ```

   (k) a seqüência de valores

   ```
    [1]  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53
   [28] 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99
   ```

   (l) o vetor

   ```
    [1] 11 10  9  8  7  6  5  4  3  2  1
   ```

(m) o vetor alfanumérico

```
[1] "Parana"       "Sao Paulo"    "Minas Gerais"
```

(n) o vetor indicador de tratamentos

```
 [1] Trat_1 Trat_1 Trat_1 Trat_2 Trat_2 Trat_2 Trat_3 Trat_3 Trat_3 Trat_4 Trat_4
[12] Trat_4
Levels: Trat_1 Trat_2 Trat_3 Trat_4
```

(o) um vetor indicador de blocos

```
 [1] Bloco_1 Bloco_2 Bloco_3 Bloco_1 Bloco_2 Bloco_3 Bloco_1 Bloco_2 Bloco_3 Bloco_1
[11] Bloco_2 Bloco_3
Levels: Bloco_1 Bloco_2 Bloco_3
```

2. Mostre comando(s) para construir uma matriz 10×10 tal que as entradas são iguais a $i * j$, sendo i a linha e j a coluna.

3. Construa um data-frame com uma tabela com três colunas: x, x^2 e $\exp(x)$, com x variando de 0 a 50.

4. A função `sum(x)` retorna a soma dos elementos do vetor x. A expressão `z<-rep(x,10)` faz o vetor z igual a uma seqüência de 10 vetores x. Use estas e outras funções para calcular a soma dos 100 primeiros termos das séries:

   - (a) 1 + 1/2 + 1/3 + 1/4 + ...
   - (b) 1 + 1/22 + 1/42 + 1/62 + 1/82 + ...
   - (c) 1/(1+1/1!)2 + 1/(1+1/2!)2 + 1/(1+1/3!)2 + ...
   - (d) 1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + ...

5. Carregue o conjunto de dados com o comando
   `data(HairEyeColor)`

   e responda as seguintes perguntas fornecendo também o comando do R para obter a resposta:

   (a) Qual a proporção de homens e mulheres na amostra?

   (b) Quantos são os homens de cabelos pretos?

   (c) Quantas mulheres tem cabelos loiros?

   (d) Qual a proporção de homens e mulheres entre as pessoas ruivas?

   (e) Quantas pessoas tem olhos verdes?

6. Considere a tabela de freqüências a seguir. Entre com os dados usando o tipo de objeto adequado e mostre os comandos para responder as perguntas abaixo.

	Fumante		Não Fumante	
Idade	Masculino	Feminino	Masculino	Feminino
Menor que 20	50	30	55	41
20 a 40	39	28	31	30
Maior que 40	37	36	25	15

   (a) qual o número total de pessoas?

(b) quantos são os fumantes e os não fumantes?

(c) quantos são homens?

(d) quantas mulheres são não fumantes?

(e) quais as proporções de fumantes entre homens e mulheres?

7 Dados no *R*

Pode-se entrar com dados no R de diferentes formas. O formato mais adequado vai depender do tamanho do conjunto de dados, e se os dados já existem em outro formato para serem importados ou se serão digitados diretamente no R.

A seguir são descritas formas de entrada de dados com indicação de quando cada uma das formas deve ser usada. Os três primeiros casos são adequados para entrada de dados diretamente no R, os seguintes descreve como importar dados já disponíveis eletronicamentede um arquivo texto, em outro sistema ou no próprio R.

7.1 Entrando com dados diretamento no *R*

7.1.1 Definindo vetores

Podemos entrar com dados definindo vetores com o comando `c()` ("c" corresponde a *concatenate*) ou usando funções que criam vetores. Veja e experimente com os seguinte exemplos.

```
> a1 <- c(2, 5, 8)
> a1
[1] 2 5 8
> a2 <- c(23, 56, 34, 23, 12, 56)
> a2
[1] 23 56 34 23 12 56
```

Esta forma de entrada de dados é conveniente quando se tem um pequeno número de dados.

Quando os dados tem algum "padrão" tal como elementos repetidos, números sequenciais pode-se usar mecanismos do R para facilitar a entrada dos dados como vetores. Examine os seguintes exemplos.

```
> a3 <- 1:10
> a3
 [1]  1  2  3  4  5  6  7  8  9 10
> a4 <- (1:10) * 10
> a4
 [1]  10  20  30  40  50  60  70  80  90 100
> a5 <- rep(3, 5)
> a5
[1] 3 3 3 3 3
> a6 <- rep(c(5, 8), 3)
> a6
[1] 5 8 5 8 5 8
> a7 <- rep(c(5, 8), each = 3)
> a7
[1] 5 5 5 8 8 8
```

7.1.2 Usando a função `scan()`

Esta função lê dados diretamento do *console*, isto é, coloca o R em modo *prompt* onde o usuário deve digitar cada dado seguido da tecla <ENTER>. Para encerrar a entrada de dados basta digitar <ENTER> duas vezes consecutivas. Veja o seguinte resultado:

```
y <- scan()
#1: 11
#2: 24
#3: 35
#4: 29
#5: 39
#6: 47
#7:
#Read 6 items

> y
[1] 11 24 35 29 39 47
```

Este formato é mais ágil que o anterior e é conveniente para digitar vetores longos. Esta função pode também ser usada para ler dados de um arquivo ou conexão, aceitando inclusive endereços de URL's (endereços da web) o que iremos mencionar me mais detalhes mais adiante.

Corrigindo e/ou alterando dados Suponha que tenhamos digitado algum dado errado que desejamos corrigir. Por exemplo, suponha que o correto seja 25 no lugar de 35. Para corrigir basta selecionar a posição do dado atribuindo o valor correto

```
> y[3] <- 25
> y
[1] 11 24 25 29 39 47
```

Vejamos ainda um outro exemplo onde todo dado acima de 30 tem seu valor alterado para 30.

```
> y[y >= 30] <- 30
> y
[1] 11 24 25 29 30 30
```

7.1.3 Usando a função `edit()`

O comando `edit(data.frame())` abre uma planilha para digitação de dados que são armazanados como *data-frames*. Data-frames são o análogo no R à uma planilha.

Portanto digitando

```
a8 <- edit(data.frame())
```

será aberta uma planilha na qual os dados devem ser digitados. Quando terminar de entrar com os dados note que no canto superior direito da planilha existe um botão <QUIT>. Pressionando este botão a planilha será fechada e os dados serão gravados no objeto indicado (no exemplo acima no objeto `a8`).

Se voce precisar abrir novamente planilha com os dados, para fazer correções e/ou inserir mais dados use o comando `fix()`. No exemplo acima voce digitaria `fix(a8)`.

Esta forma de entrada de dados é adequada quando voce tem dados que não podem ser armazenados em um único vetor, por exemplo quando há dados de mais de uma variável para serem digitados.

7.2 Lendo dados de um arquivo texto

Se os dados já estão disponíveis em formato eletrônico, isto é, já foram digitados em outro programa, voce pode importar os dados para o R sem a necessidade de digitá-los novamente.

A forma mais fácil de fazer isto é usar dados em formato texto (arquivo do tipo ASCII). Por exemplo, se seus dados estão disponíveis em uma planilha eletrônica como EXCEL ou similar, voce pode na planilha escolher a opção <SALVAR COMO> e gravar os dados em um arquivo em formato texto.

No R usa-se `scan()` mencionada anteriormente, ou então a função mais flexível `read.table()` para ler os dados de um arquivo texto e armazenar no formato de uma *data-frame*.

Exemplo 1: Como primeiro exemplo considere importar para o R os dados deste arquivo texto. Clique no link para visualizar o arquivo. Agora copie o arquivo para sua área de trabalho (*working directory* do R). Para importar este arquivo usamos:

```
ex01 <- read.table("gam01.txt")
ex01
```

Exemplo 2: Como primeiro exemplo considere importar para o R os dados deste arquivo texto. Clique no link para visualizar o arquivo. Agora copie o arquivo para sua área de trabalho (*working directory* do R).

Note que este arquivo difere do anterior em um aspecto: os nomes das variáveis estão na primeira linha. Para que o R considere isto corretamente temos que informá-lo disto com o argumento `head=T`. Portanto para importar este arquivo usamos:

```
ex02 <- read.table("exemplo02.txt", head=T)
ex02
```

Exemplo 3: Como primeiro exemplo considere importar para o R os dados deste arquivo texto. Clique no link para visualizar o arquivo. Agora copie o arquivo para sua área de trabalho (*working directory* do R).

Note que este arquivo difere do primeiro em outros aspectos: além dos nomes das variáveis estarem na primeira linha, os campos agora não são mais separados por tabulação e sim por `:`. Alm disto os caracteres decimais estão separados por vírgula, sendo que o R usa ponto pois é um programa escrito em língua inglesa. Portanto para importar corretamente este arquivo usamos então os argumentos `sep` e `dec`:

```
ex03 <- read.table("dadosfic.csv", head=T, sep=":", dec=",")
ex03
```

Para maiores informações consulte a documentação desta função com `?read.table`.

Embora `read.table()` seja provavelmente a função mais utilizada existem outras que podem ser úteis e determinadas situações.

- `read.fwf()` é conveniente para ler "fixed width formats"
- `read.fortran()` é semelhante à anterior porém usando o estilo Fortran de especificação das colunas
- `scan()` é uma função internamente utilizadas por outras mas que também pode se usada diretamente pelo usuário.
- o mesmo ocorre para `read.csv()`, `read.delim()` e `read.delim2()`

Exemplo 4: As funções permitem ler ainda dados diretamente disponíveis na *web*. Por exemplo os dados do Exemplo 1 poderiam ser lidos diretamente com o comando a seguir, sem a necessidade de copiar primeiro os dados para algum local no computador do usuário.:

```
> read.table("http://www.leg.ufpr.br/~paulojus/dados/gam01.txt")
```

7.3 Importando dados de outros programas

É possível ler dados diretamente de outros formatos que não seja texto (ASCII). Isto em geral é mais eficiente e requer menos memória do que converter para formato texto. Há funções para importar dados diretamente de EpiInfo, Minitab, S-PLUS, SAS, SPSS, Stata, Systat e Octave. Além disto é comum surgir a necessidade de importar dados de planilhas eletrônicas. Muitas funções que permitem a importação de dados de outros programas são implementadas no pacote **foreign**.

```
> require(foreign)
[1] TRUE
```

A seguir listamos (mas não todas!) algumas destas funções

- `read.dbf()` para arquivos DBASE
- `read.epiinfo()` para arquivos `.REC` do Epi-Info
- `read.mtp()` para arquivos "Minitab Portable Worksheet"
- `read.S()` para arquivos do S-PLUS `restore.data()` para "dumps"do S-PLUS
- `read.spss()` para dados do SPSS
- `read.systat()`
- `read.dta()` para dados do STATA
- `read.octave()` para dados do OCTAVE (um clone do MATLAB)
- Para dados do SAS há ao menos duas alternativas:
  - O pacote **foreign** disponibiliza `read.xport()` para ler do formato TRANSPORT do SAS e `read.ssd()` pode escrever dados permanentes do SAS (`.ssd` ou `.sas7bdat`) no formato TRANSPORT, se o SAS estiver disponível no seu sistema e depois usa internamente `read.xport()` para ler os dados no R.
  - O pacote **Hmisc** disponibiliza `sas.get()` que também requer o SAS no sistema.

Para mais detalhes consulte a documentação de cada função e/ou o manual *R Data Import/Export*.

7.4 Carregando dados já disponíveis no R

Para carregar conjuntos de dados que são já disponibilizados com o R use o comando `data()`. Por exemplo, abaixo mostramos como carregar o conjunto `mtcars` que está no pacote **datasets** e depois como localizar e carregar o conjunto de dados `topo`.

```
> data(mtcars)
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
> find("topo")
character(0)
> require(MASS)
[1] TRUE
> data(topo)
> head(topo)
    x   y   z
1 0.3 6.1 870
2 1.4 6.2 793
3 2.4 6.1 755
4 3.6 6.2 690
5 5.7 6.2 800
6 1.6 5.2 800
```

O conjunto `mtcars` está no pacote **datasets** que é carregado automaticamente quando iniciamos o R, portanto os dados estão prontamente disponíveis. Ao carregar os dados é criado um objeto `mtcars` no seu "workspace".

Já o conjunto `topo` está no pacote **MASS** que não é automaticamente carregado ao iniciar o Re portanto deve ser carregado com `require()` para depois podermos acessar os dados.

A função `data()` pode ainda ser usada para listar os conjutos de dados disponíveis. A primeira chamada a seguir lista os conjuntos de dados dos pacotes carregados. A segunda lista os conjuntos de dados de um pacote específico (no exemplo do pacote **nlme**.

```
data()
data(package="nlme")
```

7.5 Acesso a planilhas e bancos de dados relacionais

É comum que dados estajam armazenados em planilhas eletrônicas tais como *MS-Excel* ou *OpenOffice Spreadsheet*. Nestes caso, embora seja possível exportar a partir destes aplicativos os dados para o formato texto para depois serem lidos no R, possivelmente com `read.table()`, pode ser necessário ou conveniente ler os dados diretamente destes formato. Vamos colocar aqui algumas opções para importar dados do MS-Excel para o R.

- O pacote **xlsReadWrite** implementa tal funcionalidade para arquivos do tipo *.xls* do MS-Excel. No momento que este material está sendo escrito esta pacote está implementado apenas para o sistema operacional Windows.

- Um outro pacote capaz de ler dados diretamente de planilhas é o **RODBC**. No ambiente windows a função `odbcConnectExcel()` está disponível para estabelecer a conexão. Suponha que voce possua um arquivo de uma planilha MS-Excel já no seu diretório (pasta) de trabalho do R chamado `planilha.xls`, que que esta planilha tenha os dados na *aba* `Planilha1`. Para importar os dados desta parte da planilha pode-se usar os comandos a seguir.

```
> require(RODBC)
> xlscon <- odbcConnectExcel("planilha.xls")
> dados1 <- sqlFetch(xlscon, "Planilha1")
> odbcClose(xlsConnect)
> head(dados1)
```

- Em sistemas onde a linguagem *Perl* está disponível e a estrutura de planilha é simples sem macros ou fórmulas, pode-se usar a função `xls2cvs` combinada com `read.csv()` ou `read.csv2()`, sendo esta última recomendada para dados com *vírgula* como caractere separados de decimais. O *Perl* é tipicamente instalado em sistemas Linux/Unix e também livremente disponível para outros sistemas operacionais.

```
> dados <- read.csv(pipe("xls2csv planilha.xls"))
> dados <- read.csv2(pipe("xls2csv planilha.xls"))
```

- O pacote **gdata** possui a função `read.xls()` que encapsula opções mencionadas anteriormente.

Estruturas de dados mais complexas são tipicamente armazenadas em acronymDBMS's (database management system) ou acronymRDBMS's (ralational database management system). Aguns exemplos são Oracle, Microsoft SQL server, MySQL, PostgreSQL, Microsoft Access, dentre outros. O R possuiu ferramentas implementadas em pacotes para acesso a estes sistemas gerenciadores.

Para mais detalhes consulte o manual *R Data Import/Export* e a documentação dos pacotes que implemental tal funcionalidade. Alguns deles disponíveis por ocasião da redação deste texto são: **RODBC**, **DBI**, **RMySQL**, **RPostgreSQL**, **ROracle**, **RNetCDF**, **RSQLite**, dentre outros.

8 Introdução à análise descritiva

8.1 Descrição univariada

Nesta sessão vamos ver alguns (mas não todos!) comandos do R para fazer uma análise descritiva de um conjunto de dados.

Uma boa forma de iniciar uma análise descritiva adequada é verificar os tipode de variáveis disponíveis. Variáveis podem ser classificadas da seguinte forma:

- **qualitativas (categóricas)**
  - nominais
  - ordinais
- **quantitativas**
  - discretas
  - contínuas

e podem ser resumidas por tabelas, gráficos e/ou medidas.

Vamos ilustrar estes conceitos com um conjunto de dados já incluído no R, o conjunto `mtcars` que descreve características de diferentes modelos de automóvel.

Primeiro vamos carregar e inspecionar os dados. Os comandos abaixo mostram como: (i) carregar os dados, (ii) visualizar todo o conjunto de dados, (iii) obter informações sobre os dados. Sendo este um conjunto de dados que vem junto com o R ele possui documentação que explica os dados. O mesmo acontece com todos os conjuntos de dados distribuídos com o R.

```
> data(mtcars)
> mtcars
> help(mtcars)
```

Digitando `mtcars` foi possível visualizar todos os dados. Entretanto em geral não queremos visualizar todos os dados, mas apenas uma parte deles. Os dois primeiros comandos abaixo mostram apenas a parte inicial do arquivo de dados. Os demais comandos mostram como verificar a dimensão do conjunto de dados (número de linhas e colunas) e como ver o nome das variáveis.

```
> mtcars[1:6, ]
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
> dim(mtcars)
[1] 32 11
> names(mtcars)
 [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
[11] "carb"
```

Vamos agora, por simplicidade, selecionar um subconjunto destes dados com apenas algumas das variáveis. Para isto vamos criar um objeto chamado `mtc` que contém apenas as variáveis desejadas. Para selecioná-las indicamos os números das colunas correspondentes à estas variáveis.

```
> mtc <- mtcars[, c(1, 2, 4, 6, 9, 10)]
> head(mtc)
                   mpg cyl  hp    wt am gear
Mazda RX4         21.0   6 110 2.620  1    4
Mazda RX4 Wag     21.0   6 110 2.875  1    4
Datsun 710        22.8   4  93 2.320  1    4
Hornet 4 Drive    21.4   6 110 3.215  0    3
Hornet Sportabout 18.7   8 175 3.440  0    3
Valiant           18.1   6 105 3.460  0    3
> names(mtc)
[1] "mpg"  "cyl"  "hp"   "wt"   "am"   "gear"
```

Vamos anexar o objeto para facilitar a digitação com o comando abaixo. O uso e sentido deste comando será explicado mais adiante.

```
> attach(mtc)
```

NOTA: em versões mais recentes do R foi introduzido a função `with()` que dispensa o uso de `attach()`, além de ser mais segura.

Vamos agora ver uma descrição da variável número de cilindros. Vamos fazer uma tabela de frequências absolutas e gráficos de barras e de setores.

```
> tcyl <- table(cyl)
> barplot(tcyl)
> pie(tcyl)
```

Para obter frequências relativas poderíamos usar os comandos abaixo. Note duas formas alternativas.

```
> tcyl <- 100 * table(cyl)/length(cyl)
> tcyl
cyl
    4     6     8
34.375 21.875 43.750
> prop.table(tcyl)
cyl
      4       6       8
0.34375 0.21875 0.43750
```

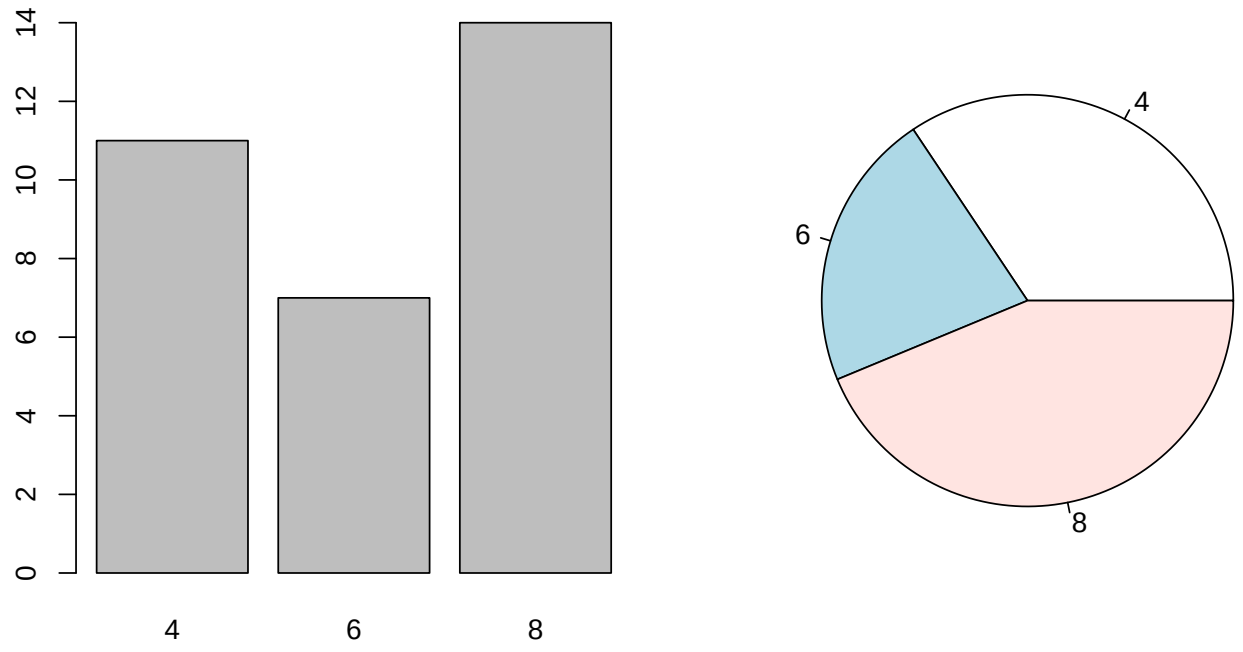


Figura 4: Dois tipos de gráficos para variável número de cilindros: de barras (esquerda) e de setores (direita).

Passando agora para uma variável quantitativa contínua vamos ver o comportamento da variável que mede o rendimento dos carros (em mpg – milhas por galão). Primeiro fazemos uma tabela de frequências, depois gráficos (histograma, box-plot e diagrama ramos-e-folhas) e finalmente obtemos algumas medidas que resumem os dados.

```
> table(cut(mpg, br = seq(10, 35, 5)))
> hist(mpg)
> boxplot(mpg)
> stem(mpg)
> summary(mpg)
```

8.2 Descrição bivariada

Vamos primeiro ver o resumo de duas variáveis categóricas: o tipo de marcha e o número de cilindros. Os comandos abaixo mostram como obter a tabela com o cruzamento destas variáveis e gráficos. Note e compare as saídas obtidas com cada um dos comandos a seguir.

```
> table(am, cyl)
   cyl
am   4  6  8
  0  3  4 12
  1  8  3  2
> prop.table(table(am, cyl))
   cyl
am        4       6       8
  0 0.09375 0.12500 0.37500
  1 0.25000 0.09375 0.06250
```

```
> prop.table(table(am, cyl), margin = 1)
   cyl
am          4         6         8
  0 0.1578947 0.2105263 0.6315789
  1 0.6153846 0.2307692 0.1538462
> prop.table(table(am, cyl), margin = 2)
   cyl
am          4         6         8
  0 0.2727273 0.5714286 0.8571429
  1 0.7272727 0.4285714 0.1428571
```

É possível também obter visualizações gráficas destes cruzamentos. Experimente os comandosa seguir, explore seus argumentos e observe os gráficos produzidos.

```
> plot(table(am, cyl))
> barplot(table(am, cyl), leg = T)
> barplot(table(am, cyl), beside = T, leg = T)
```

Agora vamos relacionar uma categórica (`tipo de câmbio`) com uma contínua (`rendimento`). O primeiro comando mostra como obter medidas resumo do rendimento para cada tipo de câmbio. A seguir obtemos as médias para cada tipo de câmbio de duas formas diferentes, a primeira selecionando os dados de cada um individualmente, e a segunda obtendo ambas as médias de uma só vez.

```
> tapply(mpg, am, summary)
$`0`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  10.40   14.95   17.30   17.15   19.20   24.40

$`1`
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  15.00   21.00   22.80   24.39   30.40   33.90
> m0 <- mean(mpg[am == 0])
> m0
[1] 17.14737
> m1 <- mean(mpg[am == 1])
> m1
[1] 24.39231
> m0m1 <- tapply(mpg, am, mean)
> m0m1
       0        1
17.14737 24.39231
```

A seguir são mostrados dois tipos de gráficos que podem ser obtidos para descrever o comportamento e associação destas variáveis.

```
> plot(am, mpg)
> points(c(0, 1), m0m1, cex = 2, col = 2, pch = 20)
```

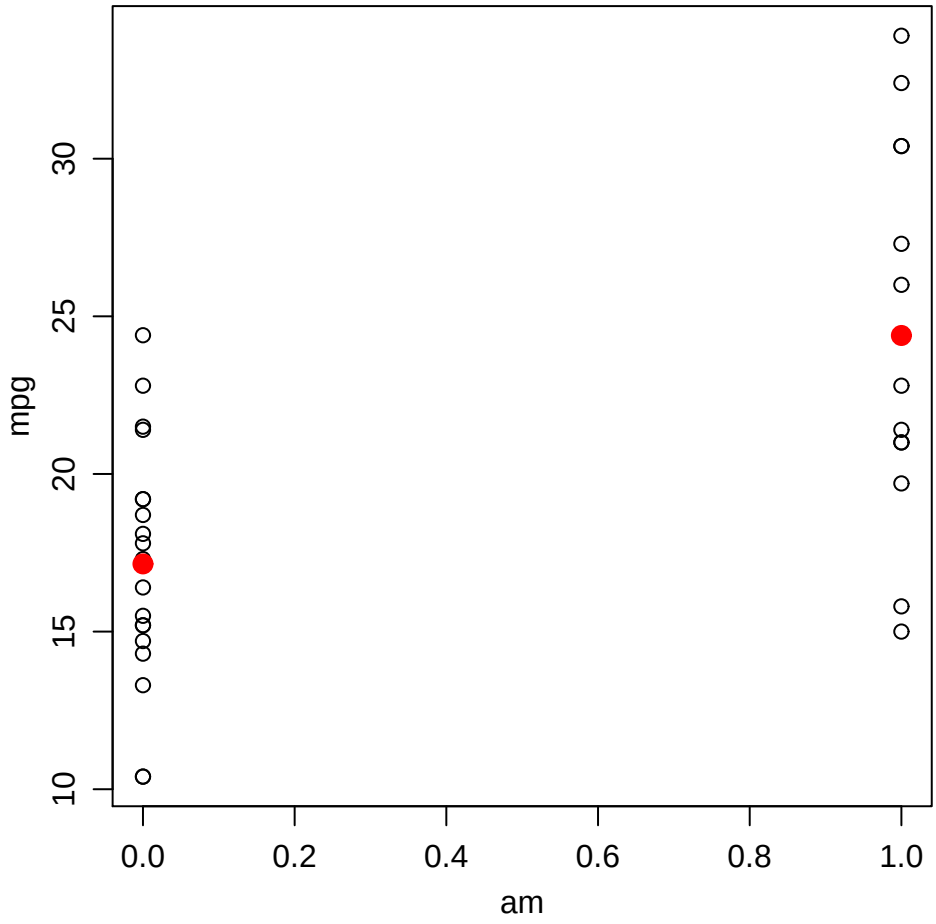


Figura 5: Dados de rendimentos para cada tipo de câmbio. Pontos vermelhos de tamanho maior indicam o rendimento médio para cada tipo de câmbio.

```
> par(mfrow = c(1, 2))
> by(hp, am, hist, main = "", xlim = c(50, 350))
> par(mfrow = c(1, 1))
```

Pode-se fazer um teste estatístico usando o teste t para comparar os redimentos de carros com diferentes tipos de câmbio e/ou com diferentes números de cilindros usando a análise de variância.

```
> t.test(mpg[am == 0], mpg[am == 1])
        Welch Two Sample t-test

data:  mpg[am == 0] and mpg[am == 1]
t = -3.7671, df = 18.332, p-value = 0.001374
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -11.280194  -3.209684
sample estimates:
mean of x mean of y
 17.14737  24.39231
> tapply(mpg, cyl, mean)
       4        6        8
26.66364 19.74286 15.10000
```

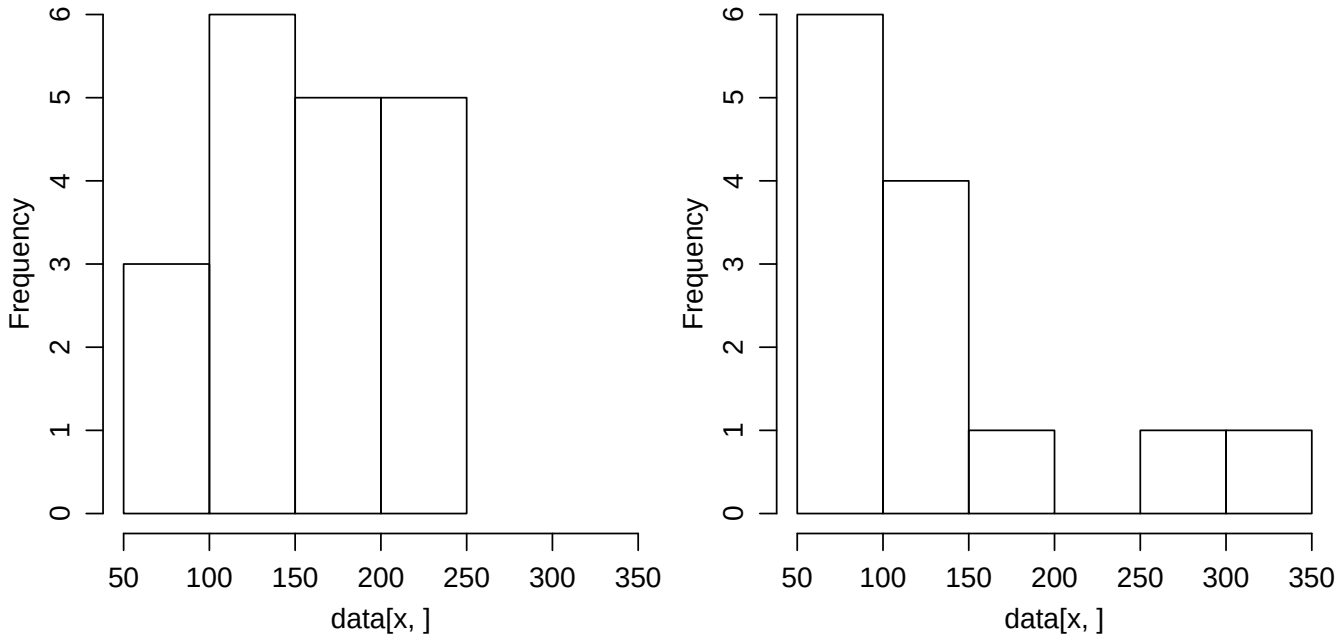


Figura 6: Histogramas dos rendimentos para cada tipo de câmbio.

```
> anova(aov(mpg ~ cyl))
Analysis of Variance Table

Response: mpg
          Df Sum Sq Mean Sq F value    Pr(>F)
cyl        1 817.71  817.71  79.561 6.113e-10 ***
Residuals 30 308.33   10.28
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Inspecione ainda o gráfico produzido pleo comando abaixo.

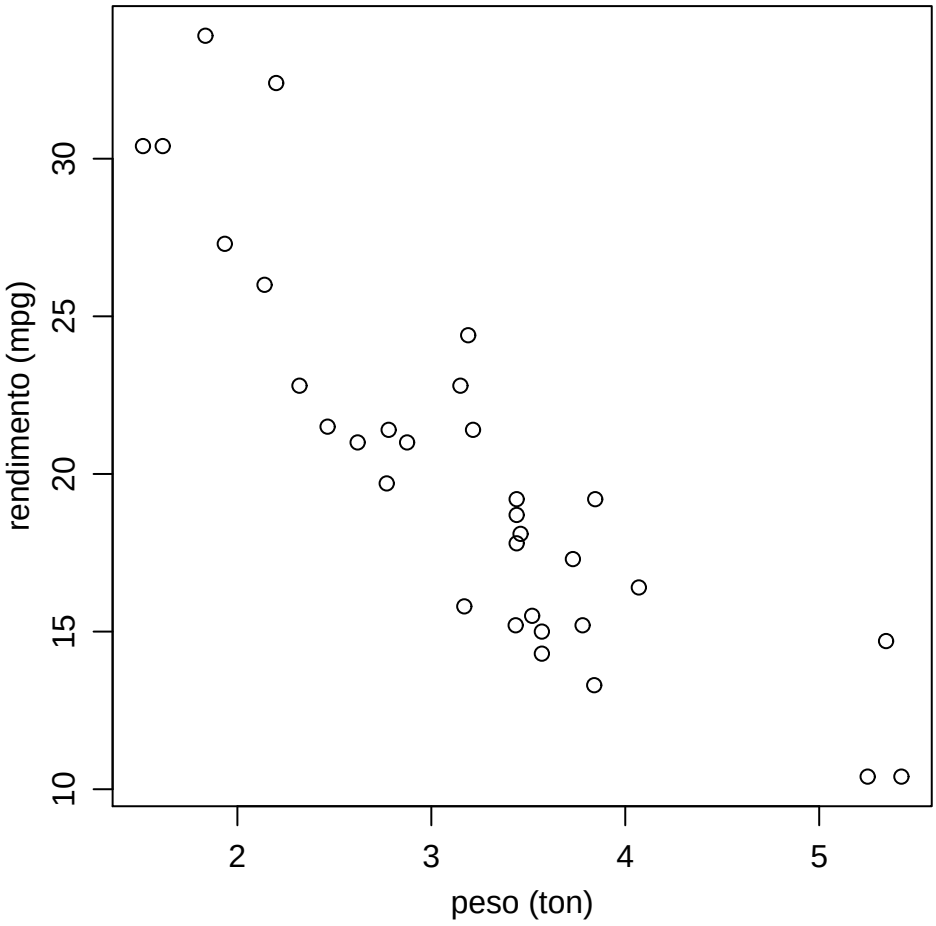
```
> plot(cyl,mpg)
```

Passamos agora para a relação entre duas variáveis contínuas (peso e rendimento) cuja relação pode ser investigada como se segue. O coeficiente de correlação linear de Pearson pode ser obtido com:

```
> cor(wt, mpg)
[1] -0.8676594
```

e o gráfico de rendimento *versus* peso com

```
> plot(wt, mpg, xlab = "peso (ton)", ylab = "rendimento (mpg)")
```



Podemos ainda usar recusos gráficos para visualizar três variáveis ao mesmo tempo. Inspecione os gráficos produzidos com os comandos a seguir.

```
> points(wt[cyl==4], mpg[cyl==4], col=2, pch=19)
> points(wt[cyl==6], mpg[cyl==6], col=3, pch=19)
> points(wt[cyl==8], mpg[cyl==8], col=4, pch=19)

> plot(wt, mpg, pch=21, bg=(2:4)[codes(factor(cyl))])
> plot(wt, mpg, pch=21, bg=(2:4)[codes(factor(am))])

> plot(hp, mpg)
> plot(hp, mpg, pch=21, bg=c(2,4)[codes(factor(am))])

> par(mfrow=c(1,2))
> plot(hp[am==0], mpg[am == 0])
> plot(hp[am==1], mpg[am == 1])
> par(mfrow=c(1,1))
```

8.3 Descrevendo um outro conjunto de dados

Vamos agora utilizar um outro conjunto de dados que já vem disponível com o R – o conjunto `airquality`.

Estes dados são medidas de: concentração de ozônio (*Ozone*), radiação solar (*Solar.R*), velocidade de vento (*Wind*) e temperatura (*Temp*) coletados diariamente (*Day*) por cinco meses (*Month*).

Primeiramente vamos carregar e visualisar os dados com os comandos:

```
> data(airquality)       # carrega os dados
> airquality             # mostra os dados
```

Vamos agora usar alguns comandos para "conhecer melhor" a estrutura dos dados. Nos comandos a seguir verificamos que`airquality` é um data-frame, obtemos os nomes das variáveis (colunas), a dimensão.

```
> is.data.frame(airquality)
[1] TRUE
> class(airquality)
[1] "data.frame"
> names(airquality)
[1] "Ozone"   "Solar.R" "Wind"    "Temp"    "Month"   "Day"
> dim(airquality)
[1] 153   6
```

Note ainda que o comando `help(airquality)` mostra a *documentação* destes deados, disponível por se tratar de um conjunto de dados já incluido no R. Bem, agora que conhecemos melhor o conjunto `airquality`, sabemos o número de dados, seu formato, o número de nome das variáveis podemos começar a analisá-los.

Veja por exemplo alguns comandos:

```
> summary(airquality)              # rápido sumário das variáveis
> summary(airquality[,1:4])        # rápido sumário apenas das 4 primeiras variáveis
> mean(airquality$Temp)            # média das temperaturas no período
> mean(airquality$Ozone)           # média do Ozone no período - note a resposta NA
> airquality$Ozone                 # a razão é que existem ``dados perdidos'' na variável Ozone
> mean(airquality$Ozone, na.rm=T) # média do Ozone no período - retirando valores perdidos
```

Note que os útimos três comandos são trabalhosos de serem digitados pois temos que digitar `airquality` a cada vez!
Mas há um mecanismo no R para facilitar isto: o *caminho de procura* ("search path"). Começe digitando e vendo s saída de:
`search()`

O programa vai mostrar o caminho de procura dos objetos. Ou seja, quando voce usa um nome do objeto o R vai procurar este objeto nos caminhos indicado, na ordem apresentada.

Pois bem, podemos "adicionar" um novo local neste caminho de procura e este novo local pode ser o nosso objeto `airquality`. Digite o seguinte e compare com o anterior:

```
> attach(airquality)   # anexando o objeto airquality no caminho de procura.
> search()             # mostra o caminho agora com o airquality incluído
> mean(Temp)           # e ... a digitação fica mais fácil e rápida !!!!
> mean(Ozone, na.rm=T) # pois com o airquality anexado o R acha as variáveis
```

NOTA: Para retirar o objeto do caminho de procura basta digitar `detach(airquality)`.

Bem, agora é com voce!

Reflita sobre os dados e use seus conhecimentos de estatística para fazer uma análise descritiva interessante destes dados.

Pense em questões relevantes e veja como usar medidas e gráficos para respondê-las. Use os comandos mostrados anteriormente. Por exemplo:

- as médias mensais variam entre si?
- como mostrar a evolução das variáveis no tempo?
- as variáveis estão relacionadas?
- etc, etc, etc

8.4 Outros dados disponíveis no R

Há vários conjuntos de dados incluídos no programa R como, por exemplo, o conjunto `mtcars`. Estes conjuntos são todos documentados, isto é, voce pode usar a função `help` para obter uma descrição dos dados. Para ver a lista de conjuntos de dados disponíveis digite `data()`. Por exemplo tente os seguintes comandos:

```
> data()
> data(women)
> women
> help(woman)
```

8.5 Exercícios

1. Experimente as funções `mean()`, `var()`, `sd()`, `median()`, `quantile()` nos dados mostrados anteriormente. Veja a documentação das funções e as opções de uso.
2. Faça uma análise descritiva adequada do conjunto de dados `women`.
3. Carregue o conjunto de dados `USArrests` com o comando `data(USArrests)`. Examine a sua documentação com `help(USArrests)` e responda as perguntas a seguir.
   (a) qual o número médio e mediano de cada um dos crimes?
   (b) encontre a mediana e quartis para cada crime.
   (c) encontre o número máximo e mínimo para cada crime.
   (d) faça um gráfico adequado para o número de assassinatos (*murder*).
   (e) faça um diagrama ramo-e-folhas para o número de estupros (*rape*).
   (f) verifique se há correlação entre os diferentes tipos de crime.
   (g) verifique se há correlação entre os crimes e a proporção de população urbana.
   (h) encontre os estados com maior e menor ocorrência de cada tipo de crime.
   (i) encontre os estados com maior e menor ocorrência per capta de cada tipo de crime.
   (j) encontre os estados com maior e menor ocorrência do total de crimes.

9 Análise descritiva

9.1 Descrição univariada

Nesta sessão vamos ver alguns (mas não todos!) comandos do R para fazer uma análise descritiva de um conjunto de dados.

Uma boa forma de iniciar uma análise descritiva adequada é verificar os tipode de variáveis disponíveis. Variáveis podem ser classificadas da seguinte forma:

- **qualitativas**
  - nominais
  - ordinais
- **quantitativas**
  - discretas
  - contínuas

e podem ser resumidas por tabelas, gráficos e/ou medidas.

9.2 Descrevendo o conjunto de dados "milsa" de Bussab & Morettin

O livro *Estatística Básica* de W. Bussab e P. Morettin traz no primeiro capítulo um conjunto de dados hipotético de atributos de 36 funcionários da companhia "Milsa". Os dados estão reproduzidos na tabela 9.2. Veja o livro para mais detalhes sobre este dados.

O que queremos aqui é ver como, no programa R:

- entrar com os dados
- fazer uma análise descritiva

Estes são dados no "estilo planilha", com variáveis de diferentes tipos: categóricas e numéricas (qualitativas e quantitativas). Portanto o formato ideal de armazenamento destes dados no R é o *data.frame*. Para entrar com estes dados no diretamente no R podemos usar o editor que vem com o programa. Para digitar rapidamente estes dados é mais fácil usar códigos para as variáveis categóricas. Desta forma, na coluna de estado civil vamos digitar o código 1 para *solteiro* e 2 para *casado*. Fazemos de maneira similar com as colunas *Grau de Instrução* e *Região de Procedência*. No comando a seguir invocamos o editor, entramos com os dados na janela que vai aparecer na sua tela e quanto saímos do editor (pressionando o botão `QUIT`) os dados ficam armazenados no objeto `milsa`. Após isto digitamos o nome do objeto (`milsa`) e podemos ver o conteúdo digitado, como mostra a tabela 9.2. Lembre-se que se voce precisar corrigir algo na digitação voce pode fazê-lo abrindo a planilha novamente com o comando `fix(milsa)`.

```
> milsa <- edit(data.frame())
> milsa
> fix(milsa)
```

Atenção: Note que além de digitar os dados na planilha digitamos também o nome que escolhemos para cada variável. Para isto basta, na planilha, clicar no nome da variável e escolher a opção `CHANGE NAME` e informar o novo nome da variável.

A planilha digitada como está ainda não está pronta. Precisamos informar para o programa que as variáveis `civil`, `instrucao` e `regiao`, NÃO são numéricas e sim categóricas. No R variáveis

Tabela 3: Dados de Bussab & Morettin

Funcionário	Est. Civil	Instrução	Nº Filhos	Salário	Ano	Mês	Região
1	solteiro	1o Grau	-	4.00	26	3	interior
2	casado	1o Grau	1	4.56	32	10	capital
3	casado	1o Grau	2	5.25	36	5	capital
4	solteiro	2o Grau	-	5.73	20	10	outro
5	solteiro	1o Grau	-	6.26	40	7	outro
6	casado	1o Grau	0	6.66	28	0	interior
7	solteiro	1o Grau	-	6.86	41	0	interior
8	solteiro	1o Grau	-	7.39	43	4	capital
9	casado	2o Grau	1	7.59	34	10	capital
10	solteiro	2o Grau	-	7.44	23	6	outro
11	casado	2o Grau	2	8.12	33	6	interior
12	solteiro	1o Grau	-	8.46	27	11	capital
13	solteiro	2o Grau	-	8.74	37	5	outro
14	casado	1o Grau	3	8.95	44	2	outro
15	casado	2o Grau	0	9.13	30	5	interior
16	solteiro	2o Grau	-	9.35	38	8	outro
17	casado	2o Grau	1	9.77	31	7	capital
18	casado	1o Grau	2	9.80	39	7	outro
19	solteiro	Superior	-	10.53	25	8	interior
20	solteiro	2o Grau	-	10.76	37	4	interior
21	casado	2o Grau	1	11.06	30	9	outro
22	solteiro	2o Grau	-	11.59	34	2	capital
23	solteiro	1o Grau	-	12.00	41	0	outro
24	casado	Superior	0	12.79	26	1	outro
25	casado	2o Grau	2	13.23	32	5	interior
26	casado	2o Grau	2	13.60	35	0	outro
27	solteiro	1o Grau	-	13.85	46	7	outro
28	casado	2o Grau	0	14.69	29	8	interior
29	casado	2o Grau	5	14.71	40	6	interior
30	casado	2o Grau	2	15.99	35	10	capital
31	solteiro	Superior	-	16.22	31	5	outro
32	casado	2o Grau	1	16.61	36	4	interior
33	casado	Superior	3	17.26	43	7	capital
34	solteiro	Superior	-	18.75	33	7	capital
35	casado	2o Grau	2	19.40	48	11	capital
36	casado	Superior	3	23.30	42	2	interior

Tabela 4: Dados digitados usando códigos para variáveis

	civil	instrucao	filhos	salario	ano	mes	regiao
1	1	1	NA	4.00	26	3	1
2	2	1	1	4.56	32	10	2
3	2	1	2	5.25	36	5	2
4	1	2	NA	5.73	20	10	3
5	1	1	NA	6.26	40	7	3
6	2	1	0	6.66	28	0	1
7	1	1	NA	6.86	41	0	1
8	1	1	NA	7.39	43	4	2
9	2	2	1	7.59	34	10	2
10	1	2	NA	7.44	23	6	3
11	2	2	2	8.12	33	6	1
12	1	1	NA	8.46	27	11	2
13	1	2	NA	8.74	37	5	3
14	2	1	3	8.95	44	2	3
15	2	2	0	9.13	30	5	1
16	1	2	NA	9.35	38	8	3
17	2	2	1	9.77	31	7	2
18	2	1	2	9.80	39	7	3
19	1	3	NA	10.53	25	8	1
20	1	2	NA	10.76	37	4	1
21	2	2	1	11.06	30	9	3
22	1	2	NA	11.59	34	2	2
23	1	1	NA	12.00	41	0	3
24	2	3	0	12.79	26	1	3
25	2	2	2	13.23	32	5	1
26	2	2	2	13.60	35	0	3
27	1	1	NA	13.85	46	7	3
28	2	2	0	14.69	29	8	1
29	2	2	5	14.71	40	6	1
30	2	2	2	15.99	35	10	2
31	1	3	NA	16.22	31	5	3
32	2	2	1	16.61	36	4	1
33	2	3	3	17.26	43	7	2
34	1	3	NA	18.75	33	7	2
35	2	2	2	19.40	48	11	2
36	2	3	3	23.30	42	2	1

categóricas são definidas usando o comando `factor()`, que vamos usar para redefinir nossas variáveis conforme os comandos a seguir. Inicialmente inspecionamos as primeiras linhas do conjunto de dados. A seguir redefinimos a variável `civil` com os *rótulos (labels)* solteiro e casado associados aos *níveis (levels)* 1 e 2. Para variável `instruçao` usamos o argumento adicional `ordered = TRUE` para indicar que é uma variável ordinal. Na variável `regiao` codificamos assim: 2=capital, 1=interior, 3=outro. Ao final inspecionamos as primeiras linhas do conjunto de dados digitando usando `head()`.

```
> head(milsa)
  funcionario civil instrucao filhos salario ano mes regiao
1           1     1         1     NA    4.00  26   3      1
2           2     2         1      1    4.56  32  10      2
3           3     2         1      2    5.25  36   5      2
4           4     1         2     NA    5.73  20  10      3
5           5     1         1     NA    6.26  40   7      3
6           6     2         1      0    6.66  28   0      1
> milsa$civil <- factor(milsa$civil, label = c("solteiro", "casado"),
+     levels = 1:2)
> milsa$instrucao <- factor(milsa$instrucao, label = c("1oGrau",
+     "2oGrau", "Superior"), lev = 1:3, ord = T)
> milsa$regiao <- factor(milsa$regiao, label = c("capital", "interior",
+     "outro"), lev = c(2, 1, 3))
> head(milsa)
  funcionario    civil instrucao filhos salario ano mes   regiao
1           1 solteiro    1oGrau     NA    4.00  26   3 interior
2           2   casado    1oGrau      1    4.56  32  10  capital
3           3   casado    1oGrau      2    5.25  36   5  capital
4           4 solteiro    2oGrau     NA    5.73  20  10    outro
5           5 solteiro    1oGrau     NA    6.26  40   7    outro
6           6   casado    1oGrau      0    6.66  28   0 interior
```

Em versões mais recentes do R foi introduzida a função `transform()` que pode ser usada alternativamente aos comandos mostrados acima para modificar ou gerar novas variáveis. Por exemplo, os comandos acima poderiam ser substituídos por:

```
> milsa <- transform(milsa, civil = factor(civil, label = c("solteiro",
+     "casado"), levels = 1:2), instrucao = factor(instrucao, label = c("1oGrau",
+     "2oGrau", "Superior"), lev = 1:3, ord = T), regiao = factor(regiao,
+     label = c("capital", "interior", "outro"), lev = c(2, 1,
+         3)))
```

Vamos ainda definir uma nova variável única `idade` a partir das variáveis `ano` e `mes` que foram digitadas. Para gerar a variável idade em anos fazemos:

```
> milsa <- transform(milsa, idade = ano + mes/12)
> milsa$idade
 [1] 26.25000 32.83333 36.41667 20.83333 40.58333 28.00000 41.00000 43.33333
 [9] 34.83333 23.50000 33.50000 27.91667 37.41667 44.16667 30.41667 38.66667
[17] 31.58333 39.58333 25.66667 37.33333 30.75000 34.16667 41.00000 26.08333
[25] 32.41667 35.00000 46.58333 29.66667 40.50000 35.83333 31.41667 36.33333
[33] 43.58333 33.58333 48.91667 42.16667
```

Uma outra forma de se obter o mesmo resultado seria:

```
> milsa$idade <- milsa$ano + milsa$mes/12
```

Agora que os dados estão prontos podemos começar a análise descritiva. A seguir mostramos como fazer análises descritivas uni e bi-variadas. Inspecione os comandos mostrados a seguir e os resultados por eleas produzidos. Sugerimos ainda que o leitor use o R para reproduzir os resultados mostrados no texto dos capítulos 1 a 3 do livro de Bussab & Morettin relacionados com este exemplo.

Inicialmente verificamos que o objeto `milsa` é um *data-frame*, usamos `names()` para ver os nomes das variáveis, e `dim()` para ver o número de linhas (36 indivíduos) e colunas (9 variáveis).

```
> is.data.frame(milsa)
[1] TRUE
> names(milsa)
[1] "funcionario" "civil"       "instrucao"   "filhos"      "salario"
[6] "ano"         "mes"         "regiao"      "idade"
> dim(milsa)
[1] 36  9
```

Como na sequência vamos fazer diversas análises com estes dados usaremos o command `attach()` para anexar o objeto ao caminho de procura para simplificar a digitação.

```
> attach(milsa)
```

NOTA: este comando deve ser digitado para que os comandos mostrados a seguir tenham efeito.

9.2.1 Análise Univariada

A análise univariada consiste basicamente em, para cada uma das variáveis individualmente:

- classificar a variável quanto a seu tipo: qualitativa (nominal ou ordinal) ou quantitativa (discreta ou contínua)
- obter tabela, gráfico e/ou medidas que resumam a variável

A partir destes resultados pode-se montar um resumo geral dos dados.

A seguir vamos mostrar como obter tabelas, gráficos e medidas com o R. Para isto vamos selecionar uma variável de cada tipo para que o leitor possa, por analogia, obter resultados para as demais.

Variável Qualitativa Nominal A variável `civil` é uma qualitativa nominal. Desta forma podemos obter: (i) uma tabela de frequências (absolutas e/ou relativas), (ii) um gráfico de setores, (iii) a "moda", i.e. o valor que ocorre com maior frequência.

Vamos primeiro listar os dados e checar se estao na forma de um *fator*, que é adequada para variáveis deste tipo.

```
> civil
 [1] solteiro casado   casado   solteiro solteiro casado   solteiro solteiro
 [9] casado   solteiro casado   solteiro solteiro casado   casado   solteiro
[17] casado   casado   solteiro solteiro casado   solteiro solteiro casado
[25] casado   casado   solteiro casado   casado   casado   solteiro casado
[33] casado   solteiro casado   casado
Levels: solteiro casado
```

```
> is.factor(civil)
[1] TRUE
```

A seguir obtemos frequências absolutas e relativas (note duas formas fiferentes de obter as frequências relativas. Note ainda que optamos por armazenar as frequências absolutas em um objeto que chamamos de `civil.tb`.

```
> civil.tb <- table(civil)
> civil.tb
civil
solteiro   casado
      16       20
> 100 * table(civil)/length(civil)
civil
solteiro   casado
44.44444 55.55556
> prop.table(civil.tb)
civil
 solteiro    casado
0.4444444 0.5555556
```

O gráfico de setores é adequado para representar esta variável conforme mostrado na Figura 9.2.1.

```
> pie(table(civil))
```

NOTA: Em computadores antigos e de baixa resolução gráfica (como por exemplo em alguns computadores da Sala A do LABEST/UFPR) o gráfico pode não aparecer de forma adequada devido limitação de memória da placa de vídeo. Se este for o caso use o comando mostrado a seguir ANTES de fazer o gráfico.

```
> X11(colortype = "pseudo.cube")
```

Finalmente encontramos a *moda* para esta variável cujo valor optamos por armazenar no objeto `civil.mo`.

```
> civil.mo <- names(civil.tb)[which.max(civil.tb)]
> civil.mo
[1] "casado"
```

Variável Qualitativa Ordinal Para exemplificar como obter análises para uma variável qualitativa ordinal vamos selecionar a variável `instrucao`.

```
> instrucao
 [1] 1oGrau   1oGrau   1oGrau   2oGrau   1oGrau   1oGrau   1oGrau   1oGrau
 [9] 2oGrau   2oGrau   2oGrau   1oGrau   2oGrau   1oGrau   2oGrau   2oGrau
[17] 2oGrau   1oGrau   Superior 2oGrau   2oGrau   2oGrau   1oGrau   Superior
[25] 2oGrau   2oGrau   1oGrau   2oGrau   2oGrau   2oGrau   Superior 2oGrau
[33] Superior Superior 2oGrau   Superior
Levels: 1oGrau < 2oGrau < Superior
```

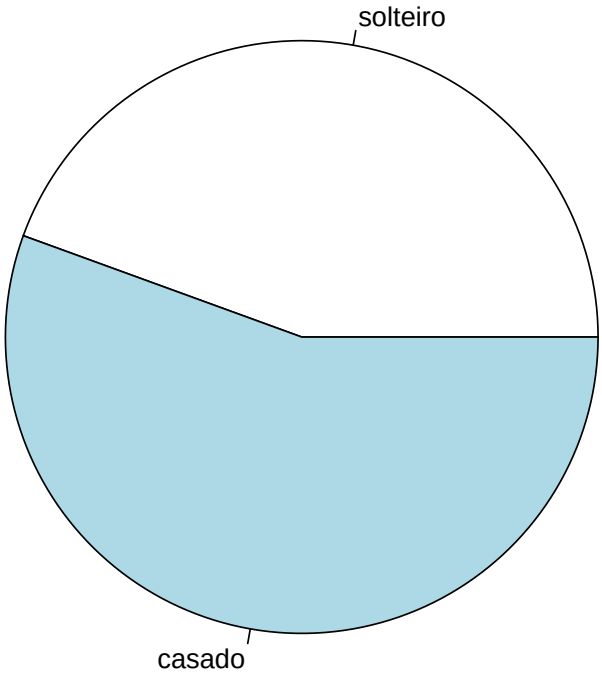


Figura 7: Gráfico de setores para variável `civil`.

```
> is.factor(instrucao)
[1] TRUE
```

As tabelas de frequências são obtidas de forma semelhante à mostrada anteriormente.

```
> instrucao.tb <- table(instrucao)
> instrucao.tb
instrucao
  1oGrau   2oGrau Superior
      12       18        6
> prop.table(instrucao.tb)
instrucao
   1oGrau    2oGrau  Superior
0.3333333 0.5000000 0.1666667
```

O gráfico de setores não é adequado para este tipo de variável por não expressar a ordem dos possíveis valores. Usamos então um gráfico de barras conforma mostrado na Figura 9.2.1.

```
> barplot(instrucao.tb)
```

Para uma variável ordinal, além da moda podemos também calcular outras medidas, tais como a mediana conforme exemplificado a seguir. Note que o comando `median()` não funciona com variáveis não numéricas e por isto usamos o comando seguinte.

```
> instrucao.mo <- names(instrucao.tb)[which.max(instrucao.tb)]
> instrucao.mo
```

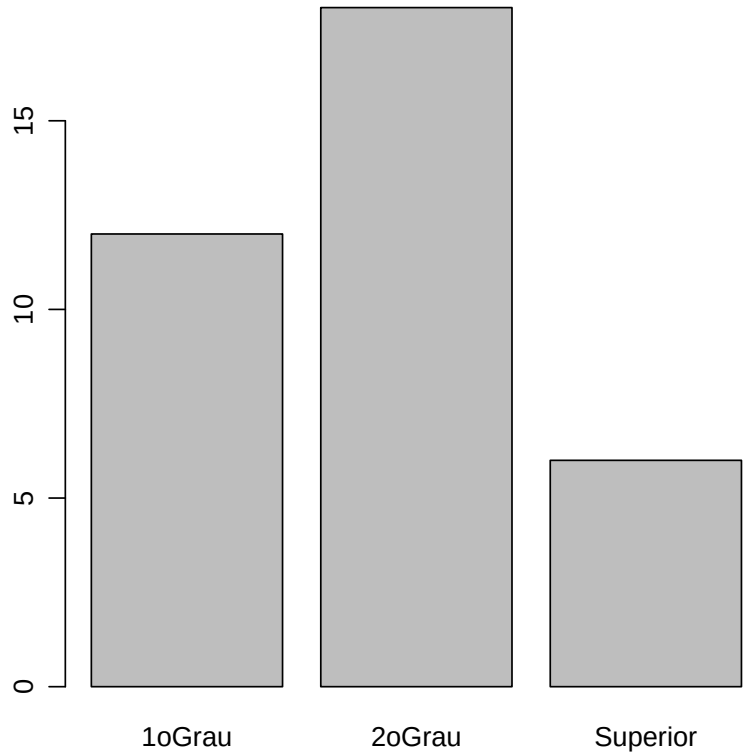


Figura 8: Gráfico de barras para variável `instrucao`.

```
[1] "2oGrau"
> median(as.numeric(instrucao))
[1] 2
> levels(milsa$instrucao)[median(as.numeric(milsa$instrucao))]
[1] "2oGrau"
```

Variável quantitativa discreta Vamos agora usar a variável `filhos` (número de filhos) para ilustrar algumas análises que podem ser feitas com uma quantitativa discreta. Note que esta deve ser uma variável numérica, e não um fator.

```
> filhos
 [1] NA  1  2 NA NA  0 NA NA  1 NA  2 NA NA  3  0 NA  1  2 NA NA  1 NA NA  0  2
[26]  2 NA  0  5  2 NA  1  3 NA  2  3
> is.factor(filhos)
[1] FALSE
> is.numeric(filhos)
[1] TRUE
```

Frequências absolutas e relativas são obtidas como anteriormente.

```
> filhos.tb <- table(filhos)
> filhos.tb
```

```
filhos
0 1 2 3 5
4 5 7 3 1
> filhos.tbr <- prop.table(filhos.tb)
> filhos.tbr
filhos
   0    1    2    3    5
0.20 0.25 0.35 0.15 0.05
```

O gráfico adequado para frequências absolutas de uma variável discreta é mostrado na Figura 9.2.1 o obtido com os comandos a seguir.

```
> plot(filhos.tb)
```

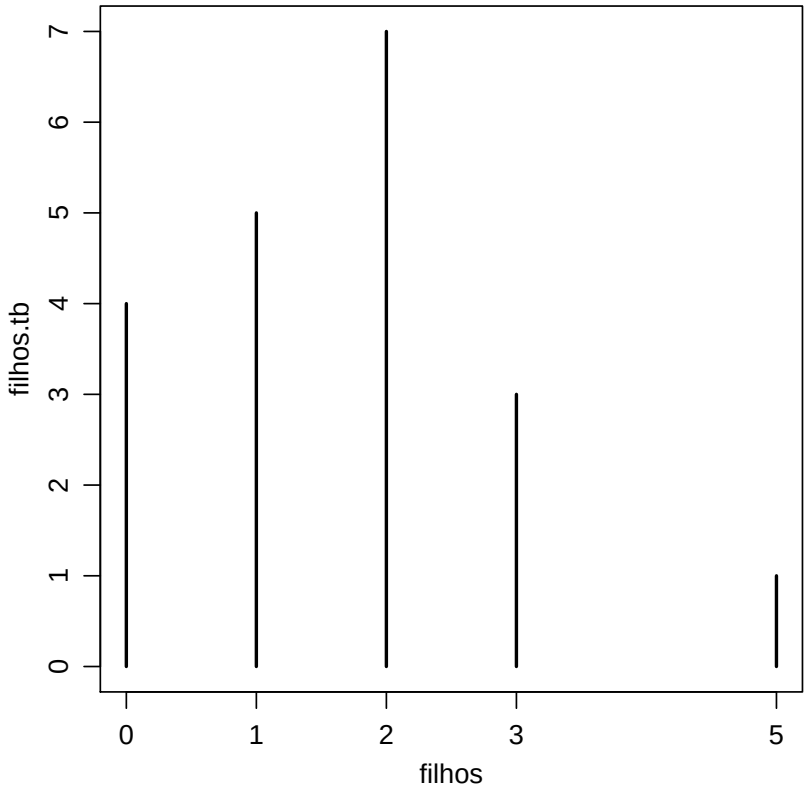


Figura 9: Gráfico de frequências absolutas para variável `filhos`.

Outra possibilidade seria fazer gráficos de frequências relativas e de prequências acumuladas conforme mostrado na Figura 9.2.1.

```
> plot(filhos.tbr)
> filhos.fac <- cumsum(filhos.tbr)
> filhos.fac
> plot(filhos.fac, type = "S")
```

Sendo a variável numérica há uma maior diversidade de medidas estatísticas que podem ser calculadas.

A seguir mostramos como obter algumas medidas de posição: moda, mediana, média e média aparada. Note que o argumento `na.rm=T` é necessário porque não há informação sobre número de

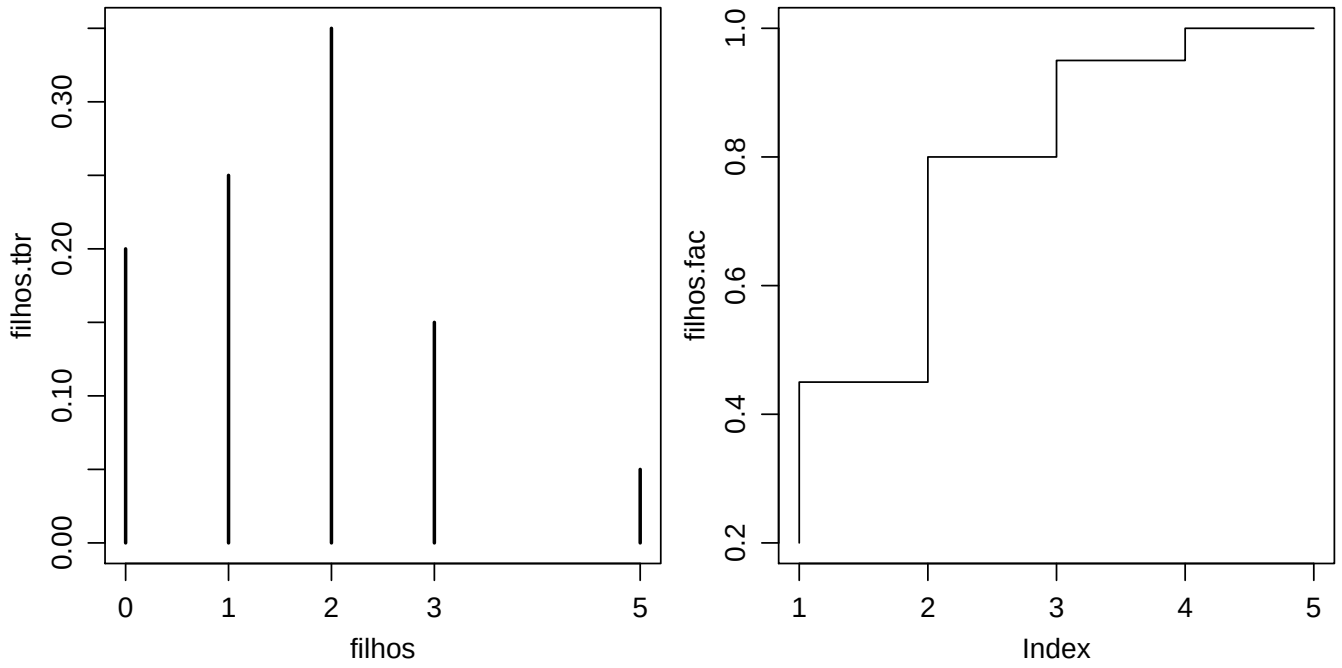


Figura 10: Gráfico de frequências relativas (esquerda) e frequências acumuladas para variável `filhos`.

filhos para alguns indivíduos. O argumento `trim=0.1` indica uma média aparada onde foram retirados 10% dos menores e 10% dos maiores dados. Ao final mostramos como obter os quartis, mínimo e máximo.

```
> filhos.mo <- names(filhos.tb)[which.max(filhos.tb)]
> filhos.mo
[1] "2"
> filhos.md <- median(filhos, na.rm = T)
> filhos.md
[1] 2
> filhos.me <- mean(filhos, na.rm = T)
> filhos.me
[1] 1.65
> filhos.me <- mean(filhos, trim = 0.1, na.rm = T)
> filhos.me
[1] 1.5625
> filhos.qt <- quantile(filhos, na.rm = T)
```

Passando agora para medidas de dispersão vejamos como obter máximo e mínimo daí a amplitude, variância e desvio padrão, coeficiente de variação. Depois obtemos os quartis e daí a amplitude interquartílica.

```
> range(filhos, na.rm = T)
[1] 0 5
> filhos.A <- diff(range(filhos, na.rm = T))
> filhos.A
[1] 5
```

```
> var(filhos, na.rm = T)
[1] 1.607895
> filhos.dp <- sd(filhos, na.rm = T)
> filhos.dp
[1] 1.268028
> filhos.cv <- 100 * filhos.dp/filhos.me
> filhos.cv
[1] 81.15379
> filhos.qt <- quantile(filhos, na.rm = T)
> filhos.ai <- filhos.qt[4] - filhos.qt[2]
> filhos.ai
75%
  1
```

Finalmente, notamos que há comandos para se obter várias medidas de uma só vez. Inspecione os resultados dos comandos abaixo.

```
> summary(filhos)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
   0.00    1.00    2.00    1.65    2.00    5.00   16.00
> fivenum(filhos)
[1] 0 1 2 2 5
```

Variável quantitativa Contínua Para concluir os exemplos para análise univariada vamos considerar a variável quantitativa contínua `salario`. Começamos mostrando os valores da variável e verificando o seu tipo no R.

```
> salario
 [1]  4.00  4.56  5.25  5.73  6.26  6.66  6.86  7.39  7.59  7.44  8.12  8.46
[13]  8.74  8.95  9.13  9.35  9.77  9.80 10.53 10.76 11.06 11.59 12.00 12.79
[25] 13.23 13.60 13.85 14.69 14.71 15.99 16.22 16.61 17.26 18.75 19.40 23.30
> is.factor(salario)
[1] FALSE
> is.numeric(salario)
[1] TRUE
```

Para se fazer uma tabela de frequências de uma contínua é preciso primeiro agrupar os dados em classes. Nos comandos mostrados a seguir verificamos inicialmente os valores máximo e mínimo dos dados, depois usamos o critério de Sturges para definir o número de classes, usamos `cut()` para agrupar os dados em classes e finalmente obtemos as frequências absolotas e relativas.

```
> range(salario)
[1]  4.0 23.3
> nclass.Sturges(salario)
[1] 7
> args(cut)
```

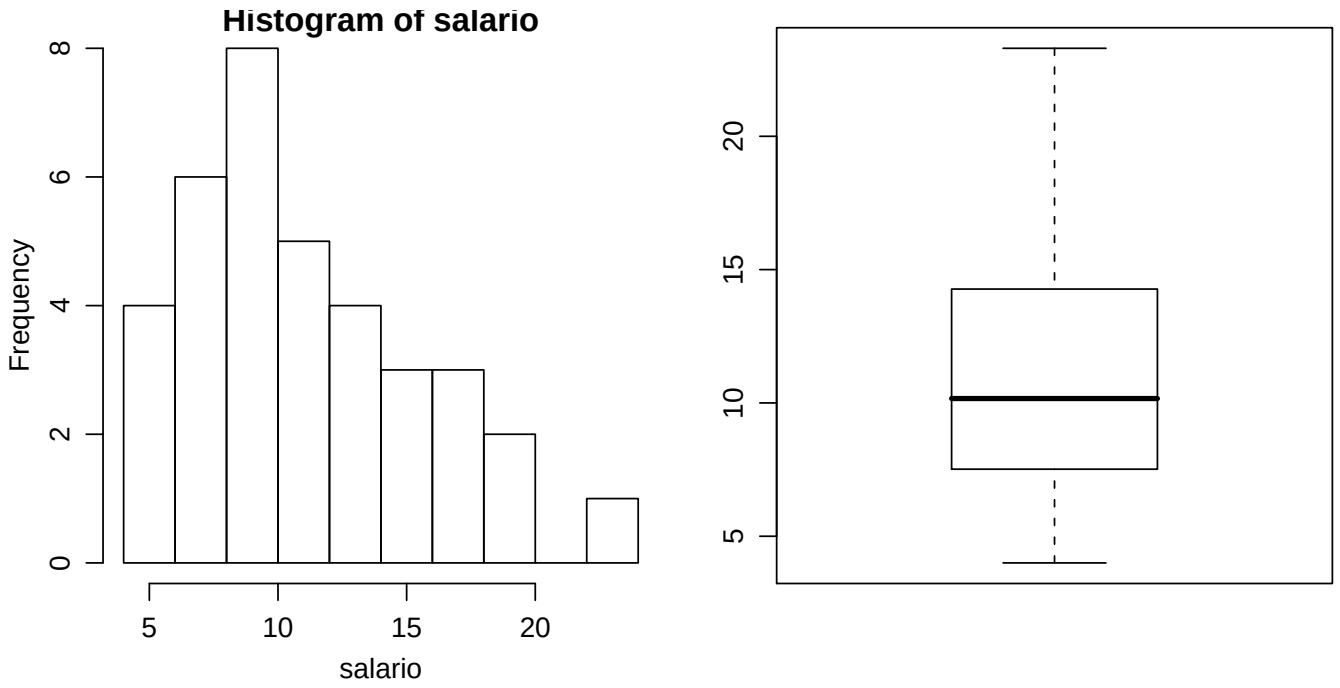


Figura 11: Histograma (esquerda) e boxplot (direita) para a variável `salario`.

```
function (x, ...)
NULL
> args(cut.default)
function (x, breaks, labels = NULL, include.lowest = FALSE, right = TRUE,
    dig.lab = 3, ...)
NULL
> salario.tb <- table(cut(salario, seq(3.5, 23.5, l = 8)))
> prop.table(salario.tb)
 (3.5,6.36] (6.36,9.21] (9.21,12.1] (12.1,14.9] (14.9,17.8] (17.8,20.6]
 0.13888889  0.27777778  0.22222222  0.16666667  0.11111111  0.05555556
(20.6,23.5]
 0.02777778
```

Na sequência vamos mostrar dois possíveis gráficos para variáveis contínuas: histograma e *box-plot* conforme Figura 9.2.1.

```
> hist(salario)
> boxplot(salario)
```

Uma outra representação gráfica para variáveis numéricas é o diagrama ramo-e-folhas que pode ser obtido conforme mostrado a seguir.

```
> stem(salario)
  The decimal point is at the |

   4 | 0637
   6 | 379446
   8 | 15791388
```

```
  10 | 5816
  12 | 08268
  14 | 77
  16 | 0263
  18 | 84
  20 |
  22 | 3
```

Finalmente medidas s obtidas da mesma forma que para variáveis discretas. Veja alguns exemplos a seguir.

```
> salario.md <- median(salario, na.rm = T)
> salario.md
[1] 10.165
> salario.me <- mean(salario, na.rm = T)
> salario.me
[1] 11.12222
> range(salario, na.rm = T)
[1]  4.0 23.3
> salario.A <- diff(range(salario, na.rm = T))
> salario.A
[1] 19.3
> var(salario, na.rm = T)
[1] 21.04477
> salario.dp <- sd(salario, na.rm = T)
> salario.dp
[1] 4.587458
> salario.cv <- 100 * salario.dp/salario.me
> salario.cv
[1] 41.24587
> salario.qt <- quantile(salario, na.rm = T)
> salario.ai <- salario.qt[4] - salario.qt[2]
> salario.ai
   75%
6.5075
> summary(salario)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  4.000   7.552  10.160  11.120  14.060  23.300
> fivenum(salario)
[1]  4.000  7.515 10.165 14.270 23.300
```

9.2.2 Análise Bivariada

Na análise bivariada procuramos identificar relaccões entre duas variáveis. Assim como na univariada estas relações podem ser resumidas por gráficos, tabelas e/ou medidas estatística. O tipo de resumo vai depender dos tipos das variáveis envolvidas. Vamos considerar três possibilidades:

- qualitativa *vs* qualitativa
- qualitativa *vs* quantitativa
- quantitativa *vs* qualitativa

Salienta-se ainda que:

- as análise mostradas a seguir não esgotam as possibilidades de análises envolvendo duas variáveis e devem ser vistas apenas como uma sugestão inicial
- relações entre duas variáveis devem ser examinadas com cautela pois podem ser mascaradas por uma ou mais variáveis adicionais não considerada na análise. Estas são chamadas *variáveis de confundimento*. Análises com variáveis de confundimento não serão discutidas neste ponto.

Qualitativa *vs* **Qualitativa** Vamos considerar as variáveis `civil` (estado civil) e `instrucao` (grau de instrução). A tabela envolvendo duas variáveis é chamada *tabela de cruzamento* e pode ser apresentada de várias formas, conforme ilustrado abaixo. A forma mais adequada vai depender dos objetivos da análise e da interpretação desejada para os dados. Iniciamente obtemos a tabela de frequências absolutas. Depois usamos `prop.table()` para obter frequência ralativas globais, por linha e por coluna.

```
> civ.gi.tb <- table(civil, instrucao)
> civ.gi.tb
          instrucao
civil      1oGrau 2oGrau Superior
  solteiro      7      6        3
  casado        5     12        3
> prop.table(civ.gi.tb)
          instrucao
civil          1oGrau     2oGrau   Superior
  solteiro 0.19444444 0.16666667 0.08333333
  casado   0.13888889 0.33333333 0.08333333
> prop.table(civ.gi.tb, margin = 1)
          instrucao
civil      1oGrau 2oGrau Superior
  solteiro 0.4375 0.3750   0.1875
  casado   0.2500 0.6000   0.1500
> prop.table(civ.gi.tb, margin = 2)
          instrucao
civil         1oGrau    2oGrau  Superior
  solteiro 0.5833333 0.3333333 0.5000000
  casado   0.4166667 0.6666667 0.5000000
```

Na Figura 9.2.2 mostramos dois gráficos de barras.

```
> barplot(civ.gi.tb, legend = T)
> barplot(civ.gi.tb, beside = T, legend = T)
```

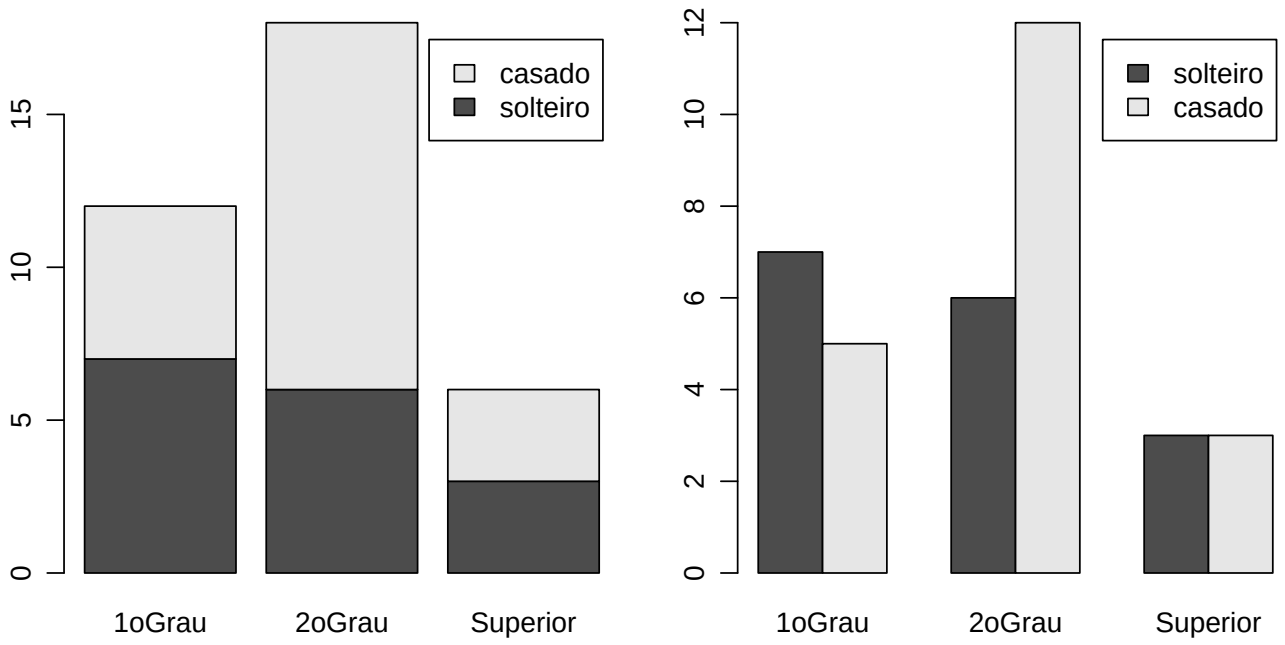


Figura 12: Dois tipos de gráficos de barras ilustrando o cruzamento das variáveis `civil` e `instrucao`.

Medidas de associação entre duas variáveis qualitativas incluem o Chi-quadrado dado por:

$$\chi^2 = \sum_{i=1}^{k} \frac{(o_i - e_i)^2}{e_i} \, ,$$

onde o_i e e_i são, respectivamente, frequências observadas e esperadas nas k posições da tabela de cruzamento das variáveis. Outras medidas derivadas desta são o o coeficiente de contingência C e o coeficiente de contingência modificado C_1 dados por:

$$C = \sqrt{\frac{\chi^2}{\chi^2 + n}} \quad , \quad C_1 = \frac{C}{[(t-1)/t]^2},$$

onde n é o número de observações e t é o mínimo entre o número de linas e colunas da tabela. Os comandos a seguir mostram como obter todas estas medidas.

```
> summary(civ.gi.tb)
Number of cases in table: 36
Number of factors: 2
Test for independence of all factors:
        Chisq = 1.9125, df = 2, p-value = 0.3843
        Chi-squared approximation may be incorrect
> names(summary(civ.gi.tb))
[1] "n.vars"    "n.cases"   "statistic" "parameter" "approx.ok" "p.value"
[7] "call"
> chisq <- summary(civ.gi.tb)$stat
> chisq
[1] 1.9125
```

```
> n <- sum(civ.gi.tb)
> n
[1] 36
> C <- sqrt(chisq/(chisq + n))
> C
[1] 0.2245999
> t <- min(dim(civ.gi.tb))
> C1 <- C/((t - 1)/t)^2
> C1
[1] 0.8983995
```

Muitas vezes é necessário reagrupar categorias porque algumas frequências são muito baixas. Por exemplo vamos criar uma nova variável para agrupar 2^o Grau e Superior usando `ifelse()` e depois podemos refazer as análises do cruzamento com esta nova variável

```
> instrucao1 <- ifelse(instrucao == "1oGrau", 1, 2)
> instrucao1 <- factor(instrucao1, label = c("1oGrau", "2o+Superior"),
+     lev = 1:2, ord = T)
> table(instrucao1)
instrucao1
     1oGrau 2o+Superior
         12          24
> table(civil, instrucao1)
          instrucao1
civil      1oGrau 2o+Superior
  solteiro      7           9
  casado        5          15
> summary(table(civil, instrucao1))
Number of cases in table: 36
Number of factors: 2
Test for independence of all factors:
        Chisq = 1.4062, df = 1, p-value = 0.2357
```

Qualitativa *vs* **Quantitativa** Para exemplificar este caso vamos considerar as variáveis `instrucao` e `salario`.

Para se obter uma tabela de frequências é necessário agrupar a variável quantitativa em classes. No exemplo a seguir vamos agrupar a variável salário em 4 classes definidas pelos quartis usando `cut()`. Após agrupar esta variável obtemos a(s) tabela(s) de cruzamento como mostrado no caso anterior.

```
> quantile(salario)
     0%     25%     50%     75%    100%
 4.0000  7.5525 10.1650 14.0600 23.3000
> salario.cl <- cut(salario, quantile(salario))
> ins.sal.tb <- table(instrucao, salario.cl)
> ins.sal.tb
```

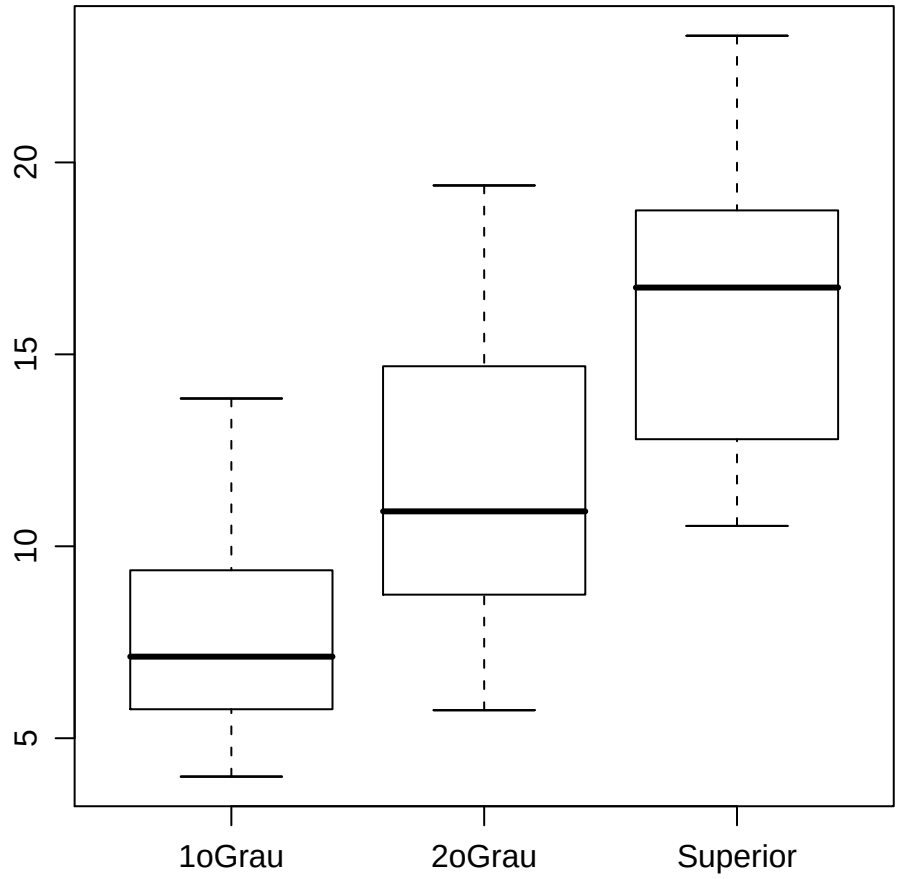


Figura 13: Boxplot da variável `salario` para cada nível da variável `instrucao`.

```
          salario.cl
instrucao  (4,7.55] (7.55,10.2] (10.2,14.1] (14.1,23.3]
  1oGrau          6           3           2           0
  2oGrau          2           6           5           5
  Superior        0           0           2           4
> prop.table(ins.sal.tb, margin = 1)
          salario.cl
instrucao   (4,7.55] (7.55,10.2] (10.2,14.1] (14.1,23.3]
  1oGrau   0.5454545   0.2727273   0.1818182   0.0000000
  2oGrau   0.1111111   0.3333333   0.2777778   0.2777778
  Superior 0.0000000   0.0000000   0.3333333   0.6666667
```

No gráfico vamos considerar que neste exemplo a instrução deve ser a variável explicativa e portanto colocada no eixo-X e o salário é a variável resposta e portanto no eixo-Y. Isto é, consideramos que a instrução deve explicar, ainda que parcialmente, o salário (e não o contrário!). Vamos então obter um *boxplot* dos salários para cada nível de instrução. Note que o função abaixo usamos a notação de *formula* do R, com `salario instrucao` indicando que a variável `salario` é explicada ($\sim$) pela variável `instrucao`.

```
> boxplot(salario ~ instrucao)
```

Poderíamos ainda fazer gráficos com a variável `salario` agrupada em classes, e neste caso os gráficos seriam como no caso anterior com duas variáveis qualitativas.

Para as medidas o usual é obter um resumo da quantitativa como mostrado na análise univariada, porém agora infromando este resumo para cada nível do fator qualitativo. A seguir mostramos alguns exemplos de como obter a média, desvio padrão e o resumo de cinco números do salário para cada nível de instrução.

```
> tapply(salario, instrucao, mean)
   1oGrau    2oGrau  Superior
 7.836667 11.528333 16.475000
> tapply(salario, instrucao, sd)
  1oGrau   2oGrau Superior
2.956464 3.715144 4.502438
> tapply(salario, instrucao, quantile)
$`1oGrau`
     0%     25%     50%     75%    100%
 4.0000  6.0075  7.1250  9.1625 13.8500

$`2oGrau`
     0%     25%     50%     75%    100%
 5.7300  8.8375 10.9100 14.4175 19.4000

$Superior
     0%     25%     50%     75%    100%
10.5300 13.6475 16.7400 18.3775 23.3000
```

Quantitativa *vs* **Quantitativa** Para ilustrar este caso vamos considerar as variáveis `salario` e `idade`. Para se obter uma tabela é necessário agrupar as variáveis em classes conforma fizemos no caso anterior. Nos comandos abaixo agrupamos as duas variáveis em classes definidas pelos respectivos quartis gerando portanto uma tabela de cruzamento 4×4.

```
> idade.cl <- cut(idade, quantile(idade))
> table(idade.cl)
idade.cl
(20.8,30.7] (30.7,34.9] (34.9,40.5] (40.5,48.9]
          8           9           9           9
> salario.cl <- cut(salario, quantile(salario))
> table(salario.cl)
salario.cl
   (4,7.55] (7.55,10.2] (10.2,14.1] (14.1,23.3]
          8           9           9           9
> table(idade.cl, salario.cl)
             salario.cl
idade.cl      (4,7.55] (7.55,10.2] (10.2,14.1] (14.1,23.3]
  (20.8,30.7]        2           2           2           1
  (30.7,34.9]        1           3           3           2
  (34.9,40.5]        1           3           2           3
  (40.5,48.9]        3           1           2           3
```

```
> prop.table(table(idade.cl, salario.cl), mar = 1)
             salario.cl
idade.cl       (4,7.55] (7.55,10.2] (10.2,14.1] (14.1,23.3]
  (20.8,30.7] 0.2857143   0.2857143   0.2857143   0.1428571
  (30.7,34.9] 0.1111111   0.3333333   0.3333333   0.2222222
  (34.9,40.5] 0.1111111   0.3333333   0.2222222   0.3333333
  (40.5,48.9] 0.3333333   0.1111111   0.2222222   0.3333333
```

Caso queiramos definir um número menos de classes podemos fazer como no exemplo a seguir onde cada variável é dividida em 3 classes e gerando um tabela de cruzamento 3×3.

```
> idade.cl1 <- cut(idade, quantile(idade, seq(0, 1, len = 4)))
> salario.cl1 <- cut(salario, quantile(salario, seq(0, 1, len = 4)))
> table(idade.cl1, salario.cl1)
             salario.cl1
idade.cl1     (4,8.65] (8.65,12.9] (12.9,23.3]
  (20.8,32.1]        3           5           2
  (32.1,37.8]        4           3           5
  (37.8,48.9]        3           4           5
> prop.table(table(idade.cl1, salario.cl1), mar = 1)
             salario.cl1
idade.cl1      (4,8.65] (8.65,12.9] (12.9,23.3]
  (20.8,32.1] 0.3000000   0.5000000   0.2000000
  (32.1,37.8] 0.3333333   0.2500000   0.4166667
  (37.8,48.9] 0.2500000   0.3333333   0.4166667
```

O gráfico adequado para representar duas variáveis quantitativas é um diagrama de dispersão. Note que se as variáveis envolvidas puderem ser classificadas como "explicativa"e "resposta"devemos colocar a primeira no eixo-X e a segunda no eixo-Y. Neste exemplo é razoável admitir que a idade deve explicar, ao menos parcialmente, o salário e portanto fazemos o gráfico com idade n eixo-X.

```
> plot(idade, salario)
```

Para quantificar a associação entre variáveis deste tipo usamos um coeficiente de correlação. A função `cor()` do R possui opção para três coeficientes tendo como *default* o coeficiente de correlação linear de Pearson.

```
> cor(idade, salario)
[1] 0.3651397
> cor(idade, salario, method = "kendall")
[1] 0.214456
> cor(idade, salario, method = "spearman")
[1] 0.2895939
```

Lembre que ao iniciar as análises com este conjunto de dados anexamos os dados com o comando `attach(milsa)`. Portanto ao terminar as análises com estes dados devemos desanexar este conjunto de dados com o `detach()`

```
> detach(milsa)
```

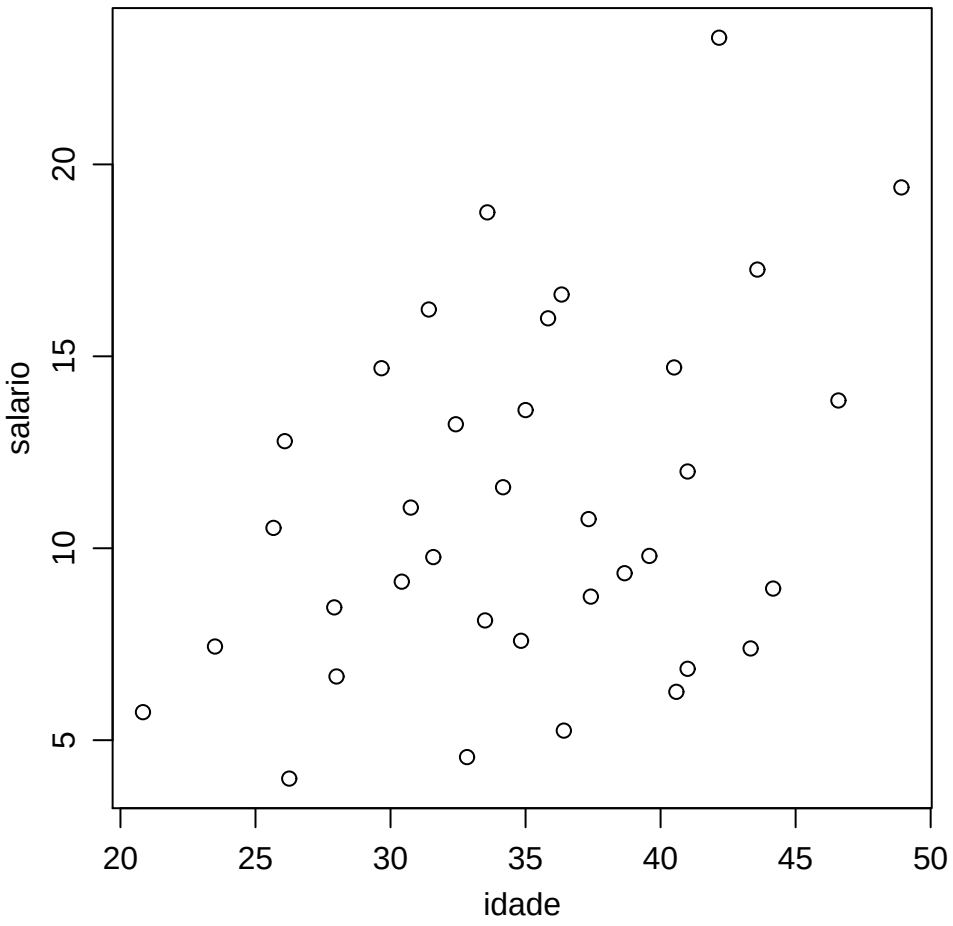


Figura 14: Diagrama de dispersão para as variáveis `salario` e `idade`.

9.3 Uma demonstração de recursos gráficos do R

O R vem com algumas demonstrações (*demos*) de seus recursos "embutidas" no programa. Para listar as demos disponíveis digite na linha de comando:

```
> demo()
```

Para rodar uma delas basta colocar o nome da escolhida entre os parênteses. As *demos* são úties para termos uma idéia dos recursos disponíveis no programa e para ver os comandos que devem ser utilizados.

Por exemplo, vamos rodar a *demo* de recursos gráficos. Note que os comandos vão aparecer na janela de comandos e os gráficos serão automaticamente produzidos na janela gráfica. A cada passo voce vai ter que teclar `ENTER` para ver o próximo gráfico.

- no "prompt" do programa R digite:

  ```
  > demo(graphics)
  ```

- Voce vai ver a seguinte mensagem na tela:

  ```
  demo(graphics)
  ---- ~~~~~~~~

  Type  <Return>   to start :
  ```

- pressione a tecla ENTER
- a "demo" vai ser iniciada e uma tela gráfica irá se abrir. Na tela de comandos serão mostrados comandos que serão utilizados para gerar um gráfico seguidos da mensagem:

  ```
  Hit <Return> to see next plot:
  ```

- inspecione os comandos e depois pressione novamente a tecla ENTER.
  Agora voce pode visualizar na janela gráfica o gráfico produzido pelos comandos mostrados anteriormente. Inspecione o gráfico cuidadosamente verificando os recursos utilizados (título, legendas dos eixos, tipos de pontos, cores dos pontos, linhas, cores de fundo, etc).
- agora na tela de comandos apareceram novos comandos para produzir um novo gráfico e a mensagem:

  ```
  Hit <Return> to see next plot:
  ```

- inspecione os novos comandos e depois pressione novamente a tecla ENTER.
  Um novo gráfico surgirá ilustrando outros recursos do programa.
  Prossiga inspecionando os gráficos e comandos e pressionando ENTER até terminar a "demo".
  Experimente outras demos como `demo(persp)` e `demo(image)`, por exemplo.

- para ver o código fonte (comandos) de uma `demo` voce pode utilizar comandos como se seguem (e de forma análoga para outras `"demos"`:

  ```
  > file.show(system.file("demo/graphics.R", package="graphics"))
  > file.show(system.file("demo/plotmath.R", package="graphics"))
  > file.show(system.file("demo/persp.R", package="graphics"))
  ```

9.4 Outros dados disponíveis no R

Há vários conjuntos de dados incluídos no programa R como, por exemplo, o conjunto `mtcars`. Estes conjuntos são todos documentados, isto é, voce pode usar a função `help` para obter uma descrição dos dados. Para ver a lista de conjuntos de dados disponíveis digite `data()`. Por exemplo tente os seguintes comandos:

```
> data()
> data(women)
> women
> help(woman)
```

10 Gráficos no R

10.1 Exemplos dos recursos gráficos

O R vem com algumas demonstrações (*demos*) de seus recursos "embutidas" no programa. Para listar as *demos* disponíveis digite na linha de comando:

```
> demo()
```

Para rodar uma delas basta colocar o nome da escolhida entre os parênteses. As *demos* são úties para termos uma idéia dos recursos disponíveis no programa e para ver os comandos que devem ser utilizados.

Por exemplo, vamos rodar a *demo* de recursos gráficos. Note que os comandos vão aparecer na janela de comandos e os gráficos serão automaticamente produzidos na janela gráfica. A cada passo voce vai ter que teclar `ENTER` para ver o próximo gráfico.

- no "prompt" do programa R digite:

  ```
  > demo(graphics)
  ```

- Voce vai ver a seguinte mensagem na tela:

  ```
  demo(graphics)
  ---- ~~~~~~~~

  Type  <Return>   to start :
  ```

- pressione a tecla ENTER
- a "demo" vai ser iniciada e uma tela gráfica irá se abrir. Na tela de comandos serão mostrados comandos que serão utilizados para gerar um gráfico seguidos da mensagem:

  ```
  Hit <Return> to see next plot:
  ```

- inspecione os comandos e depois pressione novamente a tecla ENTER.
  Agora voce pode visualizar na janela gráfica o gráfico produzido pelos comandos mostrados anteriormente. Inspecione o gráfico cuidadosamente verificando os recursos utilizados (título, legendas dos eixos, tipos de pontos, cores dos pontos, linhas, cores de fundo, etc).
- agora na tela de comandos apareceram novos comandos para produzir um novo gráfico e a mensagem:

  ```
  Hit <Return> to see next plot:
  ```

- inspecione os novos comandos e depois pressione novamente a tecla ENTER.
  Um novo gráfico surgirá ilustrando outros recursos do programa.
  Prossiga inspecionando os gráficos e comandos e pressionando ENTER até terminar a "demo".
  Experimente outras demos como `demo(persp)` e `demo(image)`, por exemplo.

- para ver o código fonte (comandos) de uma `demo` voce pode utilizar comandos como se seguem (e de forma análoga para outras `"demos"`:

  ```
  > file.show(system.file("demo/graphics.R", package="graphics"))
  > file.show(system.file("demo/plotmath.R", package="graphics"))
  > file.show(system.file("demo/persp.R", package="graphics"))
  ```

Galeria de gráficos do R

– *R Graph Gallery* é uma página com diversos exemplos de gráficos no R e os comandos para produzí-los

10.2 Algumas configurações de gráficos no *R*

Gráficos múltiplos na janela gráfica
O principal recurso para controlar o aspecto de gráficos no R é dado pela função de configuração `par()`, que permite configurar formato, tamanho, subdivisões, margens, entre diversas outras opções. Por exemplo `par(mfrow=c(1,2))` divide a janela gráfica em um *frame* que perite acomodar dois gráficos em uma linha e `par(mfrow=c(3,4))` permite acomodar 12 gráficos em uma mesma janela arranjados em três linhas e quatro colunas.

Gráficos em arquivos
Por *default* gráficos são mostrados em uma janela na tela do computador, ou seja, a tela é o dispositivo de saída (*output device*) padrão para gráficos. Para produzir gráficos em arquivos basta redirecionar o dispositivo de saída para o formato gráfico desejado. O código a seguir mostra como gerar um um histograma de 200 amostras de uma distribuição normal padrão em um arquivo chamado `figura1.pdf` em formato `pdf`.

```
> pdf("figura1.pdf")
> hist(rnorm(200))
> dev.off()
```

Caso deseje-se o arquivo em outro formato gráfico a função adequada deve ser chamada. Por exemplo, `jpeg()` para formatos `.jpg` (or `.jpeg`) que são muito usados em páginas web, `png()`, `postscript()` (para gráficos em formato `.ps` ou `.eps`), entre outros. Alguns dos dispositivos gráficos são exclusivos de certos sistemas operacionais como poe exemplo `wmf()` para o sistema operacional `WINDOWS`. Cada uma des funçõa possuem argumentos adicionais que permitem controlar tamanho, resolução, entre outros atributos do arquivo gráfico. É importante notar que o comando `dev.off()` é compulsório devendo ser usado para que o arquivo gráfico seja ”fechado”.

Modificando gráficos
Gráficos no R são tipicamente construídos com opções padrão definidas pelo programa, mas podem ser modificados ou ter elementos adicionados conforme desejado pelo usuário.

A melhor forma para entender como modificar gráficos é pensar que cada elemento pode ser controlado por uma função, e elementos são adicionados ao gráfico para cada chamada de função específica, de forma semelhante ao que se faria ao desenhar em um papel. Um exemplo típico é a adição de legenda a um gráfico já feito, o que pode ser feito por `legend()`
NOTA: Se algo já feito deve ser mudado então é necessário repetir os comandos anteriores um a um até chegar no que se deseja modificar. Este comportamento difere de alguns outros programas que permitem modificar um gráfico já desenhado.

```
> x <- rnorm(200)
> hist(x)
```

```
> hist(x, main = "", axes = F, xlab = "dados", ylab = "frequências absolutas")
> axis(1, at = seq(-2.5, 3.5, by = 0.5), pos = 0)
> axis(2, at = seq(0, 50, by = 10), pos = -2.5)
```

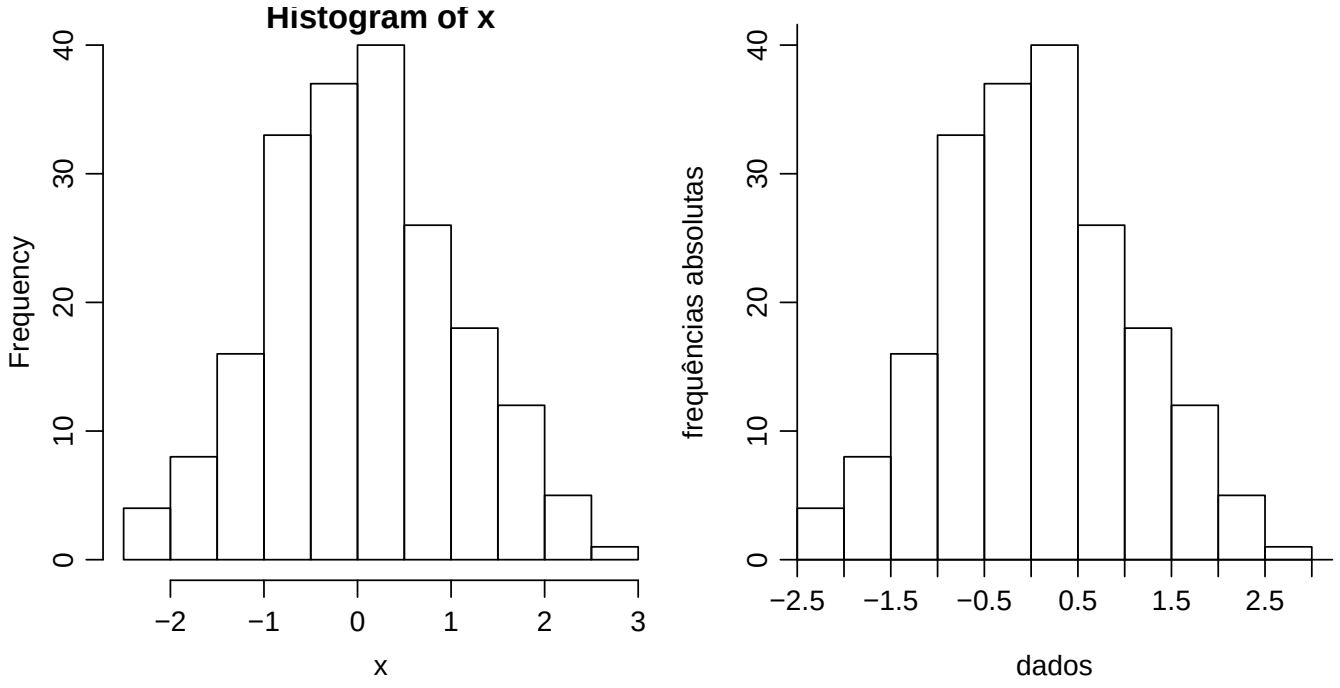


Figura 15: Histograma gerado com opções padrão (esquerda) e modificadas (direita).

Vejamos na Figura reffig:eixos um exemplo frequentemente citado por usuários. No gráfico da esquerda está o histograma dos dados de uma amostra de tamanho 200 produzido com opções padrão (*default*) da função `hist()` a partir dos seguintes comandos. No gráfico da direita nota-se que o título foi removido, o texto dos eixos foi modificado e a posição dos eixos foi alterada fazendo com que as barras do histograma sejam desenhadas junto aos eixos. Para isto na chamada de `hist()` passamos um valor vazio para o argumento `main` o que causa a remoção do título do gráfico. Os texto dos eixos são definidos por `xlab` e `ylab`. Finalmente, para modificar os eixos iniciamos removendo os eixos do gráfico inicial com `axes=FALSE` e depois os adicionamos com `axis()` na posição desejada, sendo que no primeiro argumento da função as opções 1 e 2 correspondem aos eixos das abcissas e ordenadas, respectivamente.

11 Explorando distribuições de probabilidade empíricas

Na Sessão 13 vimos com usar distribuições de probabilidade no R. Estas distribuições tem expressões conhecidas e são indexadas por um ou mais parâmetros. Portanto, conhecer a distribuição e seu(s) parâmetro(s) é suficiente para caracterizar completamente o comportamento distribuição e extrair resultados de interesse.

Na prática em estatística em geral somente temos disponível uma amostra e não conhecemos o mecanismo (distribuição) que gerou os dados. Muitas vezes o que se faz é: (i) assumir que os dados são provenientes de certa distribuição, (ii) estimar o(s) parâmetro(s) a partir dos dados. Depois disto procura-se verificar se o ajuste foi "bom o suficiente", caso contrário tenta-se usar uma outra distribuição e recomeça-se o processo.

A necessidade de estudar fenômenos cada vez mais complexos levou ao desenvolvimento de métodos estatísticos que às vezes requerem um flexibilidade maior do que a fornecida pelas distribuições de probabilidade de forma conhecida. Em particular, métodos estatísticos baseados em simulação podem gerar amostras de quantidades de interesse que não seguem uma distribuição de probabilidade de forma conhecida. Isto ocorre com frequência em métodos de *inferência Bayesiana* e métodos computacionalmente intensivos como *bootstrap*, *testes Monte Carlo*, dentre outros.

Nesta sessão vamos ver como podemos, a partir de um conjunto de dados explorar os possíveis formatos da distribuição geradora sem impor nenhuma forma paramétrica para função de densidade.

11.1 Estimação de densidades

A estimação de densidades é implementada no R pela função `density()`. O resultado desta função é bem simples e claro: ela produz uma função de densidade obtida a partir dos dados sem forma paramétrica conhecida. Veja este primeiro exemplo que utiliza o conjunto de dados `precip` que já vem com o R e contém valores médios de precipitação em 70 cidades americanas. Nos comandos a seguir vamos carregar o conjunto de dados, fazer um histograma de frequências relativas e depois adicionar a este histograma a linha de densidade estimada, conforma mostra a Figura 16.

```
> data(precip)
> hist(precip, prob = T)
> precip.d <- density(precip)
> lines(precip.d)
```

Portanto podemos ver que `density()` "suaviza" o histograma, capturando e concentrando-se nos principais aspectos dos dados disponíveis. Vamos ver na Figura 17 uma outra forma de visualizar os dados e sua densidade estimada, agora sem fazer o histograma.

```
> plot(precip.d)
> rug(precip)
```

Embora os resultados mostrados acima seja simples e fáceis de entender, há muita coisa por trás deles! Não vamos aqui estudar com detalhes esta função e os fundamentos teóricos nos quais se baseiam esta implementação computacional pois isto estaria muito além dos objetivos e escopo deste curso. Vamos nos ater às informações principais que nos permitam compreender o básico necessário sobre o uso da função. Para maiores detalhes veja as referências na documentação da função, que pode ser vista digitando `help(density)`

Basicamente, `density()` produz o resultado visto anteriormente criando uma sequência de valores no eixo-X e estimando a densidade em cada ponto usando os dados ao redor deste ponto. Podem ser dados pesos aos dados vizinhos de acordo com sua proximidade ao ponto a ser estimado. Vamos examinar os argumentos da função.

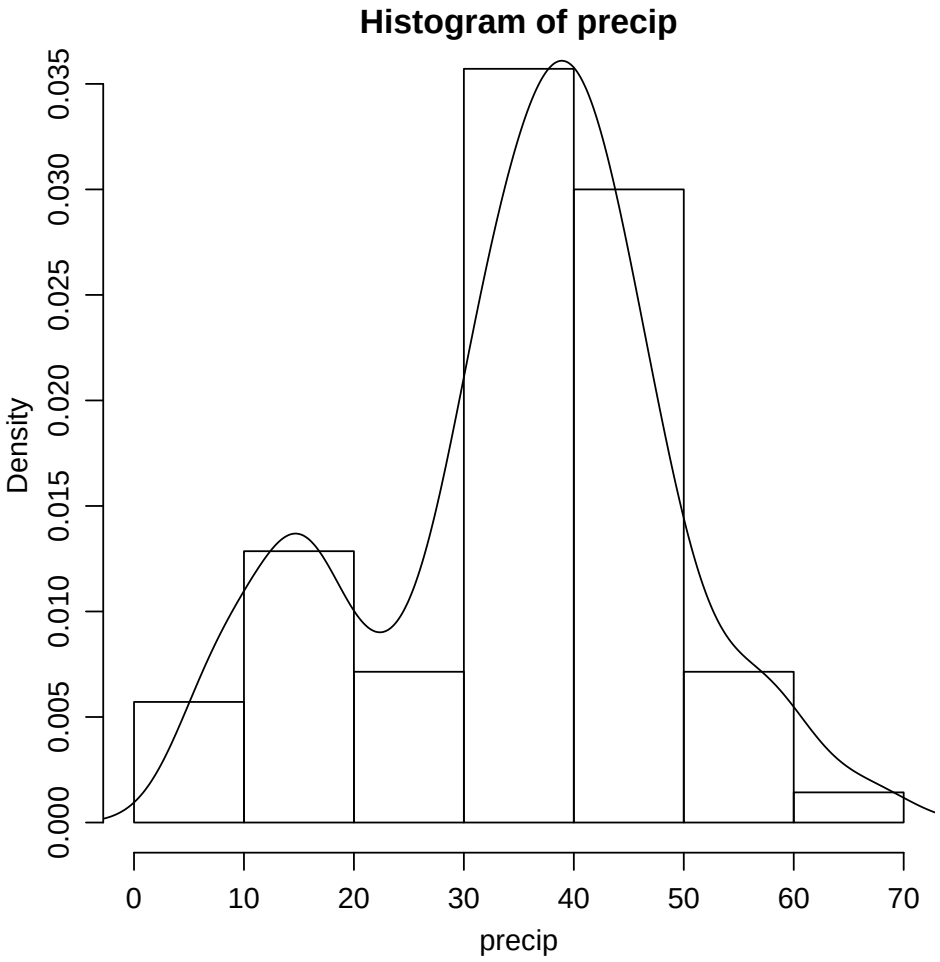


Figura 16: Histograma para os dados `precip` e a densidade estimada usando a função `density`.

```
> args(density)
function (x, ...)
NULL
```

Os dois argumentos chave são portanto `bw` e `kernel` que controlam a distância na qual se procuram vizinhos e o peso a ser dado a cada vizinho, respectivamente. Para ilustrar isto vamos experimentar a função com diferentes valores para o argumento `bw`. Os resultados estão na Figura 18. Podemos notar que o grau de suavização aumenta a medida de aumentamos os valores deste argumento e as densidades estimadas podem ser bastante diferentes!

```
> plot(density(precip, bw = 1), main = "")
> rug(precip)
> lines(density(precip, bw = 5), lty = 2)
> lines(density(precip, bw = 10), lty = 3)
> legend(5, 0.045, c("bw=1", "bw=5", "bw=10"), lty = 1:3)
```

O outro argumento importante é tipo de função de pesos, ao que chamamos de núcleo (*kernel*). O R implementa vários núcleos diferentes cujos formatos são mostrados na Figura 19.

```
> (kernels <- eval(formals(density.default)$kernel))
> plot(density(0, bw = 1), xlab = "", main = "kernels com bw = 1")
> for (i in 2:length(kernels)) lines(density(0, bw = 1, kern = kernels[i]),
```

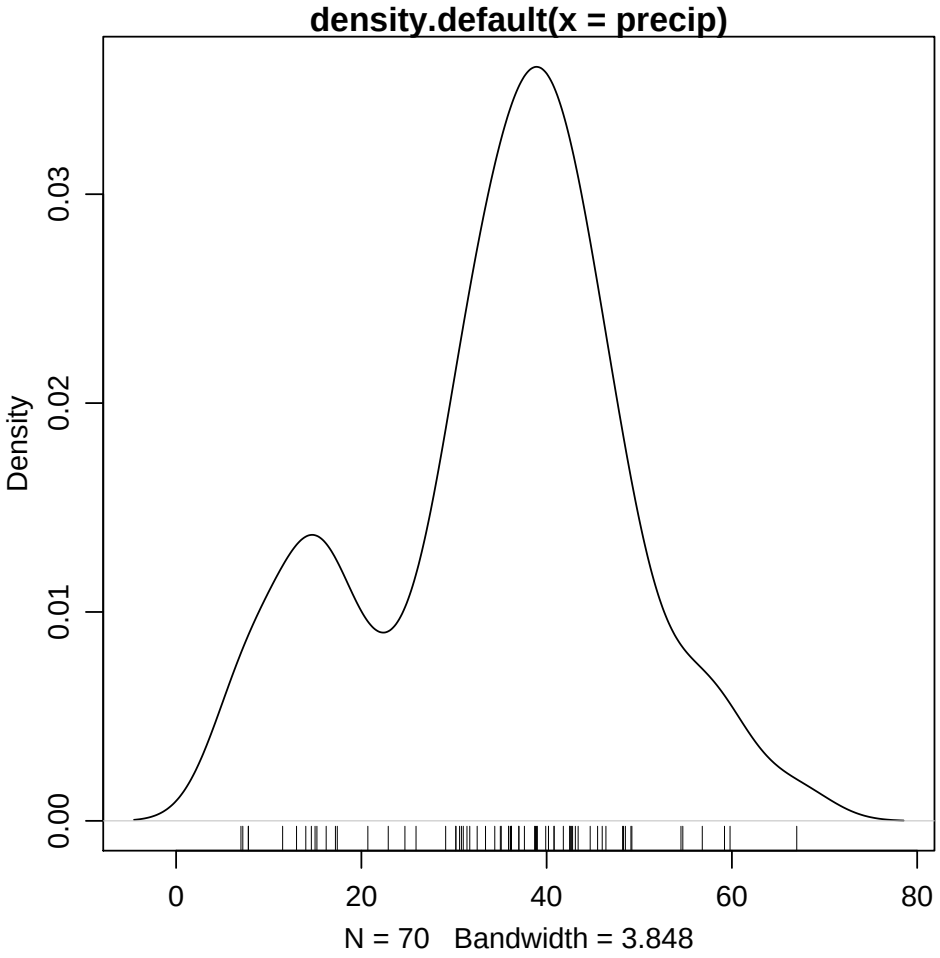


Figura 17: Dados `precip` e a densidade estimada usando a função `density`.

```
+     col = i)
> legend(1.5, 0.4, legend = kernels, col = seq(kernels), lty = 1,
+     cex = 0.8, y.int = 1)
```

Utilizando diferentes núcleos no conjunto de dados `precip` obtemos os resultados mostrados na Figura 20. Note que as densidades estimadas utilizando os diferentes núcleos são bastante similares!

```
> plot(density(precip), main = "")
> rug(precip)
> lines(density(precip, ker = "epa"), lty = 2)
> lines(density(precip, ker = "rec"), col = 2)
> lines(density(precip, ker = "tri"), lty = 2, col = 2)
> lines(density(precip, ker = "biw"), col = 3)
> lines(density(precip, ker = "cos"), lty = 3, col = 3)
> legend(0, 0.035, legend = c("gaussian", "epanechnikov", "rectangular",
+     "triangular", "biweight", "cosine"), lty = rep(1:2, 3), col = rep(1:3,
+     each = 2))
```

Portanto, inspecionando os resultados anteriores podemos concluir que a *largura de banda* (*bandwidth* – `bw`) é o que mais influencia a estimação de densidade, isto é, é o argumento mais importante. O tipo de núcleo (`kernel`) é de importância secundária.

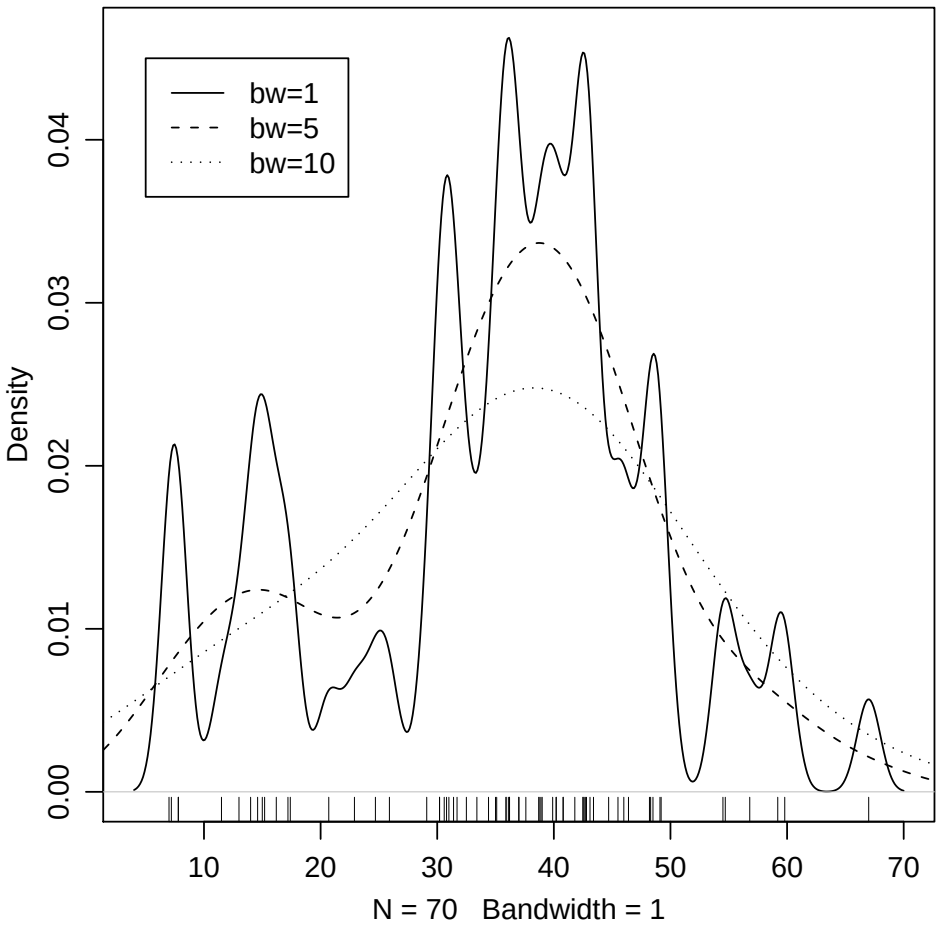


Figura 18: Densidade estimada usando a função `density` com diferentes valores para o argumento `bw`.

Bem, a esta altura voce deve estar se perguntando: mas como saber qual a largura de banda adequada? A princípio podemos tentar diferentes valores no argumento `bw` e inspecionar os resultados. O problema é que esta escolha é subjetiva. Felizmente para nós vários autores se debruçaram sobre este problema e descobriram métodos automáticos de seleção que que comportam bem na maioria das situações práticas. Estes métodos podem ser especificados no mesmo argumento `bw`, passando agora para este argumento caracteres que identificam o valor, ao invés de um valor numérico. No comando usado no início desta sessão onde não especificamos o argumento `bw` foi utilizado o valor "default" que é o método `"nrd0"` que implementa a regra prática de Silverman. Se quisermos mudar isto para o método de Sheather & Jones podemos fazer como nos comandos abaixo que produzem o resultado mostrado na Figura 21.

```
> precip.dSJ <- density(precip, bw = "sj")
> plot(precip.dSJ)
> rug(precip)
```

Os detalhes sobre os diferentes métodos implementados estão na documentação de `bw.nrd()`. Na Figura 22 ilustramos resultados obtidos com os diferentes métodos.

```
> data(precip)
> plot(density(precip, n = 1000))
> rug(precip)
```

```
[1] "gaussian"     "epanechnikov" "rectangular"  "triangular"   "biweight"
[6] "cosine"       "optcosine"
```

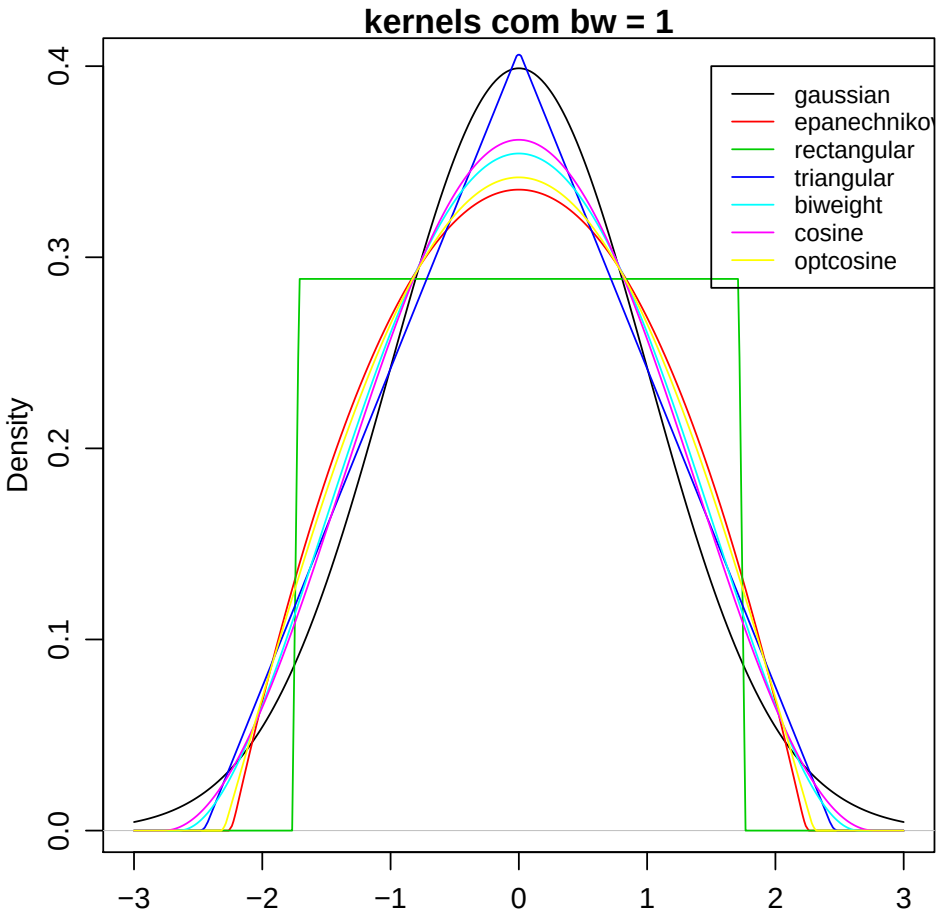


Figura 19: Diferentes núcleos implementados pela função `density`.

```
> lines(density(precip, bw = "nrd"), col = 2)
> lines(density(precip, bw = "ucv"), col = 3)
> lines(density(precip, bw = "bcv"), col = 4)
> lines(density(precip, bw = "SJ-ste"), col = 5)
> lines(density(precip, bw = "SJ-dpi"), col = 6)
> legend(55, 0.035, legend = c("nrd0", "nrd", "ucv", "bcv", "SJ-ste",
+     "SJ-dpi"), col = 1:6, lty = 1)
```

11.2 Exercícios

1. Carregar o conjunto de dados `faithful` e obter estimação de densidade para as variáveis 'tempo de erupção' e 'duração da erupção.

2. Carregar o conjunto `airquality` e densidades estimadas para as 4 variáveis medidas neste conjunto de dados.

3. Rodar e estudar os exemplos da sessão *examples* da documentação da função `density`.

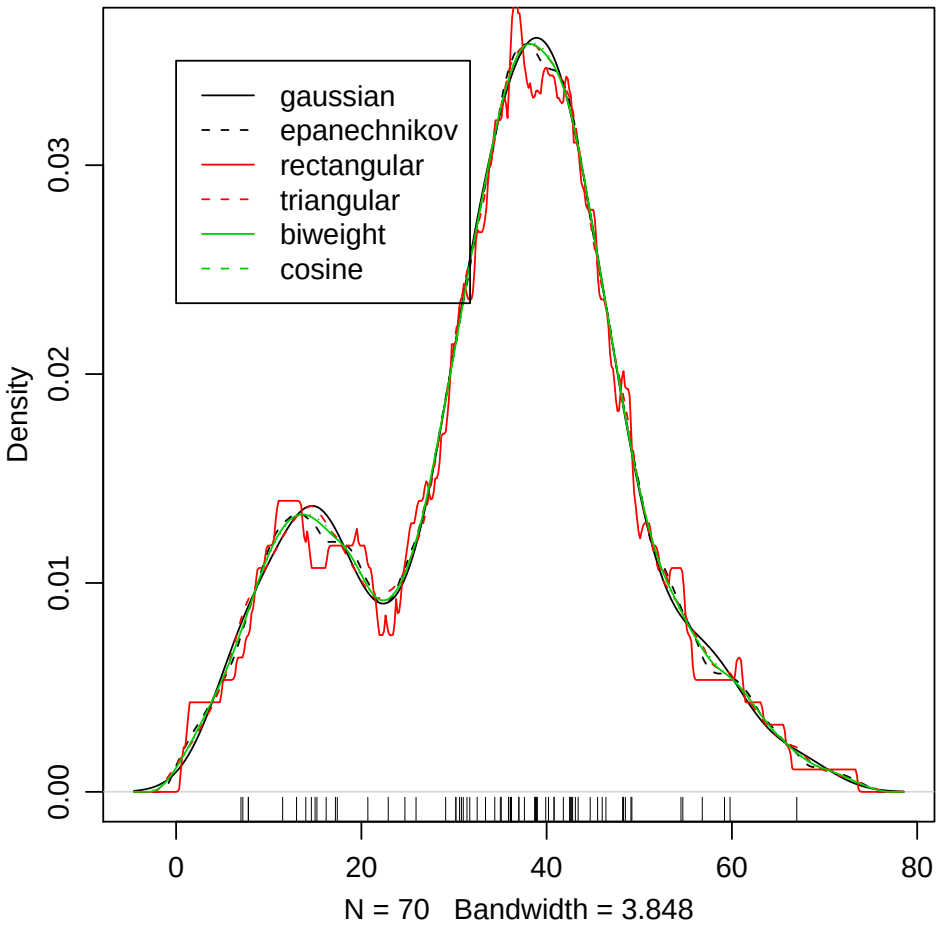


Figura 20: Densidade estimada usando a função `density` com diferentes valores para o argumento `kernel`.

12 Conceitos básicos sobre distribuições de probabilidade

O objetivo desta sessão é mostrar o uso de funções do R em problemas de probabilidade. Exercícios que podem (e devem!) ser resolvidos analiticamente são usados para ilustrar o uso do programa e alguns de seus recursos para análises numéricas.

Os problemas nesta sessão foram retirados do livro:
Bussab, W.O. & Morettin, P.A. *Estatística Básica*. 4^a edição. Atual Editora. 1987.
Note que há uma edição mais nova: (5^a edição, 2003 - Ed. Saraiva)

EXEMPLO 1 (adaptado de Bussab & Morettin, página 132, exercício 1)
Dada a função

$$f(x) = \begin{cases} 2\exp(-2x) & \text{, se } x \geq 0 \\ 0 & \text{, se } x < 0 \end{cases}$$

(a) mostre que está função é uma f.d.p.

(b) calcule a probabilidade de que $X > 1$

(c) calcule a probabilidade de que $0.2 < X < 0.8$

Para ser f.d.p. a função não deve ter valores negativos e deve integrar 1 em seu domínio. Vamos começar definindo esta função como uma *função* no R para qual daremos o nome de $f1$. A seguir

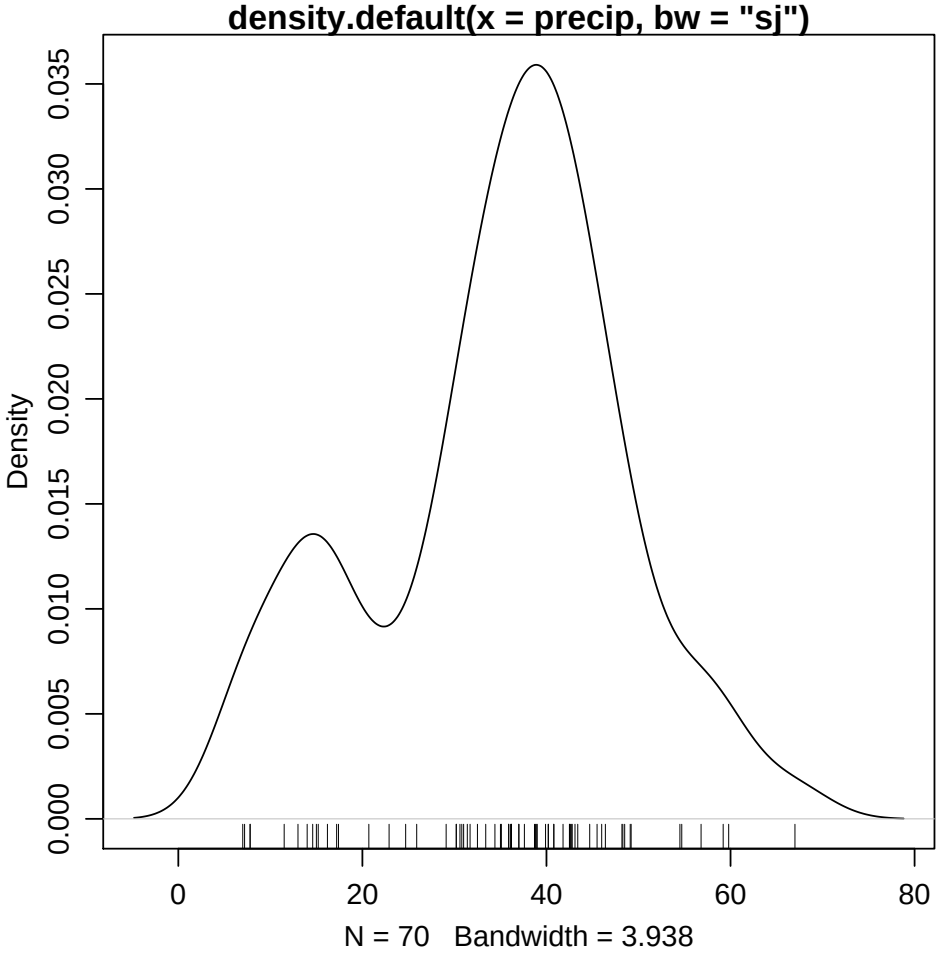


Figura 21: Densidade estimada para os dados `precip` usando a função `density` com critério de Sheather & Jones para seleção da largura de banda.

fazemos o gráfico da função. Como a função tem valores positivos para x no intervalo de zero a infinito temos, na prática, para fazer o gráfico, que definir um limite em x até onde vai o gráfico da função. Vamos achar este limite tentando vários valores, conforme mostram os comandos abaixo. O gráfico escolhido e mostrado na Figura 23 foi o produzido pelo comando `plot(f1,0,5)`.

```
> f1 <- function(x) {
+     fx <- ifelse(x < 0, 0, 2 * exp(-2 * x))
+     return(fx)
+ }
> plot(f1)
> plot(f1, 0, 10)
> plot(f1, 0, 5)
```

Para verificar que a a integral da função é igual a 1 podemos usar a função `integrate()` que efetua integração numérica. A função recebe como argumentos o objeto com a função a ser integrada e os limites de integração. Neste exemplo o objeto é `f1` definido acima e o domínio da função é $[0, \infty]$. A saída da função mostra o valor da integral (1) e o erro máximo da aproximação numérica.

```
> integrate(f1, 0, Inf)
1 with absolute error < 5e-07
```

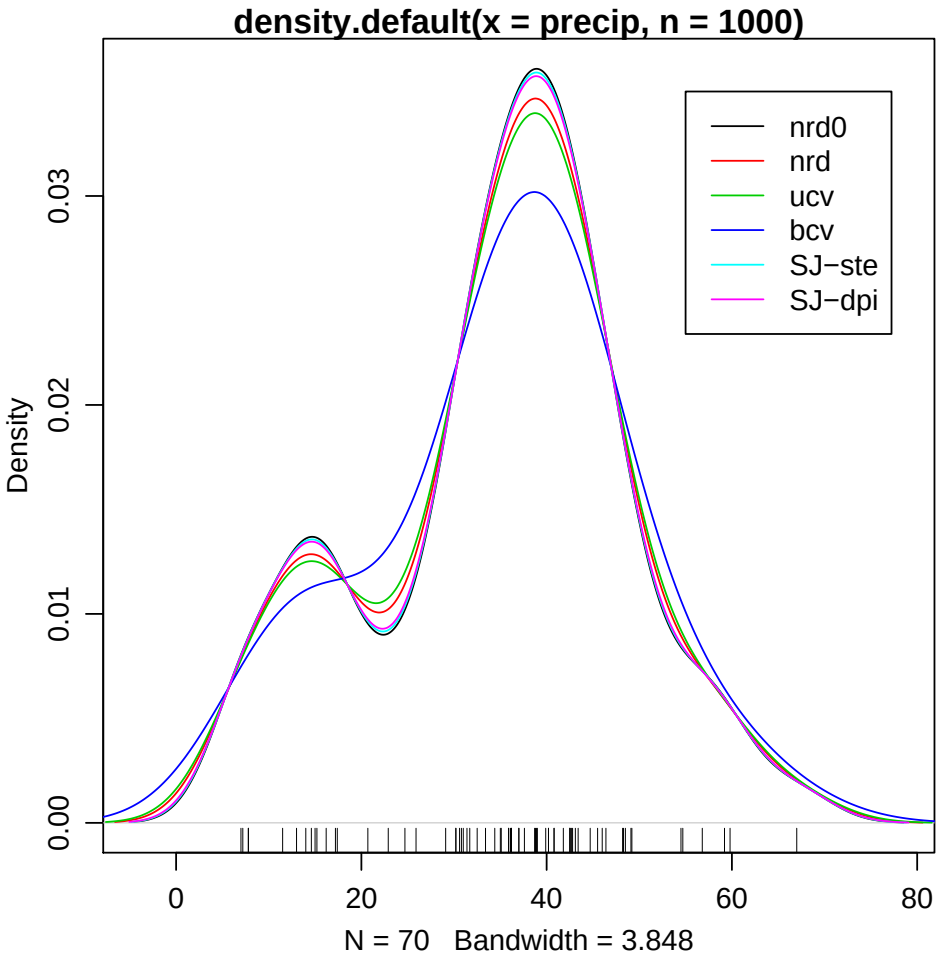


Figura 22: Diferentes métodos para largura de banda implementados pela função `density`.

Para fazer cálculos pedidos nos itens (b) e (c) lembramos que a probabilidade é dada pela área sob a curva da função no intervalo pedido. Desta forma as soluções seriam dadas pelas expressões

$$p_b = P(X > 1) = \int_1^{\infty} f(x)dx = \int_1^{\infty} 2\,e^{-2x}dx$$

$$p_c = P(0,2 < X < 0,8) = \int_{0,2}^{0,8} f(x)dx = \int_{0.2}^{0.8} 2\,e^{-2x}dx$$

cuja representação gráfica é mostrada na Figura 24. Os comandos do R a seguir mostram como fazer o gráfico de função. O comando `plot()` desenha o gráfico da função. Para destacar as áreas que correspondem às probabilidades pedidas vamos usar a função `polygon()`. Esta função adiciona a um gráfico um polígono que é definido pelas coordenadas de seus vértices. Para sombrear a área usa-se o argumento `density`. Finalmente, para escrever um texto no gráfico usamos a função `text()` com as coordenadas de posição do texto.

```
> plot(f1, 0, 5)
> polygon(x = c(1, seq(1, 5, l = 20)), y = c(0, f1(seq(1, 5, l = 20))),
+     density = 10)
> polygon(x = c(0.2, seq(0.2, 0.8, l = 20), 0.8), y = c(0, f1(seq(0.2,
+     0.8, l = 20)), 0), col = "gray")
> text(c(1.2, 0.5), c(0.1, 0.2), c(expression(p[b], p[c])))
```

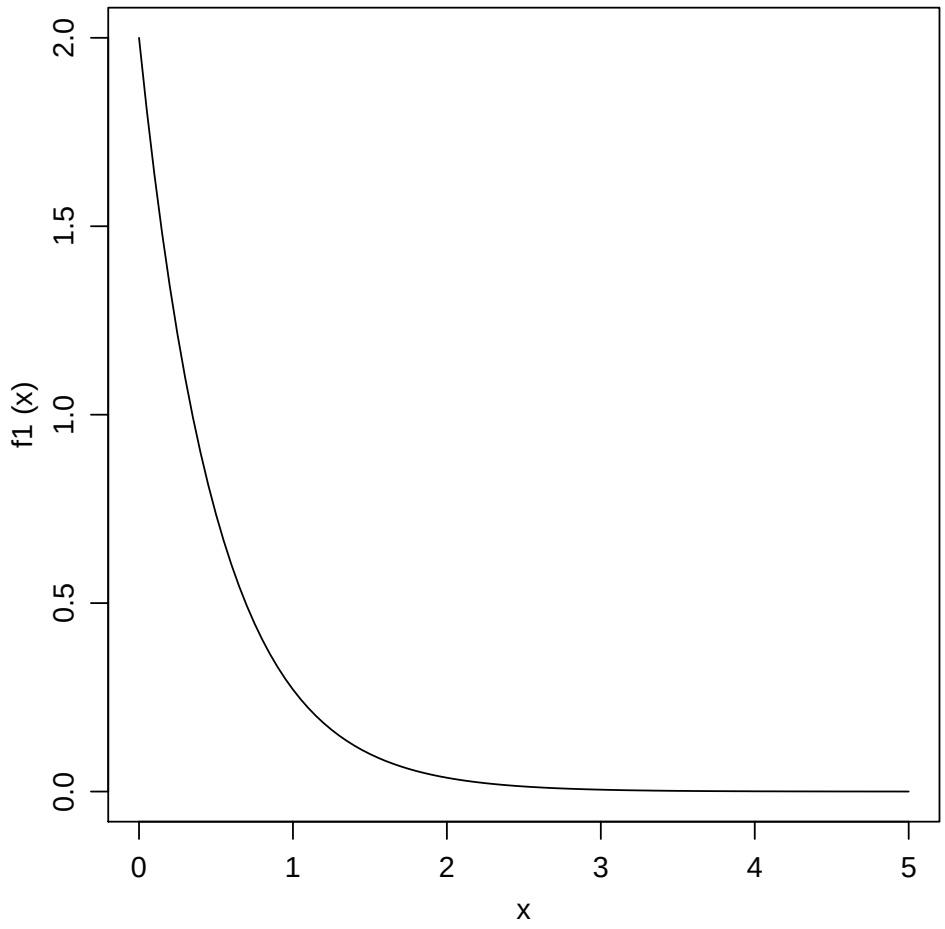


Figura 23: Gráfico da função de probabilidade do Exemplo 1.

E para obter as probabilidades pedidas usamos `integrate()`.

```
> integrate(f1, 1, Inf)
0.1353353 with absolute error < 2.1e-05
> integrate(f1, 0.2, 0.8)
0.4684235 with absolute error < 5.2e-15
```

EXEMPLO 2 (Bussab & Morettin, página 139, exercício 10)
A demanda diária de arroz em um supermercado, em centenas de quilos, é uma v.a. X com f.d.p.

$$f(x) = \begin{cases} \frac{2}{3}x \text{ , se } 0 \leq x < 1 \\ -\frac{x}{3} + 1 \text{ , se } 1 \leq x < 3 \\ 0 \text{ , se } x < 0 \text{ ou } x \geq 3 \end{cases} \tag{3}$$

(a) Calcular a probabilidade de que sejam vendidos mais que 150 kg.

(b) Calcular a venda esperada em 30 dias.

(c) Qual a quantidade que deve ser deixada à disposição para que não falte o produto em 95% dos dias?

Novamente começamos definindo um objeto do R que contém a função dada em 3.

Neste caso definimos um vetor do mesmo tamanho do argumento `x` para armazenar os valores de $f(x)$ e a seguir preenchemos os valores deste vetor para cada faixa de valor de x.

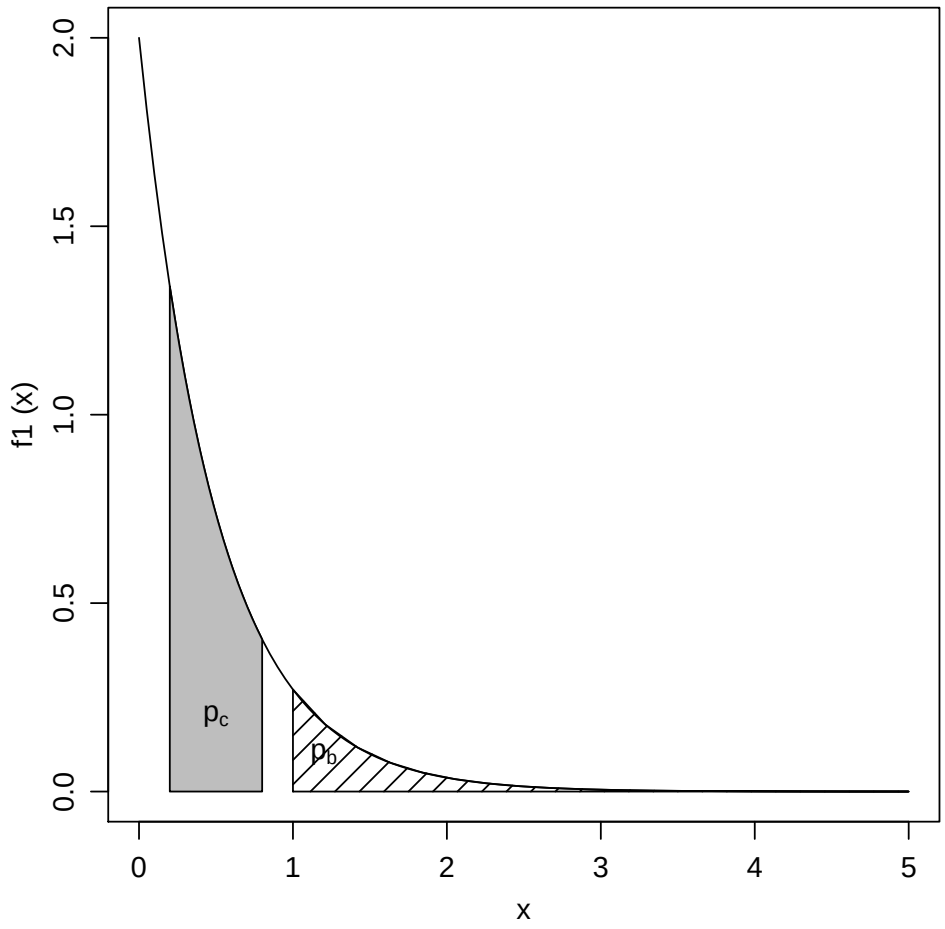


Figura 24: Probabilidades pedidas nos itens (b) e (c) do Exemplo 1.

```
> f2 <- function(x) {
+     fx <- numeric(length(x))
+     fx[x < 0] <- 0
+     fx[x >= 0 & x < 1] <- 2 * x[x >= 0 & x < 1]/3
+     fx[x >= 1 & x <= 3] <- (-x[x >= 1 & x <= 3]/3) + 1
+     fx[x > 3] <- 0
+     return(fx)
+ }
```

A seguir verificamos que a integral da função é 1 e fazemos o seu gráfico mostrado na Figura 25.

```
> integrate(f2, 0, 3)
1 with absolute error < 1.1e-15
> plot(f2, -1, 4)
```

Agora vamos responder às questões levantadas. Na questão (a) pede-se a probabilidade de que sejam vendidos mais que 150 kg (1,5 centenas de quilos), portanto a probabilidade $P[X > 1,5]$. A probabilidade corresponde à área sob a função no intervalo pedido ou seja $P[X > 1,5] = \int_{1,5}^{\infty} f(x)dx$ e esta integral pode ser resolvida numericamente com o comando:

```
> integrate(f2, 1.5, Inf)
0.3749999 with absolute error < 3.5e-05
```

```
1 with absolute error < 1.1e-15
```

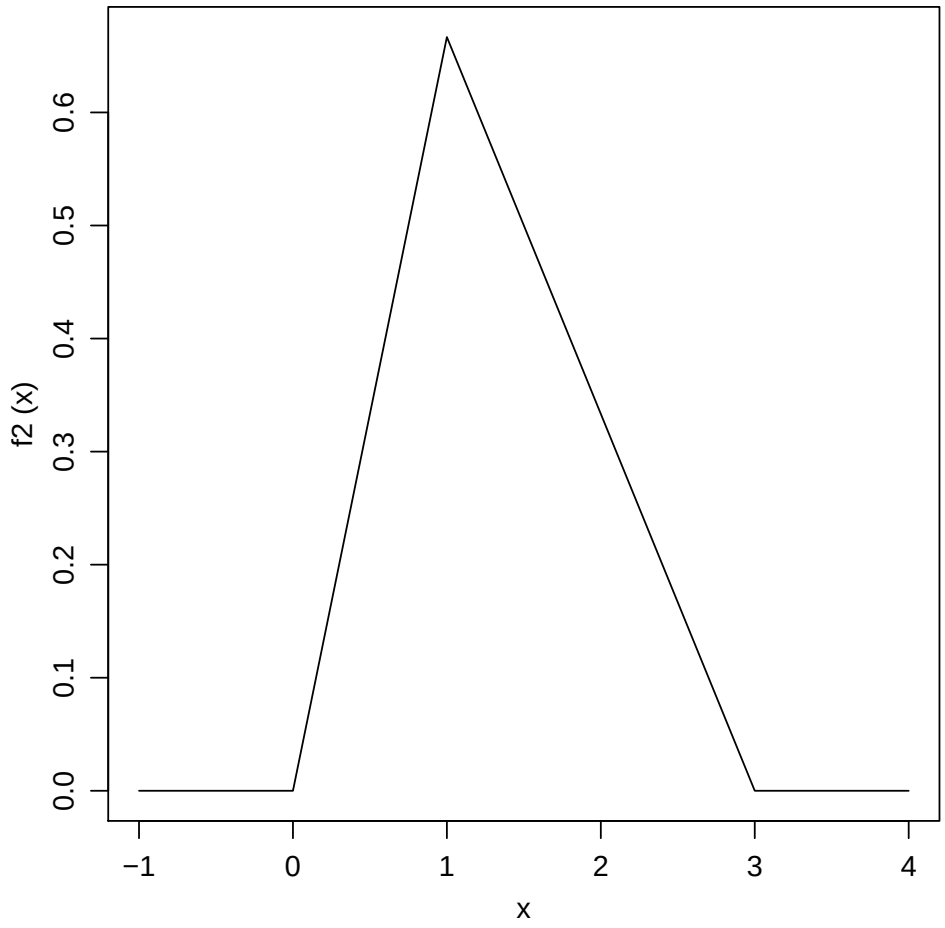


Figura 25: Gráfico da função densidade de probabilidade do Exemplo 2.

A venda esperada em trinta dias é 30 vezes o valor esperado de venda em um dia. Para calcular a esperança $E[X] = \int xf(x)dx$ definimos uma nova função e resolvemos a integral. A função `integrate` retorna uma lista onde um dos elementos (`$value`) é o valor da integral.

```
> ef2 <- function(x) {
+     x * f2(x)
+ }
> integrate(ef2, 0, 3)
1.333333 with absolute error < 7.3e-05
> 30 * integrate(ef2, 0, 3)$value
[1] 40
```

Na questão (c) estamos em busca do quantil 95% da distribuição de probabilidades, ou seja o valor de x que deixa 95% de massa de probabilidade abaixo dele. Este valor que vamos chamar de k é dado por:

$$\int_0^k f(x)dx = 0.95.$$

Para encontrar este valor vamos definir uma função que calcula a diferença (em valor absoluto) entre 0.95 e a probabilidade associada a um valor qualquer de x. O quantil será o valor que minimiza esta

probabilidade. Este é portanto um problema de otimização numérica e para resolvê-lo vamos usar a função `optimize()` do R, que recebe como argumentos a função a ser otimizada e o intervalo no qual deve procurar a solução. A resposta mostra o valor do quantil $x = 2.452278$ e a função objetivo com valor muito próximo de 0, que era o que desejávamos.

```
> f <- function(x) abs(0.95 - integrate(f2, 0, x)$value)
> optimise(f, c(0, 3))
$minimum
[1] 2.452278

$objective
[1] 7.573257e-08
```

A Figura 26 ilustra as soluções dos itens (a) e (c) e os comandos abaixo foram utilizados para obtenção destes gráficos.

```
> par(mfrow = c(1, 2), mar = c(3, 3, 0, 0), mgp = c(2, 1, 0))
> plot(f2, -1, 4)
> polygon(x = c(1.5, 1.5, 3), y = c(0, f2(1.5), 0), dens = 10)
> k <- optimise(f, c(0, 3))$min
> plot(f2, -1, 4)
> polygon(x = c(0, 1, k, k), y = c(0, f2(1), f2(k), 0), dens = 10)
> text(c(1.5, k), c(0.2, 0), c("0.95", "k"), cex = 2.5)
```

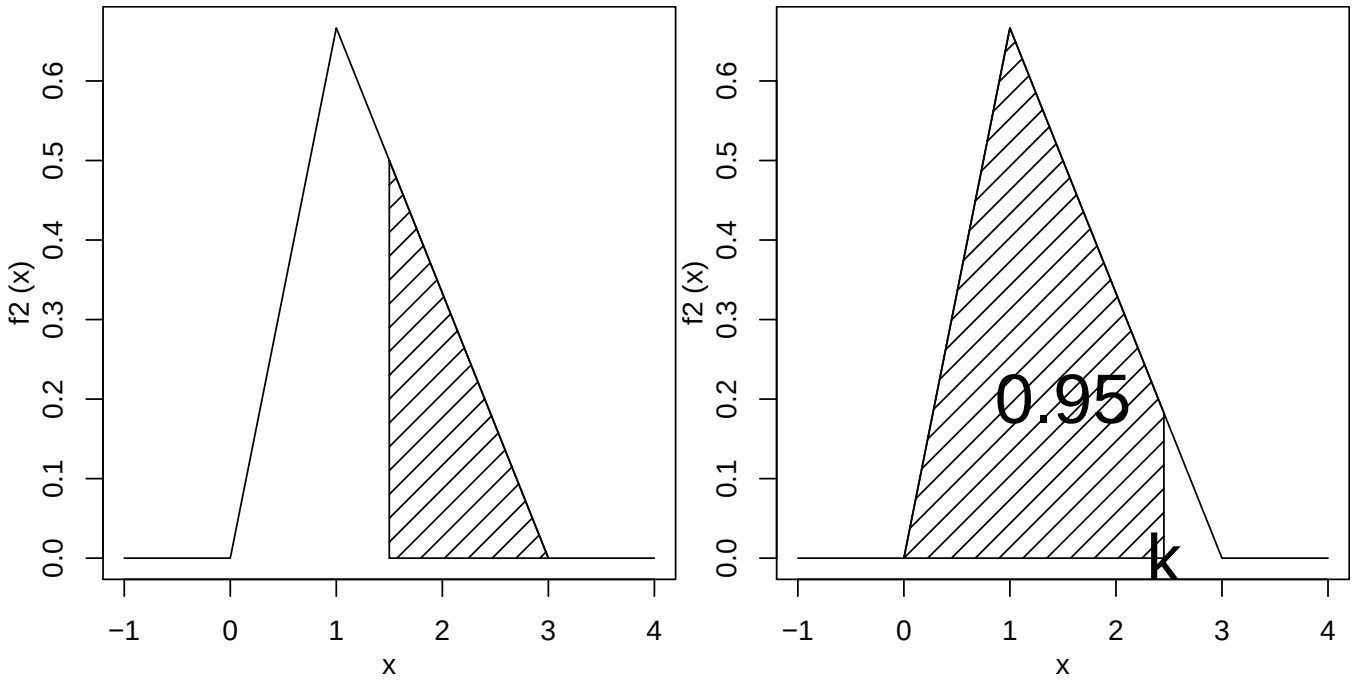


Figura 26: Gráficos indicando as soluções dos itens (a) e (c) do Exemplo 2.

Finalmente lembramos que os exemplos discutidos aqui são simples e não requerem soluções numéricas, devendo ser resolvidos analiticamente. Utilizamos estes exemplos somente para ilustrar a obtenção de soluções numéricas com o uso do R, que na prática deve ser utilizado em problemas mais complexos onde soluções analíticas não são triviais ou mesmo impossíveis.

12.1 Exercícios

1. (Bussab & Morettin, 5a edição, pag. 194, ex. 28)
   Em uma determinada localidade a distribuição de renda, em u.m. (unidade monetária) é uma variável aleatória X com função de distribuição de probabilidade:

$$f(x) = \begin{cases} \frac{1}{10}x + \frac{1}{10} & \text{se } 0 \leq x \leq 2 \\ -\frac{3}{40}x + \frac{9}{20} & \text{se } 2 < x \leq 6 \\ 0 & \text{se } x < 0 \text{ ou } x > 6 \end{cases}$$

   (a) mostre que $f(x)$ é uma f.d.p..

   (b) calcule os quartis da distribuição.

   (c) calcule a probabilidade de encontrar uma pessoa com renda acima de 4,5 u.m. e indique o resultado no gráfico da distribuição.

   (d) qual a renda média nesta localidade?

13 Distribuições de Probabilidade

O programa R inclui funcionalidade para operações com distribuições de probabilidades. Para cada distribuição há 4 operações básicas indicadas pelas letras:

`d` calcula a densidade de probabilidade $f(x)$ no ponto

`p` calcula a função de probabilidade acumulada $F(x)$ no ponto

`q` calcula o quantil correspondente a uma dada probabilidade

`r` retira uma amostra da distribuição

Para usar os funções deve-se combinar uma das letras acima com uma abreviatura do nome da distribuição, por exemplo para calcular probabilidades usamos: `pnorm()` para normal, `pexp()` para exponencial, `pbinom()` para binomial, `ppois()` para Poisson e assim por diante.

Vamos ver com mais detalhes algumas distribuições de probabilidades.

13.1 Distribuição Normal

A funcionalidade para distribuição normal é implementada por argumentos que combinam as letras acima com o termo `norm`. Vamos ver alguns exemplos com a distribuição normal padrão. Por *default* as funções assumem a distribuição normal padrão $N(\mu = 0, \sigma^2 = 1)$.

```
> dnorm(-1)
[1] 0.2419707
> pnorm(-1)
[1] 0.1586553
> qnorm(0.975)
[1] 1.959964
> rnorm(10)
 [1] -0.6340701  0.3019576 -1.5772133 -2.4928096  0.7250672 -1.5212721 -0.1771953
 [8] -0.7318969  0.3789650  0.4376788
```

O primeiro valor acima corresponde ao valor da densidade da normal

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\{-\frac{1}{2\sigma^2}(x-\mu)^2\}$$

com parâmetros $(\mu = 0, \sigma^2 = 1)$ no ponto -1. Portanto, o mesmo valor seria obtido substituindo x por -1 na expressão da normal padrão:

```
> (1/sqrt(2 * pi)) * exp((-1/2) * (-1)^2)
[1] 0.2419707
```

A função `pnorm(-1)` calcula a probabilidade $P(X \leq -1)$. O comando `qnorm(0.975)` calcula o valor de a tal que $P(X \leq a) = 0.975$. Finalmente, o comando `rnorm(10)` gera uma amostra de 10 elementos da normal padrão. Note que os valores que voce obtém rodando este comando podem ser diferentes dos mostrados acima.

As funções acima possuem argumentos adicionais, para os quais valores padrão (*default*) foram assumidos, e que podem ser modificados. Usamos `args()` para ver os argumentos de uma função e `help()` para visualizar a documentação detalhada:

```
> args(rnorm)
function (n, mean = 0, sd = 1)
NULL
```

As funções relacionadas à distribuição normal possuem os argumentos `mean` e `sd` para definir média e desvio padrão da distribuição que podem ser modificados como nos exemplos a seguir. Note nestes exemplos que os argumentos podem ser passados de diferentes formas.

```
> qnorm(0.975, mean = 100, sd = 8)
[1] 115.6797
> qnorm(0.975, m = 100, s = 8)
[1] 115.6797
> qnorm(0.975, 100, 8)
[1] 115.6797
```

Para informações mais detalhadas pode-se usar `help()`. O comando

```
> help(rnorm)
```

irá exibir em uma janela a documentação da função que pode também ser chamada com `?rnorm`. Note que ao final da documentação são apresentados exemplos que podem ser rodados pelo usuário e que auxiliam na compreensão da funcionalidade.
Note também que as 4 funções relacionadas à distribuição normal são documentadas conjuntamente, portanto `help(rnorm)`, `help(qnorm)`, `help(dnorm)` e `help(pnorm)` irão exibir a mesma documentação.

Cálculos de probabilidades usuais, para os quais utilizávamos tabelas estatísticas podem ser facilmente obtidos como no exemplo a seguir.

Seja X uma v.a. com distribuição $N(100, 100)$. Calcular as probabilidades:

1. $P[X < 95]$
2. $P[90 < X < 110]$
3. $P[X > 95]$

Calcule estas probabilidades de forma usual, usando a tabela da normal. Depois compare com os resultados fornecidos pelo R. Os comandos do R para obter as probabilidades pedidas são:

```
> pnorm(95, 100, 10)
[1] 0.3085375
> pnorm(110, 100, 10) - pnorm(90, 100, 10)
[1] 0.6826895
> 1 - pnorm(95, 100, 10)
[1] 0.6914625
> pnorm(95, 100, 10, lower = F)
[1] 0.6914625
```

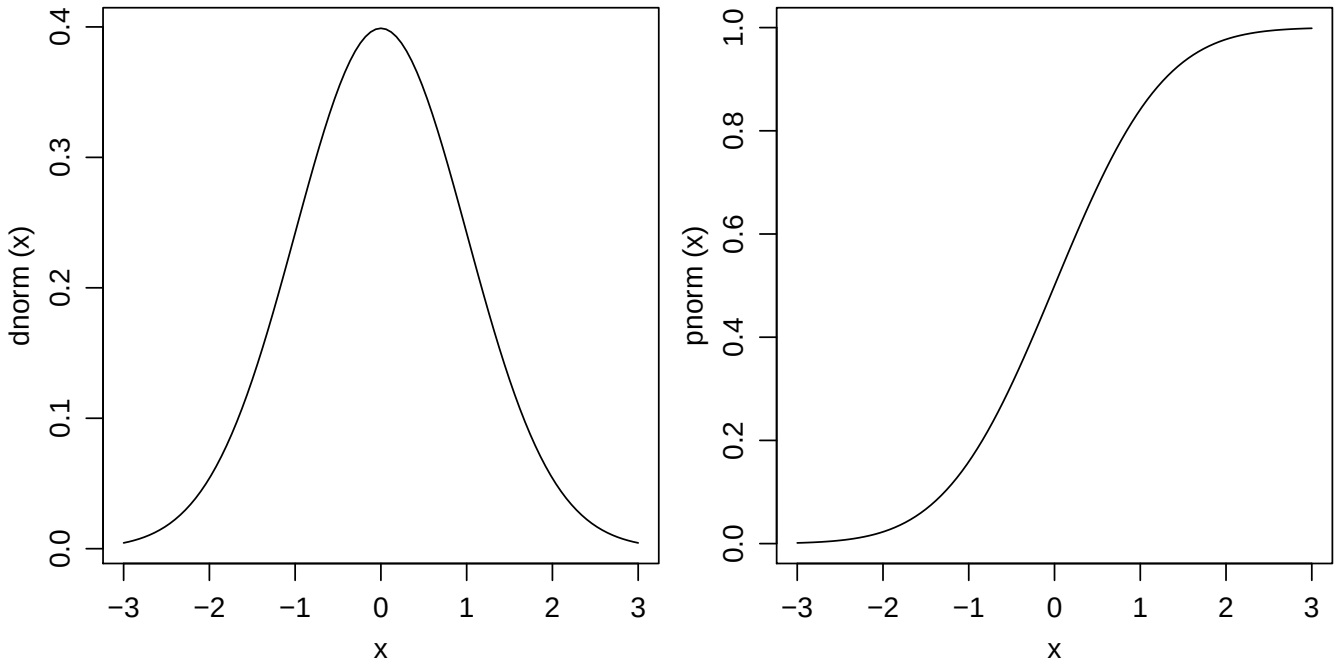


Figura 27: Funções de densidade e probabilidade da distribuição normal padrão.

Note que a última probabilidade foi calculada de duas formas diferentes, a segunda usando o argumento `lower` que implementa um algorítmo de cálculo de probabilidades mais estável numericamente.

A seguir vamos ver comandos para fazer gráficos de distribuições de probabilidade. Vamos fazer gráficos de funções de densidade e de probabilidade acumulada. Estude cuidadosamente os comandos abaixo e verifique os gráficos por eles produzidos. A Figura 27 mostra gráficos da densidade (esquerda) e probabilidade acumulada (direita) da normal padrão, produzidos com os comandos a seguir. Para fazer o gráfico consideramos valores de X entre -3 e 3 que correspondem a +/- três desvios padrões da média, faixa que concentra 99,73% da massa de probabilidade da distribuição normal.

```
> plot(dnorm, -3, 3)
> plot(pnorm, -3, 3)
```

A Figura 28 mostra gráficos da densidade (esquerda) e probabilidade acumulada (direita) da $N(100, 64)$. Para fazer estes gráficos tomamos uma sequência de valores de x entre 70 e 130 e para cada um deles calculamos o valor das funções $f(x)$ e $F(x)$. Depois unimos os pontos $(x, f(x))$ em um gráfico e $(x, F(x))$ no outro.

```
> x <- seq(70, 130, len = 100)
> fx <- dnorm(x, 100, 8)
> plot(x, fx, type = "l")
> Fx <- pnorm(x, 100, 8)
> plot(x, Fx, type = "l")
```

Note que, alternativamente, os mesmos gráficos poderiam ser produzidos com os comandos a seguir.

```
> plot(function(x) dnorm(x, 100, 8), 70, 130)
> plot(function(x) pnorm(x, 100, 8), 70, 130)
```

Comandos usuais do R podem ser usados para modificar a aparência dos gráficos. Por exemplo, podemos incluir títulos e mudar texto dos eixos conforme mostrado na gráfico da esquerda da Figura 29

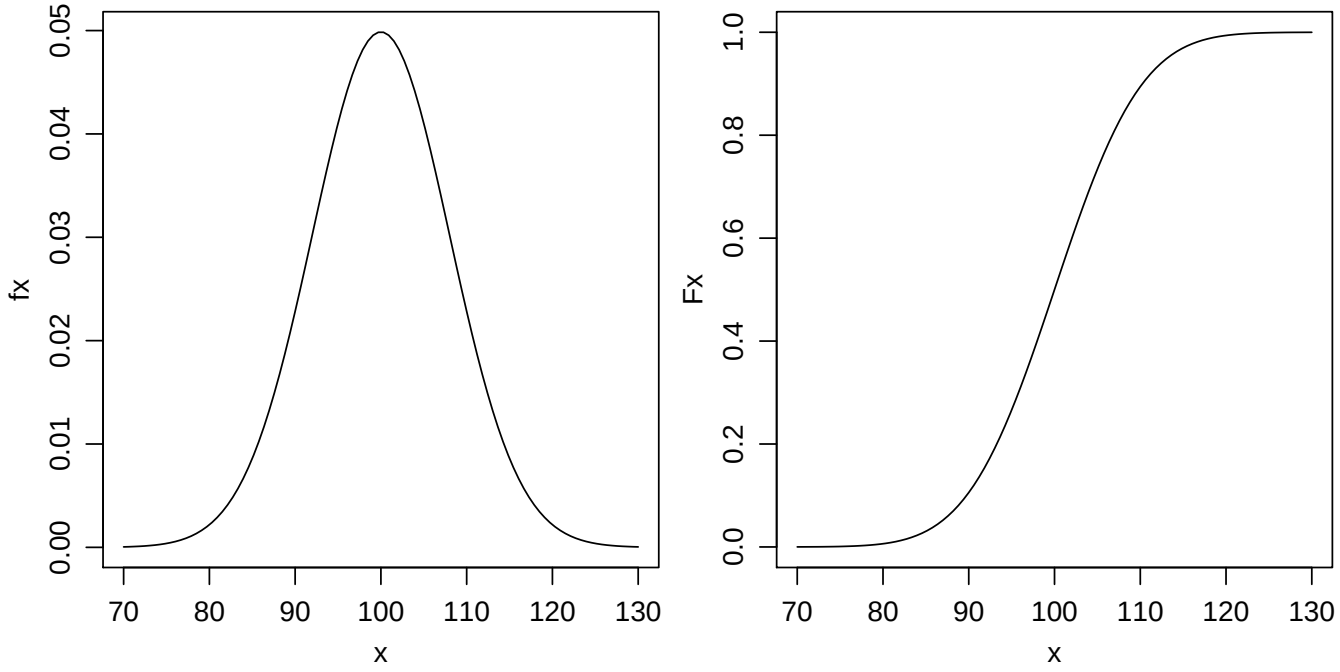


Figura 28: Funções de densidade de probabilidade (esquerda) e função de distribuição acumulada (direita) da $N(100, 64)$.

e nos dois primeiros comandos abaixo. Os demais comandos mostram como colocar diferentes densidades em um mesmo gráfico como ilustrado à direita da mesma Figura.

```
> plot(dnorm, -3, 3, xlab = "valores de X", ylab = "densidade de probabilidade")
> title("Distribuicão Normal\nX ~ N(100, 64)")
> plot(function(x) dnorm(x, 100, 8), 60, 140, ylab = "f(x)")
> plot(function(x) dnorm(x, 90, 8), 60, 140, add = T, col = 2)
> plot(function(x) dnorm(x, 100, 15), 60, 140, add = T, col = 3)
> legend(110, 0.05, c("N(100,64)", "N(90,64)", "N(100,225)"), fill = 1:3)
```

13.2 Distribuição Binomial

Cálculos para a distribuição binomial são implementados combinando as *letras básicas* vistas acima com o termo `binom`. Vamos primeiro investigar argumentos e documentação com `args()` e `dbinom()`.

```
> args(dbinom)
function (x, size, prob, log = FALSE)
NULL
```

```
> help(dbinom)
```

Seja X uma v.a. com distribuição Binomial com $n = 10$ e $p = 0.35$. Vamos ver os comandos do R para:

1. fazer o gráfico das função de densidade
2. idem para a função de probabilidade

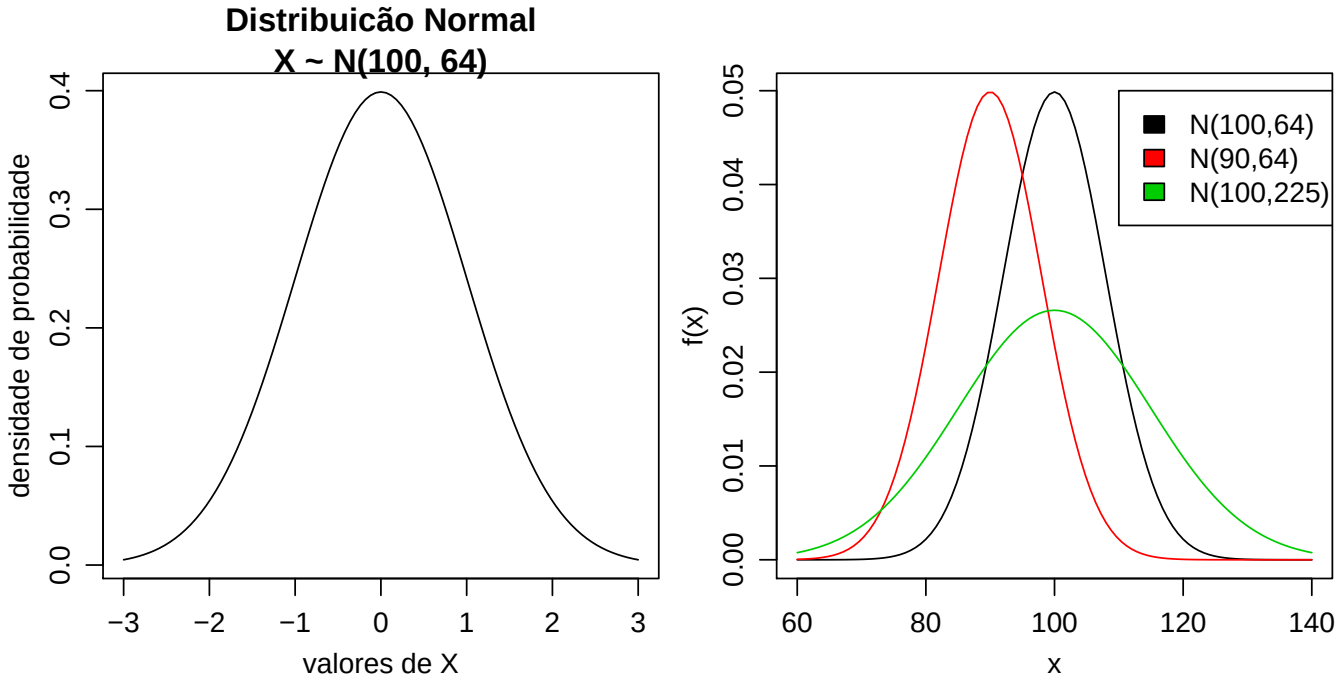


Figura 29: Gráfico com texto nos eixos e título (esquerda) e várias distribuições em um mesmo gráfico (direita).

3. calcular $P[X = 7]$
4. calcular $P[X < 8] = P[X \leq 7]$
5. calcular $P[X \geq 8] = P[X > 7]$
6. calcular $P[3 < X \leq 6] = P[4 \leq X < 7]$

Note que sendo uma distribuição discreta de probabilidades os gráficos são diferentes dos obtidos para distribuição normal e os cálculos de probabilidades devem considerar as probabilidades nos pontos. Os gráficos das funções de densidade e probabilidade são mostrados na Figura 30.

```
> x <- 0:10
> fx <- dbinom(x, 10, 0.35)
> plot(x, fx, type = "h")
> Fx <- pbinom(x, 10, 0.35)
> plot(x, Fx, type = "s")
```

As probabilidades pedidas são obtidas com os comandos a seguir.

```
> dbinom(7, 10, 0.35)
[1] 0.02120302
> pbinom(7, 10, 0.35)
[1] 0.9951787
> sum(dbinom(0:7, 10, 0.35))
[1] 0.9951787
> 1 - pbinom(7, 10, 0.35)
[1] 0.004821265
> pbinom(7, 10, 0.35, lower = F)
```

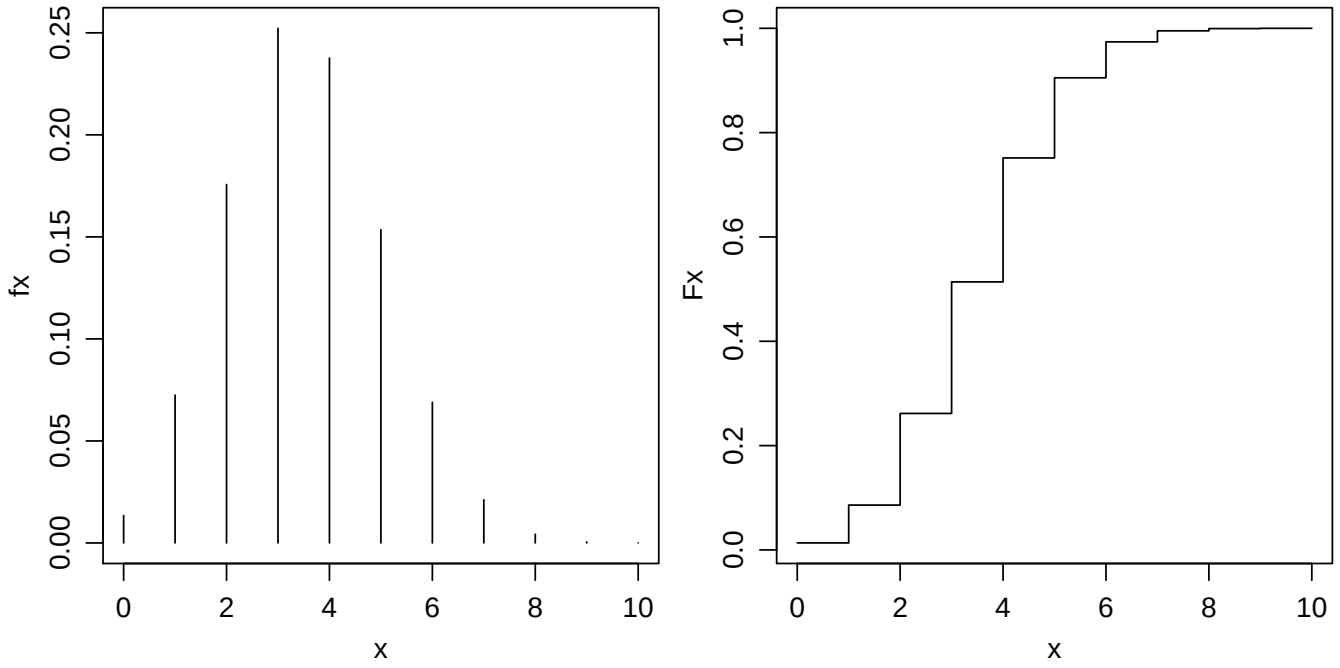


Figura 30: Funções de probabilidade (esquerda) e distribuição acumulada (direita) da $B(10, 0.35)$.

```
[1] 0.004821265
> pbinom(6, 10, 0.35) - pbinom(3, 10, 0.35)
[1] 0.4601487
> sum(dbinom(4:6, 10, 0.35))
[1] 0.4601487
```

13.3 Distribuição Uniforme

13.3.1 Uniforme Contínua

Para a distribuição uniforme *contínua* usa-se as funções `*unif()` onde `*` deve ser p, q, d ou r como mencionado anteriormente. Nos comandos a seguir inspecionamos os argumentos, sorteamos 5 valores da $U(0, 1)$ e calculamos a probabilidade acumulada até 0,75.

```
> args(runif)
function (n, min = 0, max = 1)
NULL
> runif(5)
[1] 0.4887607 0.2191140 0.7390514 0.6375592 0.3148147
> punif(0.75)
[1] 0.75
```

Portanto, o *default* é uma distribuição uniforme no intervalo $[0, 1]$ e os argumentos opcionais são `min` e `max`. Por exemplo, para simular 5 valores de $X \sim U(5, 20)$ usamos:

```
> runif(5, min = 5, max = 20)
[1]  6.443248  8.537205 11.909895  9.028358 15.125244
```

13.3.2 Uniforme Discreta

Não há entre as funções básicas do R uma função específica para a distribuição uniforme discreta com opções de prefixos r, d, p e d, provavelmente devido a sua simplicidade, embora algumas outras funções possam ser usadas. Por exemplo para sortear números pode-se usar `sample()`, como no exemplo a seguir onde são sorteados 15 valores de uma uniforma discreta com valores (inteiros) entre 1 e 10 ($X \sim U_d(1, 10)$).

```
> sample(1:10, 15, rep = T)
 [1]  6 10  3  6 10  4  9  1  3  2  8  6  6  7  8
```

13.4 A função `sample()`

A função `sample()` **não** é restrita à distribuição uniforme discreta, podendo ser usada para sorteios, com ou sem reposição (argumento `replace`, *default* sem reposição), com a possibilidade de associar diferentes probabilidades a cada elemento (argumento `prob`, *default* probabilidades iguais para os elementos).

```
> args(sample)
function (x, size, replace = FALSE, prob = NULL)
NULL
```

Vejamos alguns exemplos:

- sorteio de 3 números entre os inteiros de 0 a 20

  ```
  > sample(0:20, 3)
  [1]  9 13  6
  ```

- sorteio de 5 números entre os elementos de um certo vetor

  ```
  > x <- c(23, 34, 12, 22, 17, 28, 18, 19, 20, 13, 18)
  > sample(x, 5)
  [1] 28 34 19 13 17
  ```

- sorteio de 10 números entre os possíveis resultados do lançamento de um dado, com reposição

  ```
  > sample(1:6, 10, rep = T)
   [1] 2 4 2 5 2 4 2 1 3 5
  ```

- idem ao anterior, porém agora com a probabilidade de cada face proporcional ao valor da face.

  ```
  > sample(1:6, 10, prob = 1:6, rep = T)
   [1] 4 5 5 4 3 6 3 3 1 6
  ```

Este último exemplo ilustra ainda que os valores passados para o argumento `prob` não precisam ser probabilidades, são apenas entendidos como *pesos*. A própria função trata isto internamente fazendo a ponderação adequada.

13.5 Exercícios

Nos exercícios abaixo iremos também usar o R como uma calculadora estatística para resolver alguns exemplos/exercícios de probabilidade tipicamente apresentados em um curso de estatística básica.

Os exercícios abaixo com indicação de página foram retirados de:

Magalhães, M.N. & Lima, A.C.P. (2001) **Noções de Probabilidade e Estatística**. 3 ed. São Paulo, IME-USP. 392p.

1. (Ex 1, pag 67) Uma moeda viciada tem probabilidade de cara igual a 0.4. Para quatro lançamentos independentes dessa moeda, estude o comportamento da variável *número de caras* e faça um gráfico de sua função de distribuição.

2. (Ex 5, pag 77) Sendo X uma variável seguindo o modelo Binomial com parâmetro $n = 15$ e $p = 0.4$, pergunta-se:

   - $P(X \geq 14)$
   - $P(8 < X \leq 10)$
   - $P(X < 2 \text{ ou } X \geq 11)$
   - $P(X \geq 11 \text{ ou } X > 13)$
   - $P(X > 3 \text{ e } X < 6)$
   - $P(X \leq 13 \mid X \geq 11)$

3. (Ex 8, pag 193) Para $X \sim N(90, 100)$, obtenha:

   - $P(X \leq 115)$
   - $P(X \geq 80)$
   - $P(X \leq 75)$
   - $P(85 \leq X \leq 110)$
   - $P(|X - 90| \leq 10)$
   - O valor de a tal que $P(90 - a \leq X \leq 90 + a) = \gamma,\ \gamma = 0.95$

4. Faça os seguintes gráficos:

   - da função de densidade de uma variável com distribuição de Poisson com parâmetro $\lambda = 5$
   - da densidade de uma variável $X \sim N(90, 100)$
   - sobreponha ao gráfico anterior a densidade de uma variável $Y \sim N(90, 80)$ e outra $Z \sim N(85, 100)$
   - densidades de distribuições χ^2 com 1, 2 e 5 graus de liberdade.

5. A probabilidade de indivíduos nascerem com certa característica é de 0,3. Para o nascimento de 5 indivíduos e considerando os nascimentos como eventos independentes, estude o comportamento da variável *número de indivíduos com a característica* e faça um gráfico de sua função de distribuição.

Resistência	2	3	4	5	6
pi	0,1	0,1	0,4	0,2	0,2

6. Sendo X uma variável seguindo o modelo Normal com média $\mu = 130$ e variância $\sigma^2 = 64$, pergunta-se: (a) $P(X \geq 120)$ (b) $P(135 < X \leq 145)$ (c) $P(X < 120 \text{ ou } X \geq 150)$

7. (Ex 3.6, pag 65) Num estudo sobre a incidência de câncer foi registrado, para cada paciente com este diagnóstico o número de casos de câncer em parentes próximos (pais, irmãos, tios, filhos e sobrinhos). Os dados de 26 pacientes são os seguintes:

Paciente	1	2	3	4	5	6	7	8	9	10	11	12	13
Incidência	2	5	0	2	1	5	3	3	3	2	0	1	1
Paciente	14	15	16	17	18	19	20	21	22	23	24	25	26
Incidência	4	5	2	2	3	2	1	5	4	0	0	3	3

   Estudos anteriores assumem que a incidência de câncer em parentes próximos pode ser modelada pela seguinte função discreta de probabilidades:

Incidência	0	1	2	3	4	5
p_i	0.1	0.1	0.3	0.3	0.1	0.1

   - os dados observados concordam com o modelo teórico?
   - faça um gráfico mostrando as frequências teóricas (esperadas) e observadas.

8. A distribuição da soma de duas variáveis aleatórias uniformes não é uniforme. Verifique isto gerando dois vetores x e y com distribuição uniforme $[0, 1]$ com 3000 valores cada e fazendo $z = x + y$. Obtenha o histograma para x, y e z. Descreva os comandos que utilizou.

9. (extraído de Magalhães e Lima, 2001) A resistência (em toneladas) de vigas de concreto produzidas por uma empresa, comporta-se como abaixo:

   Simule a resistência de 5000 vigas a partir de valores gerados de uma uniforme [0,1]. (Dica: Use o comando `ifelse()` do R). Verifique o histograma.

14 Complementos sobre distribuições de probabilidade

Agora que já nos familiarizamos com o uso das distribuições de probabilidade vamos ver alguns detalhes adicionais sobre seu funcionamento.

14.1 Probabilidades e integrais

A probabilidade de um evento em uma distribuição contínua é uma área sob a curva da distribuição. Vamos reforçar esta idéia revisitando um exemplo visto na aula anterior.

Seja X uma v.a. com distribuição $N(100, 100)$. Para calcular a probabilidade $P[X < 95]$ usamos o comando:

```
> pnorm(95, 100, 10)
[1] 0.3085375
```

Vamos agora "esquecer" o comando `pnorm()` e ver uma outra forma de resolver usando integração numérica. Lembrando que a normal tem a função de densidade dada por

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\{-\frac{1}{2\sigma^2}(x-\mu)^2\}$$

vamos definir uma função no R para a densidade normal deste problema:

```
> fn <- function(x) {
+     fx <- (1/sqrt(2 * pi * 100)) * exp((-1/200) * (x - 100)^2)
+     return(fx)
+ }
```

Para obter o gráfico desta distribuição mostrado na Figura 31 usamos o fato que a maior parte da função está no intervalo entre a média +/- três desvios padrões, portanto entre 70 e 130. Podemos então fazer como nos comandos que se seguem. Para marcar no gráfico a área que corresponde a probabilidade pedida criamos um polígono com coordenadas `ax` e `ay` definindo o perímetro desta área.

```
> x <- seq(70, 130, l = 200)
> fx <- fn(x)
> plot(x, fx, type = "l")
> ax <- c(70, 70, x[x < 95], 95, 95)
> ay <- c(0, fn(70), fx[x < 95], fn(95), 0)
> polygon(ax, ay, dens = 10)
```

Para calcular a área pedida sem usar a função `pnorm()` podemos usar a função de integração numérica. Note que esta função, diferentemente da `pnorm()` reporta ainda o erro de aproximação numérica.

```
> integrate(fn, -Inf, 95)
0.3085375 with absolute error < 2.1e-06
```

Portanto para os demais ítens do problema $P[90 < X < 110]$ e $P[X > 95]$ fazemos:

```
> integrate(fn, 90, 110)
0.6826895 with absolute error < 7.6e-15
> integrate(fn, 95, +Inf)
0.6914625 with absolute error < 8.1e-05
```

e os resultados acima evidentemente coincidem com os obtidos anterioriormente usando `pnorm()`.

Note ainda que na prática não precisamos definir e usar a função fn pois ela fornece o mesmo resultado que a função `dnorm()`.

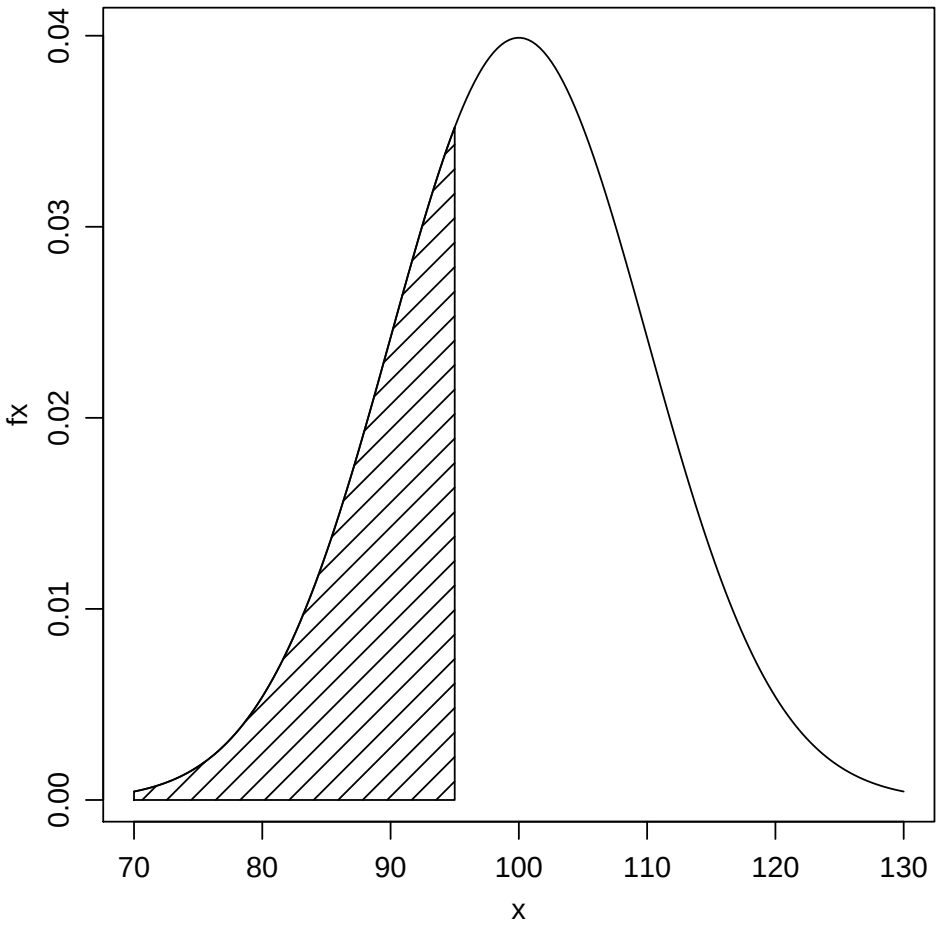


Figura 31: Funções de densidade da $N(100, 100)$ com a área correspondente à $P[X \leq 95]$.

14.2 Distribuição exponencial

A função de densidade de probabilidade da distribuição exponencial com parâmetro λ e denotada $Exp(\lambda)$ é dada por:

$$f(x) = \begin{cases} \frac{1}{\lambda}\, e^{-x/\lambda} & \text{para } x \geq 0 \\ 0 & \text{para } x < 0 \end{cases}$$

Seja uma variável X com distribuição exponencial de parâmetro $\lambda = 500$. Calcular a probabilidade $P[X \geq 400]$.

A solução analítica pode ser encontrada resolvendo

$$P[X \geq 400] = \int_{400}^{\infty} f(x)dx = \int_{400}^{\infty} \frac{1}{\lambda}\, e^{-x/\lambda} dx$$

que é uma integral que pode ser resolvida analiticamente. Fica como exercício encontrar o valor da integral acima.

Para ilustrar o uso do R vamos também obter a resposta usando integração numérica. Para isto vamos criar uma função com a expressão da exponencial e depois integrar no intervalo pedido e este resultado deve ser igual ao encontrado com a solução analítica.

```
> fexp <- function(x, lambda = 500) {
+     fx <- ifelse(x < 0, 0, (1/lambda) * exp(-x/lambda))
+     return(fx)
```

```
+ }
> integrate(fexp, 400, Inf)
0.449329 with absolute error < 5e-06
```

Note ainda que poderíamos obter o mesmo resultado simplesmente usando a função `pexp()` com o comando `pexp(400, rate=1/500, lower=F)`, onde o argumento corresponde a $1/\lambda$ na equação da exponencial.

A Figura 32 mostra o gráfico desta distribuição com indicação da área correspondente à probabilidade pedida. Note que a função é positiva no intervalo $(0, +\infty)$ mas para fazer o gráfico consideramos apenas o intervalo $(0, 2000)$.

```
> x <- seq(0, 2000, l = 200)
> fx <- dexp(x, rate = 1/500)
> plot(x, fx, type = "l")
> ax <- c(400, 400, x[x > 400], 2000, 2000)
> ay <- c(0, dexp(c(400, x[x > 400], 2000), 1/500), 0)
> polygon(ax, ay, dens = 10)
```

14.3 Esperança e Variância

Sabemos que para a distribuição exponencial a esperança $E[X] = \int_0^\infty x f(x)dx = \lambda$ e a variância $Var[X] = \int_0^\infty (x - E[X])^2 f(x)dx = \lambda^2$ pois podem ser obtidos analiticamente.

Novamente para ilustrar o uso do R vamos "esquecer" que conhecemos estes resultados e vamos obtê-los numericamente. Para isto vamos definir funções para a esperança e variância e fazer a integração numérica.

```
> e.exp <- function(x, lambda = 500) {
+     ex <- x * (1/lambda) * exp(-x/lambda)
+     return(ex)
+ }
> integrate(e.exp, 0, Inf)
500 with absolute error < 0.00088
> ex <- integrate(e.exp, 0, Inf)$value
> ex
[1] 500
> v.exp <- function(x, lambda = 500, exp.x) {
+     vx <- ((x - exp.x)^2) * (1/lambda) * exp(-x/lambda)
+     return(vx)
+ }
> integrate(v.exp, 0, Inf, exp.x = ex)
250000 with absolute error < 6.9
```

14.4 Gerador de números aleatórios

A geração da amostra depende de um *gerador de números aleatórios* que é controlado por uma *semente* (*seed* em inglês). Cada vez que o comando `rnorm()` é chamado diferentes elementos da amostra são produzidos, porque a *semente* do gerador é automaticamente modificada pela função. Em geral o usuário não precisa se preocupar com este mecanismo. Mas caso necessário `set.seed()`

```
> x <- seq(0, 2000, l = 200)
> fx <- dexp(x, rate = 1/500)
> plot(x, fx, type = "l")
> ax <- c(400, 400, x[x > 400], 2000, 2000)
> ay <- c(0, dexp(c(400, x[x > 400], 2000), 1/500), 0)
> polygon(ax, ay, dens = 10)
```

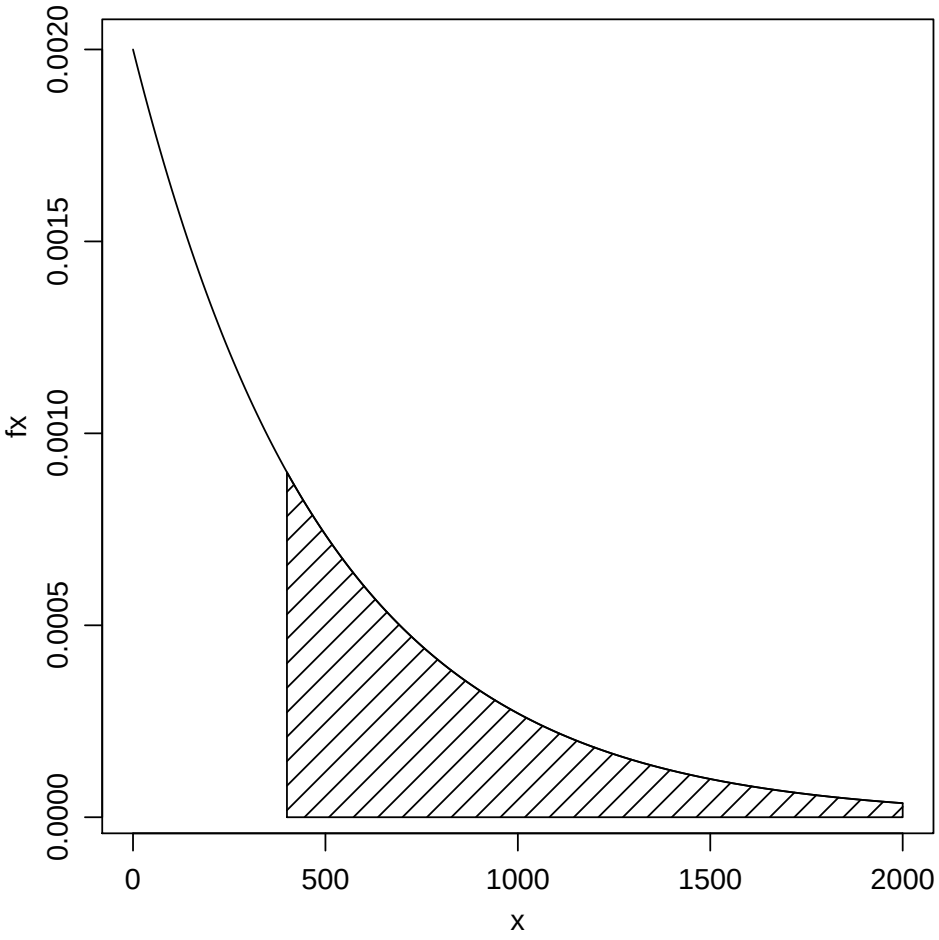


Figura 32: Função de densidade da $Exp(500)$ com a área correspondente à $P[X \geq 400]$.

pode ser usada para controlar o comportamento do gerador de números aleatórios. Esta função define o valor inicial da semente que é mudado a cada geração subsequente de números aleatórios. Portanto para gerar duas amostras idênticas basta usar `set.seed()` conforme ilustrado abaixo.

```
> set.seed(214)
> rnorm(5)
[1] -0.46774980  0.04088223  1.00335193  2.02522505  0.30640096
> rnorm(5)
[1]  0.4257775  0.7488927  0.4464515 -2.2051418  1.9818137
> set.seed(214)
> rnorm(5)
[1] -0.46774980  0.04088223  1.00335193  2.02522505  0.30640096
```

Nos comandos acima mostramos que depois da primeira amostra ser retirada a semente é mudada e por isto os elementos da segunda amostra são diferentes dos da primeira. Depois retornamos a semente ao seu estado original a a próxima amostra tem portanto os mesmos elementos da primeira.

Para saber mais sobre geração de números aleatórios no R veja |help(.Random.seed)| e |help(set.seed)|

14.5 Argumentos vetoriais e lei da reciclagem

As funções de probabilidades aceitam também vetores em seus argumentos conforme ilustrado nos exemplo abaixo.

```
> qnorm(c(0.05, 0.95))
[1] -1.644854  1.644854
> rnorm(4, mean = c(0, 10, 100, 1000))
[1]   0.4257775  10.7488927 100.4464515 997.7948582
> rnorm(4, mean = c(10, 20, 30, 40), sd = c(2, 5))
[1] 13.963627  6.872238 28.553964 35.584654
```

Note que no último exemplo a *lei da reciclagem* foi utilizada no vetor de desvios padrão, i.e. os desvios padrão utilizados foram $(2, 5, 2, 5)$.

14.6 Aproximação pela Normal

Nos livros texto de estatística podemos ver que as distribuições binomial e Poisson podem ser aproximadas pela normal. Isto significa que podemos usar a distribuição normal para calcular probabilidades *aproximadas* em casos em que seria "trabalhoso" calcular as probabilidades *exatas* pela binomial ou Poisson. Isto é especialmente importante no caso de usarmos calculadoras e/ou tabelas para calcular probabilidades. Quando usamos um computador esta aproximação é menos importante, visto que é fácil calcular as probabilidades exatas com o auxílio do computador. De toda forma vamos ilustrar aqui este resultado.

Vejamos como fica a aproximação no caso da distribuição binomial. Seja $X \sim B(n, p)$. Na prática, em geral a aproximação é considerada aceitável quando $np \geq 5$ e $n(1-p) \geq 5$ e sendo tanto melhor quanto maior for o valor de n. A aproximação neste caso é de que $X \sim B(n, p) \approx N(np, np(1-p))$.

Seja $X \sim B(10, 1/2)$ e portanto com a aproximação $X \approx N(5, 2.5)$. A Figura 33 mostra o gráfico da distribuição binomial e da aproximação pela normal.

```
> xb <- 0:10
> px <- dbinom(xb, 10, 0.5)
> plot(xb, px, type = "h")
> xn <- seq(0, 10, len = 100)
> fx <- dnorm(xn, 5, sqrt(2.5))
> lines(xn, fx)
```

Vamos também calcular as seguintes probabilidades exatas e aproximadas, lembrando que ao usar a aproximação pela normal devemos usar a correção de continuidade e/ou somando e subtraindo 0.5 ao valor pedido.

- $P[X < 6]$
  Neste caso $P[X_B < 6] = P[X_B \leq 5] \approx P[X_N \leq 5.5]$

  ```
  > pbinom(5, 10, 0.5)
  ```

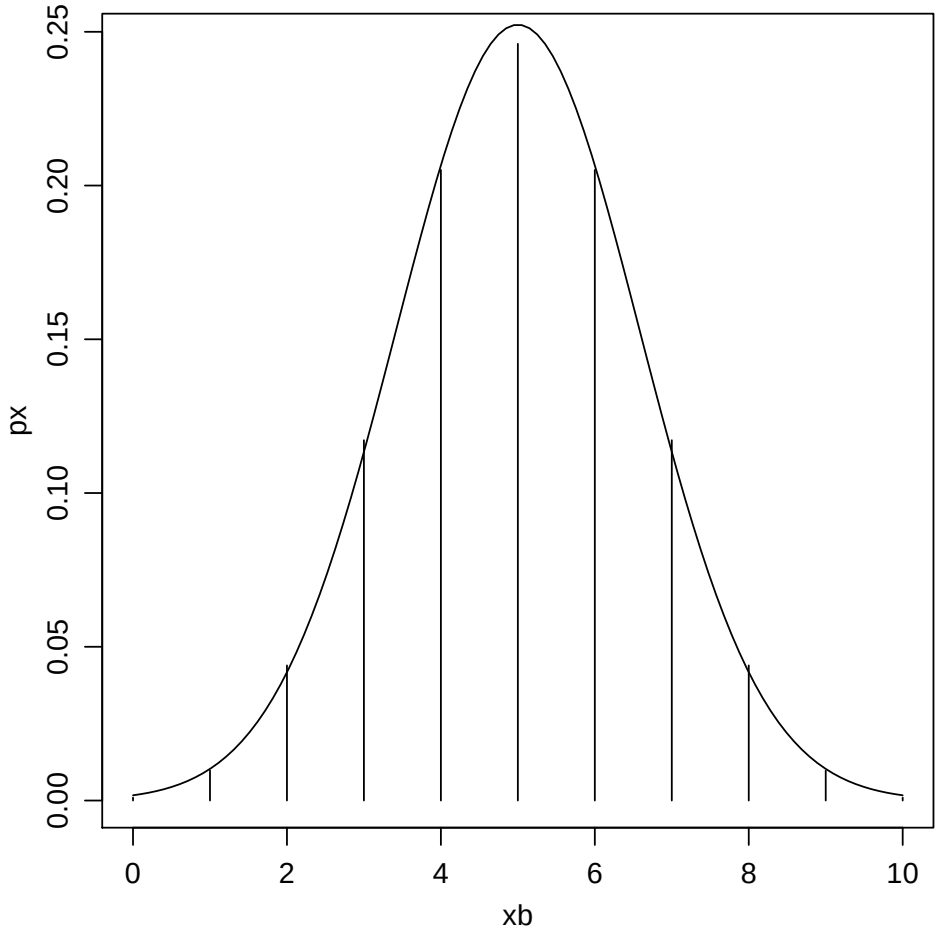


Figura 33: Função de probabilidade da $B(10, 1/2)$ e a aproximação pela $N(5, 2.5)$.

```
[1] 0.6230469
> pnorm(5.5, 5, sqrt(2.5))
[1] 0.6240852
```

- $P[X \leq 6]$
  Neste caso $P[X_B \leq 6] \approx P[X_N \leq 6.5]$

```
> pbinom(6, 10, 0.5)
[1] 0.828125
> pnorm(6.5, 5, sqrt(2.5))
[1] 0.8286091
```

- $P[X > 2]$
  Neste caso $P[X_B > 2] = 1 - P[X_B \leq 2] \approx 1 - P[X_N \leq 2.5]$

```
> 1 - pbinom(2, 10, 0.5)
[1] 0.9453125
```

```
> 1 - pnorm(2.5, 5, sqrt(2.5))
[1] 0.9430769
```

- $P[X \geq 2]$
  Neste caso $P[X_B \geq 2] = 1 - P[X_B \leq 1] \approx P[X_N \leq 1.5]$

```
> 1 - pbinom(1, 10, 0.5)
[1] 0.9892578
> 1 - pnorm(1.5, 5, sqrt(2.5))
[1] 0.9865717
```

- $P[X = 7]$
  Neste caso $P[X_B = 7] \approx P[6.5 \leq X_N \leq 7.5]$

```
> dbinom(7, 10, 0.5)
[1] 0.1171875
> pnorm(7.5, 5, sqrt(2.5)) - pnorm(6.5, 5, sqrt(2.5))
[1] 0.1144677
```

- $P[3 < X \leq 8]$
  Neste caso $P[3 < X_B \leq 8] = P[X_B \leq 8] - P[X_B \leq 3] \approx P[X_N \leq 8.5] - P[X_N \leq 3.5]$

```
> pbinom(8, 10, 0.5) - pbinom(3, 10, 0.5)
[1] 0.8173828
> pnorm(8.5, 5, sqrt(2.5)) - pnorm(3.5, 5, sqrt(2.5))
[1] 0.8151808
```

- $P[1 \leq X \leq 5]$
  Neste caso $P[1 \leq X_B \leq 5] = P[X_B \leq 5] - P[X_B \leq 0] \approx P[X_N \leq 5.5] - P[X_N \leq 0.5]$

```
> pbinom(5, 10, 0.5) - pbinom(0, 10, 0.5)
[1] 0.6220703
> pnorm(5.5, 5, sqrt(2.5)) - pnorm(0.5, 5, sqrt(2.5))
[1] 0.6218719
```

14.7 Exercícios

1. (Bussab & Morettin, pag. 198, ex. 51)
   A função de densidade de probabilidade de distribuição Weibull é dada por:

$$f(x) = \begin{cases} \lambda\, x^{\lambda-1}\, e^{-x^{\lambda}} & \text{para } x \geq 0 \\ 0 & \text{para } x < 0 \end{cases}$$

   (a) Obter $E[X]$ para $\lambda = 2$. Obter o resultado analitica e computacionalmente.
   Dica: para resolver você vai precisar da definição da função Gama:

$$\Gamma(a) = \int_0^{\infty} x^{a-1}\, e^{-x} dx$$

   (b) Obter $E[X]$ para $\lambda = 5$.

   (c) Obter as probabilidades:

   - $P[X > 2]$
   - $P[1.5 < X < 6]$
   - $P[X < 8]$

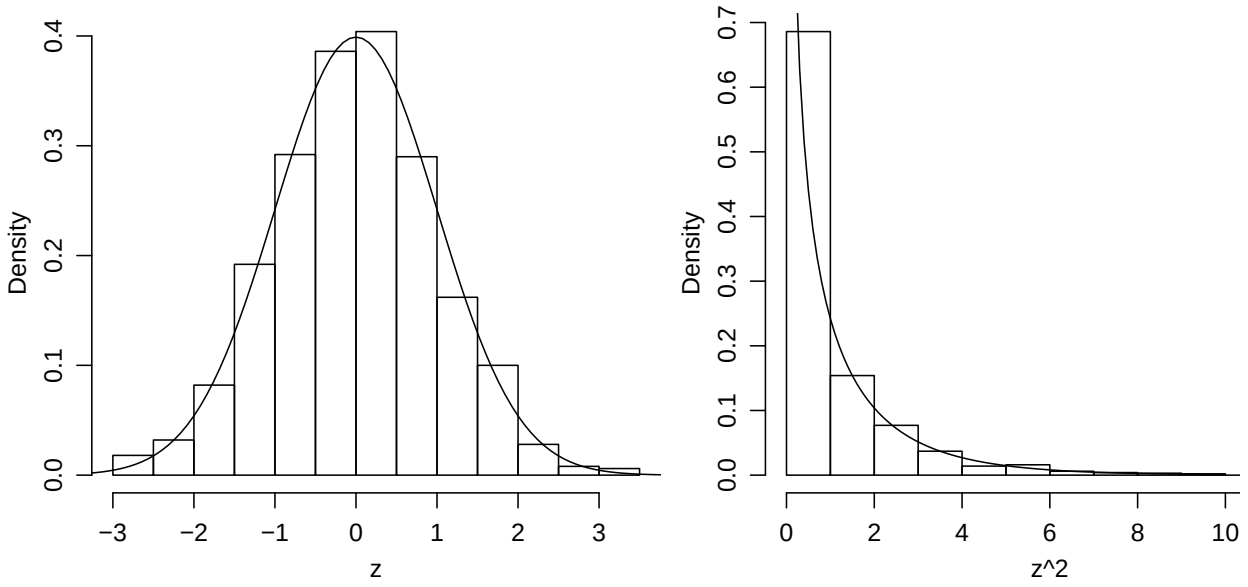


Figura 34: Histograma das amostra da e a curva teórica da distribuição normal padrão (esquerda) e histograma dos valores ao quadrado com a curva teórica da distribuição $\chi^2_{(1)}$ (direita).

15 Usando simulação para ilustrar resultados

Podemos utilizar recursos computacionais e em particular simulações para inferir distribuições amostrais de quantidades de interesse. Na teoria de estatística existem vários resultados que podem ser ilustrados via simulação, o que ajuda na compreensão e visualização dos conceitos e resultados. Veremos alguns exemplos a seguir.

Este uso de simulações é apenas um ponto de partida pois estas são especialmente úteis para explorar situações onde resultados teóricos não são conhecidos ou não podem ser obtidos.

15.1 Relações entre a distribuição normal e a χ^2

Resultado 1: Se $Z \sim \mathrm{N}(0,1)$ então $Z^2 \sim \chi^2_{(1)}$.

Vejamos como ilustrar este resultado. Inicialmente vamos definir o valor da semente de números aleatórios para que os resultados possam ser reproduzidos. Vamos começar gerando uma amostra de 1000 números da distribuição normal padrão. A seguir vamos fazer um histograma dos dados obtidos e sobrepor a curva da distribuição teórica. Fazemos isto com os comando abaixo e o resultado está no gráfico da esquerda da Figura 34.

```
> z <- rnorm(1000)
> hist(z, prob = T, main = "")
> curve(dnorm(x), -4, 4, add = T)
```

Note que, para fazer a comparação do histograma e da curva teórica é necessário que o histograma seja de frequências relativas e para isto usamos o argumento `prob = T`.

Agora vamos estudar o comportamento do quadrado da variável. O gráfico da direita da Figura 34 mostra o histograma dos quadrados do valores da amostra e a curva da distribuição de $\chi^2_{(1)}$.

```
> hist(z^2, prob = T, main = "")
> curve(dchisq(x, df = 1), 0, 10, add = T)
```

Nos gráficos anteriores comparamos o histograma da distribuição empírica obtida por simulação com a curva teórica da distribuição. Uma outra forma e mais eficaz forma de comparar distribuições

empíricas e teóricas é comparar os quantis das distribuições e para isto utilizamos o *qq-plot*. O *qq-plot* é um gráfico dos dados ordenados contra os quantis esperados de uma certa distribuição. Quanto mais próximo os pontos estiverem da bissetriz do primeiro quadrante mais próximos os dados observados estão da distribuição considerada. Portanto para fazer o `qqplot` seguimos os passos:

1. obter os dados,
2. obter os quantis da distribuição teórica,
3. fazer um gráfico dos dados ordenados contra os quantis da distribuição.

Vamos ilustrar isto nos comandos abaixo. Primeiro vamos considerar como dados os quadrados da amostra da normal obtida acima. Depois obtemos os quantis teóricos da distribução χ^2 usando a função `qchisq` em um conjunto de probabilidades geradas pela função `ppoints`. Por fim usamos a função `qqplot` para obter o gráfico mostrado na Figura 35, adicionando neste gráfico a bissetriz do primeiro quadrante para facilitar a avaliação do ajuste.

```
> quantis <- qchisq(ppoints(length(z)), df = 1)
> qqplot(quantis, z^2)
> abline(0, 1)
```

Note que o comando `qchisq(ppoints(length(z)), df=1)` acima está concatenando 3 comandos e calcula os quantis da χ^2 a partir de uma sequência de valores de probabilidade gerada por `ppoints`. O número de elementos desta sequência deve igual ao número de dados e por isto usamos `length(z)`.

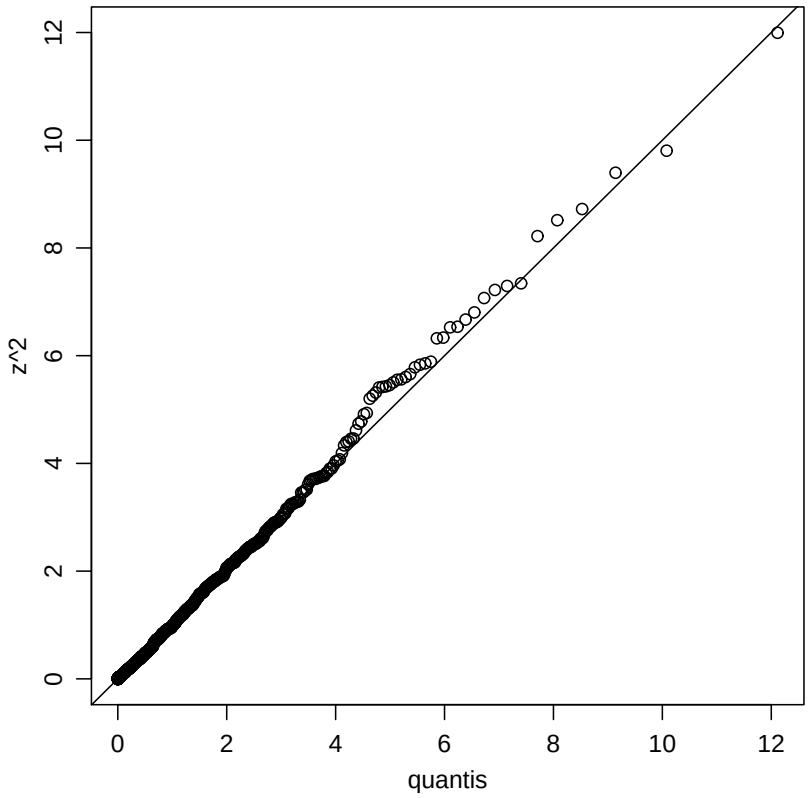


Figura 35: Comparando dados e quantis da χ^2 utilizando o *qq-plot*

Resultado 2: Se $Z_1, Z_2, \ldots Z_n \sim \mathrm{N}(0,1)$ então $\sum_1^n Z_i^2 \sim \chi^2_{(n)}$.

Para ilustrar este resultado vamos gerar 10.000 amostras de 3 elementos cada da distribuição normal padrão, elevar os valores ao quadrado e, para cada amostra, somar os quadrados dos três

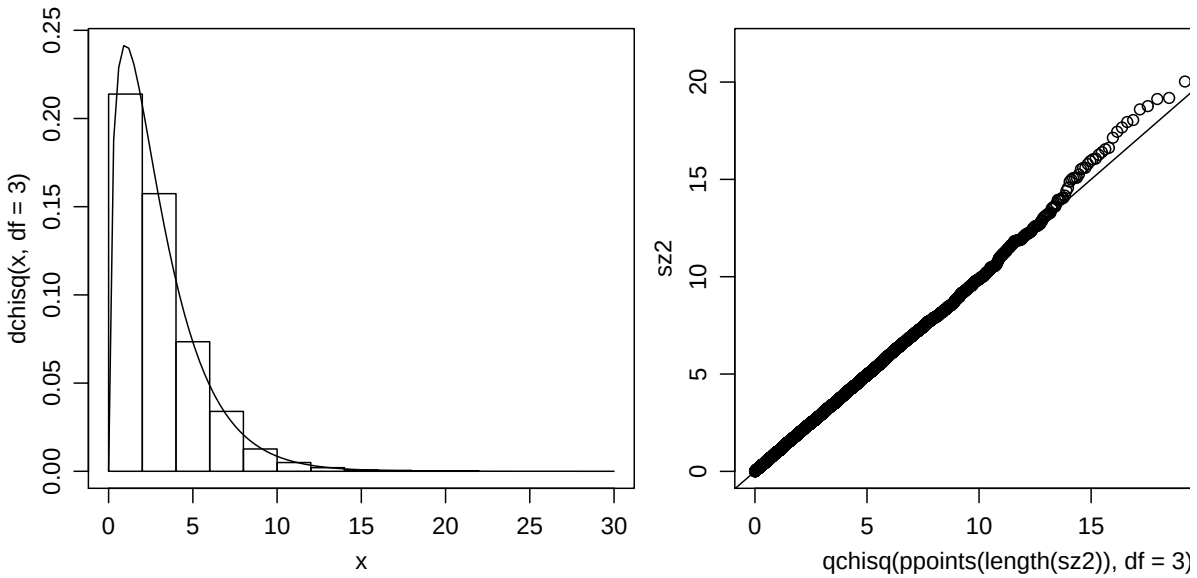


Figura 36: Histograma da uma amostra da soma dos quadrados de três valores da normal padrão e a curva teórica da distribuição de $\chi^2_{(3)}$ (esquerda) e o respectivo *qq-plot*.

números. Na Figura 36 mostramos no gráfico à esquerda, o histograma dos valores obtidos com a curva da distribuição esperada e no da direita o *qq-plot* para a distribuição $\chi^2_{(3)}$.

```
> set.seed(23)
> z <- matrix(rnorm(30000), nc = 3)
> sz2 <- apply(z^2, 1, sum)
> par(mfrow = c(1, 2))
> curve(dchisq(x, df = 3), 0, 30)
> hist(sz2, prob = T, main = "", add = T)
> qqplot(qchisq(ppoints(length(sz2)), df = 3), sz2)
> abline(0, 1)
```

15.2 Distribuição amostral da média de amostras da distribuição normal

Resultado 3: Se $Y_1, Y_2, \ldots Y_n \sim \mathrm{N}(\mu, \sigma^2)$ então $\bar{y} \sim \mathrm{N}(\mu, \sigma^2/n)$.

Neste exemplo vamos obter 1000 amostras de tamanho 20 de uma distribuição normal de média 100 e variância 30. Vamos organizar as amostras em uma matriz onde cada coluna corresponde a uma amostra. A seguir vamos calcular a média de cada amostra.

```
> set.seed(381)
> y <- matrix(rnorm(20000, mean = 100, sd = sqrt(30)), nc = 1000)
> ybar <- apply(y, 2, mean)
> mean(ybar)
[1] 99.9772
> var(ybar)
[1] 1.678735
```

Pelo **Resultado 3** acima esperamos que a média das médias amostrais seja 100 e a variância seja 1.5 (= 30/20), e que a distribuição das médias amostrais seja normal, valores bem próximos dos obtidos acima, sendo que as diferenças são devidas ao erro de simulação pro trabalharmos com amostras de

tamanho finito. Para completar vamos obter o gráfico com o histograma das médias das amostras e a distribuição teórica conforme Figura 37 e o respectivo `qq-plot`.

```
> par(mfrow = c(1, 2))
> curve(dnorm(x, mean = 100, sd = sqrt(30/20)), 95, 105)
> hist(ybar, prob = T, add = T)
> qqnorm(ybar)
> qqline(ybar)
```

Note que para obter o *qq-plot* neste exemplo utilizamos as funções `qqnorm` `qqline` já disponíveis no R para fazer *qq-plot* para distribuição normal.

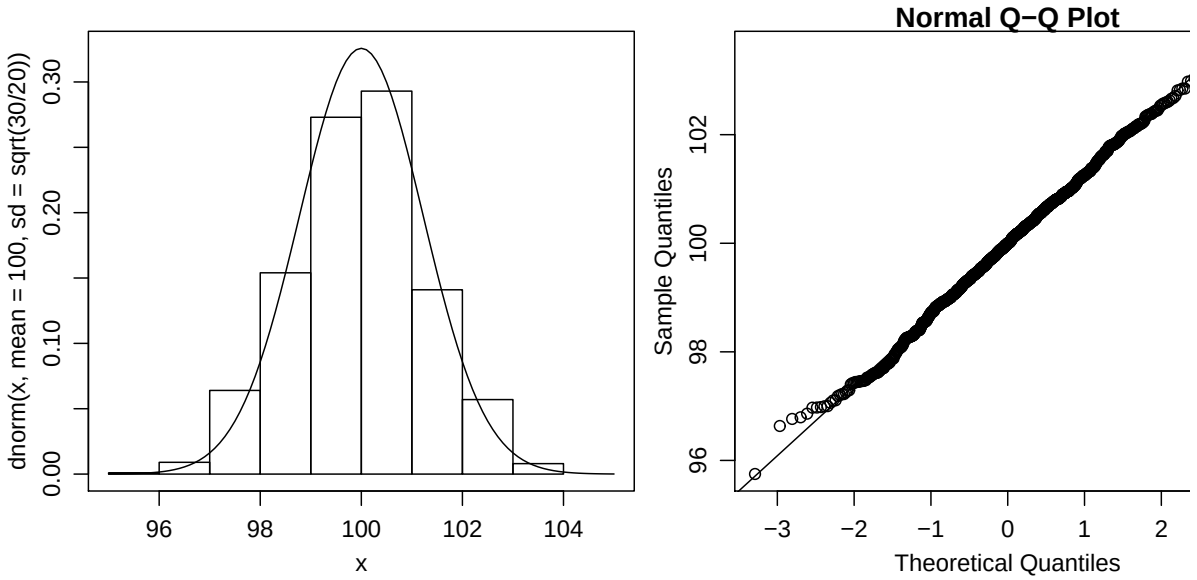


Figura 37: Histograma de uma amostra da distribuição amostral da média e a curva teórica da distribuição e o respectivo *qq-plot*.

15.3 Exercícios

1. Ilustrar usando simulação o resultado que afirma que para o estimador $S^2 = \sum \frac{(Y_i - \bar{Y})^2}{n-1}$ da variância de uma distribuição normal, a variável $V = (n-1)S^2/\sigma^2$ tem distribuição χ^2_{n-1}.
   DICA: Voce pode começar pensando nos passos necessários para ilustrar este resultado:
   - escolha os parâmetros de uma distribuição normal,
   - escolha o tamanho de amostra n e o número de simulações N,
   - gere N amostras de tamanho n,
   - para cada amostra calcule S^2 e $V = (n-1)S^2/\sigma^2$,
   - faça um histograma com os valores V e compare com a curva de uma distribuição χ^2_{n-1}.

2. No exercício anterior compare os valores teóricos $E[S^2] = \sigma^2$ e $Var[S^2] = \frac{2\sigma^2}{n-1}$ com os valores obtidos na simulação.

3. Considere uma distribuição normal de média $\mu = 0$ e variância unitária e amostras de tamanho $n = 20$ desta distribuição. Considere agora dois estimadores: $T_1 = \overline{(x)}$, a média da amostra e $T_2 = md(x)$, a mediana na amostra. Avalie e compare através de simulações a eficiência dos

dois estimadores. É possível identificar o mais eficiente? Qual a eficiência relativa? Repita o procedimento com diferentes tamanhos de amostra e verifique o efeito do tamanho da amostra na eficiência relativa.

4. Seja $Y_1, \ldots, Y_n$ a.a. de uma distribuição $\mathrm{N}(\mu, \sigma^2)$. Ilustrar o resultado que justifica o teste-t para média de uma amostra,
$$\frac{\bar{Y} - \mu}{S/\sqrt{n}} \sim t_{n-1}$$
onde S é o desvio padrão da amostra e n o tamanho da amostra.
DICA: começe verificando passo a passo, como no exercício anterior, o que é necessário para ilustrar este resultado.

5. Ilustrar o resultado que diz que o quociente de duas variáveis independentes com distribuição χ^2 tem distribuição F.

16 Intervalos de confiança – I

Nesta sessão vamos verificar como utilizar o R para obter intervalos de confiança para parâmetros de distribuições de probabilidade.

Para fins didáticos mostrando os recursos do R vamos mostrar três possíveis soluções:

1. fazendo as contas passo a passo, utilizando o R como uma calculadora
2. escrevendo uma função
3. usando uma função já existente no R

16.1 Média de uma distribuição normal com variância desconhecida

Considere o seguinte problema:

Exemplo

O tempo de reação de um novo medicamento pode ser considerado como tendo distribuição Normal e deseja-se fazer inferência sobre a média que é desconhecida obtendo um intervalo de confiança. Vinte pacientes foram sorteados e tiveram seu tempo de reação anotado. Os dados foram os seguintes (em minutos):

2.9	3.4	3.5	4.1	4.6	4.7	4.5	3.8	5.3	4.9
4.8	5.7	5.8	5.0	3.4	5.9	6.3	4.6	5.5	6.2

Entramos com os dados com o comando

```
> tempo <- c(2.9, 3.4, 3.5, 4.1, 4.6, 4.7, 4.5, 3.8, 5.3, 4.9, 4.8,
+     5.7, 5.8, 5, 3.4, 5.9, 6.3, 4.6, 5.5, 6.2)
```

Sabemos que o intervalo de confiança para média de uma distribuição normal com variância desconhecida, para uma amostra de tamanho n é dado por:

$$\left(\bar{x} - t_t\sqrt{\frac{S^2}{n}} \quad , \quad \bar{x} + t_t\sqrt{\frac{S^2}{n}}\right)$$

onde t_t é o quantil de ordem $1 - \alpha/2$ da distribuição t de Student, com $n - 1$ graus de liberdade.

Vamos agora obter a resposta das três formas diferentes mencionadas acima.

16.1.1 Fazendo as contas passo a passo

Nos comandos a seguir calculamos o tamanho da amostra, a média e a variância amostral.

```
> n <- length(tempo)
> n
[1] 20
> t.m <- mean(tempo)
> t.m
[1] 4.745
> t.v <- var(tempo)
> t.v
```

```
[1] 0.992079
```

A seguir montamos o intervalo utilizando os quantis da distribuição t, para obter um IC a 95% de confiança.

```
> t.ic <- t.m + qt(c(0.025, 0.975), df = n - 1) * sqrt(t.v/length(tempo))
> t.ic
[1] 4.278843 5.211157
```

16.1.2 Escrevendo uma função

Podemos generalizar a solução acima agrupando os comandos em uma função. Nos comandos primeiro definimos a função e a seguir utilizamos a função criada definindo intervalos a 95% e 99%.

```
> ic.m <- function(x, conf = 0.95) {
+     n <- length(x)
+     media <- mean(x)
+     variancia <- var(x)
+     quantis <- qt(c((1 - conf)/2, 1 - (1 - conf)/2), df = n - 1)
+     ic <- media + quantis * sqrt(variancia/n)
+     return(ic)
+ }
> ic.m(tempo)
[1] 4.278843 5.211157
> ic.m(tempo, conf = 0.99)
[1] 4.107814 5.382186
```

Escrever uma função é particularmente útil quando um procedimento vai ser utilizados várias vezes.

16.1.3 Usando a função `t.test`

Mostramos as soluções acima para ilustrar a flexibilidade e o uso do programa. Entretanto não precisamos fazer isto na maioria das vezes porque o R já vem com várias funções para procedimentos estatísticos já escritas. Neste caso a função `t.test` pode ser utilizada como vemos no resultado do comando a sequir que coincide com os obtidos anteriormente.

```
> t.test(tempo)

        One Sample t-test

data:  tempo
t = 21.3048, df = 19, p-value = 1.006e-14
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 4.278843 5.211157
sample estimates:
mean of x
    4.745
```

16.2 Exercícios

Em cada um dos exercícios abaixo tente obter os intervalos das três formas mostradas acima.

1. Pretende-se estimar a proporção p de cura, através de uso de um certo medicamento em doentes contaminados com cercária, que é uma das formas do verme da esquitosomose. Um experimento consistiu em aplicar o medicamento em 200 pacientes, escolhidos ao acaso, e observar que 160 deles foram curados. Montar o intervalo de confiança para a proporção de curados.
   Note que há duas expressões possíveis para este IC: o "otimista"e o "conservativo". Encontre ambos intervalos.

2. Os dados abaixo são uma amostra aleatória da distribuição $Bernoulli(p)$. Obter IC's a 90% e 99%.

   ```
   0 0 0 1 1 0 1 1 1 1 0 1 1 0 1 1 1 1 0 1 1 1 1 1 1
   ```

3. Encontre intervalos de confiança de 95% para a média de uma distribuição Normal com variância 1 dada a amostra abaixo

   ```
   9.5 10.8 9.3 10.7 10.9 10.5 10.7 9.0 11.0 8.4
   10.9 9.8 11.4 10.6 9.2 9.7 8.3 10.8 9.8 9.0
   ```

4. Queremos verificar se duas máquinas produzem peças com a mesma homogeneidade quanto a resistência à tensão. Para isso, sorteamos dias amostras de 6 peças de cada máquina, e obtivemos as seguintes resistências:

Máquina A	145	127	136	142	141	137
Máquina B	143	128	132	138	142	132

   Obtenha intervalos de confiança para a razão das variâncias e para a diferença das médias dos dois grupos.

17 Testes de hipótese

Os exercícios abaixo são referentes ao conteúdo de *Testes de Hipóteses* conforme visto na disciplina de Estatística Geral II.

Eles devem ser resolvidos usando como referência qualquer texto de Estatística Básica.
Procure resolver primeiramente sem o uso de programa estatístico.

A idéia é relembrar como são feitos alguns testes de hipótese básicos e corriqueiros em estatística.

Nesta sessão vamos verificar como utilizar o R para fazer teste de hipóteses sobre parâmetros de distribuições para as quais os resultados são bem conhecidos.

Os comandos e cálculos são bastante parecidos com os vistos em intervalos de confiança e isto nem poderia ser diferente visto que intervalos de confiança e testes de hipótese são relacionados.

Assim como fizemos com intervalos de confiança, aqui sempre que possível e para fins didáticos mostrando os recursos do R vamos mostrar três possíveis soluções:

1. fazendo as contas passo a passo, utilizando o R como uma calculadora
2. escrevendo uma função
3. usando uma função já existente no R

17.1 Comparação de variâncias de uma distribuição normal

Queremos verificar se duas máquinas produzem peças com a mesma homogeneidade quanto a resistência à tensão. Para isso, sorteamos dias amostras de 6 peças de cada máquina, e obtivemos as seguintes resistências:

Máquina A	145	127	136	142	141	137
Máquina B	143	128	132	138	142	132

O que se pode concluir fazendo um teste de hipótese adequado?

Solução:

Da teoria de testes de hipótese sabemos que, assumindo a distribuição normal, o teste para a hipótese:

$$H_0 : \sigma_A^2 = \sigma_B^2 \quad versus \quad H_a : \sigma_A^2 \neq \sigma_B^2$$

que é equivalente à

$$H_0 : \frac{\sigma_A^2}{\sigma_B^2} = 1 \quad versus \quad H_a : \frac{\sigma_A^2}{\sigma_B^2} \neq 1$$

é feito calculando-se a estatística de teste:

$$F_{calc} = \frac{S_A^2}{S_B^2}$$

e em seguida comparando-se este valor com um valor da tabela de F e/ou calculando-se o p-valor associado com $n_A - 1$ e $n_B - 1$ graus de liberdade. Devemos também fixar o nível de significância do teste, que neste caso vamos definir como sendo 5%.

Para efetuar as análises no R vamos primeiro entrar com os dados nos objetos que vamos chamar de `ma` e `mb` e calcular os tamanhos das amostras que vão ser armazenados nos objetos `na` e `nb`.

```
> ma <- c(145, 127, 136, 142, 141, 137)
> na <- length(ma)
> na
```

```
[1] 6
> mb <- c(143, 128, 132, 138, 142, 132)
> nb <- length(mb)
> nb
[1] 6
```

17.1.1 Fazendo as contas passo a passo

Vamos calcular a estatística de teste. Como temos o computador a disposição não precisamos de da tabela da distribuição F e podemos calcular o p-valor diretamente.

```
> ma.v <- var(ma)
> ma.v
[1] 40
> mb.v <- var(mb)
> mb.v
[1] 36.96667
> fcalc <- ma.v/mb.v
> fcalc
[1] 1.082056
> pval <- 2 * pf(fcalc, na - 1, nb - 1, lower = F)
> pval
[1] 0.9331458
```

No cálculo do P-valor acima multiplicamos o valor encontrado por 2 porque estamos realizando um teste bilateral.

17.1.2 Escrevendo uma função

Esta fica por sua conta!
Escreva a sua própria função para testar hipóteses sobre variâncias de duas distribuições normais.

17.1.3 Usando uma função do R

O R já tem implementadas funções para a maioria dos procedimentos estatísticos "usuais". Por exemplo, para testar variâncias neste exemplo utilizamos `var.test()`. Vamos verificar os argumentos da função.

```
> args(var.test)
function (x, ...)
NULL
```

Note que esta saída não é muito informativa. Este tipo de resultado indica que `var.test()` é um método com mais de uma função associada. Portanto devemos pedir os argumentos da função "default".

```
> args(getS3method("var.test", "default"))
function (x, y, ratio = 1, alternative = c("two.sided", "less",
    "greater"), conf.level = 0.95, ...)
NULL
```

Neste argumentos vemos que a função recebe dois vetores de de dados (x e y), que por "default" a hipótese nula é que o quociente das variâncias é 1 e que a alternativa pode ser bilateral ou unilateral. Como `"two.sided"` é a primeira opção o "default" é o teste bilateral. Finalmente o nível de confiança é 95% ao menos que o último argumento seja modificado pelo usuário. Para aplicar esta função nos nossos dados basta digitar:

```
> var.test(ma, mb)
        F test to compare two variances

data:  ma and mb
F = 1.0821, num df = 5, denom df = 5, p-value = 0.9331
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.1514131 7.7327847
sample estimates:
ratio of variances
          1.082056
```

e note que a saída inclui os resultados do teste de hipótese bem como o intervalo de confiança. A decisão baseia-se em verificar se o P-valor é menor que o definido inicialmente.

17.2 Exercícios

Os exercícios a seguir foram retirados do livro de Bussab & Morettin (2003).

Note que nos exercícios abaixo nem sempre voce poderá usar funções de teste do R porque em alguns casos os dados brutos não estão disponíveis. Nestes casos voce deverá fazer os cálculos usando o R como calculadora.

1. Uma máquina automática de encher pacotes de café enche-os segundo uma distribuição normal, com média μ e variância $400g^2$. O valor de μ pode ser fixado num mostrador situado numa posição um pouco inacessível dessa máquina. A máquina foi regulada para $\mu = 500g$. Desejamos, de meia em meia hora, colher uma amostra de 16 pacotes e verificar se a produção está sob controle, isto é, se $\mu = 500g$ ou não. Se uma dessas amostras apresentasse uma média $\bar{x} = 492g$, voce pararia ou não a produção para verificar se o mostrador está na posição correta?

2. Uma companhia de cigarros anuncia que o índice médio de nicotina dos cigarros que fabrica apresenta-se abaixo de $23mg$ por cigarro. Um laboratório realiza 6 análises desse índice, obtendo: 27, 24, 21, 25, 26, 22. Sabe-se que o índice de nicotina se distribui normalmente, com variância igual a $4,86mg^2$. Pode-se aceitar, ao nível de 10%, a afirmação do fabricante.

3. Uma estação de televisão afirma que 60% dos televisores estavam ligados no seu programa especial de última segunda feira. Uma rede competidora deseja contestar essa afirmação, e decide, para isso, usar uma amostra de 200 famílias obtendo 104 respostas afirmativas. Qual a conclusão ao nível de 5% de significância?

4. O tempo médio, por operário, para executar uma tarefa, tem sido 100 minutos, com um desvio padrão de 15 minutos. Introduziu-se uma modificação para diminuir esse tempo, e, após certo período, sorteou-se uma amostra de 16 operários, medindo-se o tempo de execução de cada um. O tempo médio da amostra foi de 85 minutos, o o desvio padrão foi 12 minutos. Estes resultados trazem evidências estatísticas da melhora desejada?

5. Num estudo comparativo do tempo médio de adaptação, uma amostra aleatória, de 50 homens e 50 mulheres de um grande complexo industrial, produziu os seguintes resultados:

Estatísticas	Homens	Mulheres
Médias	3,2 anos	3,7 anos
Desvios Padrões	0,8 anos	0,9 anos

Pode-se dizer que existe diferença significativa entre o tempo de adaptação de homens e mulheres?

A sua conclusão seria diferente se as amostras tivessem sido de 5 homens e 5 mulheres?

18 Intervalos de confiança e testes de hipótese

Nesta sessão vamos ver mais alguns exemplos sobre como utilizar o R para obter intervalos de confiança e testar hipóteses sobre parâmetros de interesse na população, a partir de dados obtidos em amostras. Para isto vamos ver alguns problemas típicos de cursos de estatística básica.

18.1 Teste χ^2 de independência

Quando estudamos a relação entre duas variáveis qualitativas fazemos uma tabela com o resultado do cruzamento desta variáveis. Em geral existe interesse em verificar se as variáveis estão associadas e para isto calcula-se uma medida de associação tal como o χ^2, coeficiente de contingência C, ou similar. O passo seguinte é verificar se existe evidência suficiente nos dados para declarar que a variáveis estão associadas. Uma possível forma de testar tal hipótese é utilizando o teste χ^2.

Para ilustrar o teste vamos utilizar o conjunto de dados `HairEyeColor` que já vem disponível com o R. Para carregar e visualizar os dados use os comando abaixo.

```
> data(HairEyeColor)
> HairEyeColor
> as.data.frame(HairEyeColor)
```

Para saber mais sobre estes dados veja `help(HairEyeColor)` Note que estes dados já vem "resumidos" na forma de uma tabela de frequências tri-dimensional, com cada uma das dimensões correspondendo a um dos atributos - cor dos cabelos, olhos e sexo.

Para ilustrar aqui o teste χ^2 vamos verificar se existe associação entre 2 atributos: cor dos olhos e cabelos entre os indivíduos do sexo feminino. Portanto as hipóteses são:

$$\begin{aligned} H_0: &\quad \text{não existe associação} \\ H_a: &\quad \text{existe associação} \end{aligned}$$

Vamos adotar $\alpha = 5\%$ como nível de significância. Nos comandos abaixo primeiro isolamos apenas a tabela com os indivíduos do sexo masculino e depois aplicamos o teste sobre esta tabela.

```
> HairEyeColor[,,1]
       Eye
Hair    Brown Blue Hazel Green
  Black    32   11    10     3
  Brown    38   50    25    15
  Red      10   10     7     7
  Blond     3   30     5     8
> chisq.test(HairEyeColor[,,1])

        Pearson's Chi-squared test

data:  HairEyeColor[, , 1]
X-squared = 42.1633, df = 9, p-value = 3.068e-06

Warning message:
Chi-squared approximation may be incorrect in: chisq.test(HairEyeColor[, , 1])
```

O $p - value$ sugere que a associação é significativa. Entretanto este resultado deve ser visto com cautela pois a mensagem de alerta (*Warning message*) emitida pelo programa chama atenção ao fato de que há várias caselas com baixa frequência na tabela e portanto as condições para a validade do teste não são perfeitamente satisfeitas.

Uma possibilidade neste caso é então usar o $p - value$ calculado por simulação, ao invés do resultado assintótico usado no teste tradicional.

```
> chisq.test(HairEyeColor[,,1], sim=T)

        Pearson's Chi-squared test with simulated p-value (based on 2000
        replicates)

data:  HairEyeColor[, , 1]
X-squared = 42.1633, df = NA, p-value = 0.0004998
```

Note que agora a mensagem de alerta não é mais emitida e que a significância foi confirmada (P-valor < 0.05). Note que se voce rodar este exemplo poderá obter um $p - value$ um pouco diferente porque as simulações não necessariamente serão as mesmas.

Lembre-se de inspecionar `help(chisq.test)` para mais detalhes sobre a implementação deste teste no R.

18.2 Teste para o coeficiente de correlação linear de Pearson

Quando temos duas variáveis quantitativas podemos utilizar o coeficiente de correlação linear para medir a associação entre as variáveis, se a relação entre elas for linear. Para ilustrar o teste para o coeficiente linear de Pearson vamos estudar a relação entre o peso e rendimento de carros. Para isto vamos usar as variáveis `wt` (peso) e `mpg` (milhas por galão) do conjunto de dados `mtcars`.

```
> data(mtcars)
> attach(mtcars)
> cor(wt, mpg)
[1] -0.8676594
> cor.test(wt, mpg)

        Pearson's product-moment correlation

data:  wt and mpg
t = -9.559, df = 30, p-value = 1.294e-10
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.9338264 -0.7440872
sample estimates:
       cor
-0.8676594

> detach(mtcars)
```

Portanto o p-valor acima mmostra que a correlação encontrada de -0.87 difere significativamente de zero. Note que uma análise mais cuidadosa deveria incluir o exame do gráfico entre estas duas variáveis para ver se o coeficiente de correlação linear é adequado para medir a associação.

18.3 Comparação de duas médias

Quando temos uma variável qualitativa com dois níveis e outra quantitativa a análise em geral recai em comparar as médias da quantitativa para cada grupo da qualitativa. Para isto podemos utilizar o *testeT*. Há diferentes tipos de teste T: para amostras independentes ou pareadas, variâncias iguais ou desiguais. Além disto podemos fazer testes uni ou bilaterais. Todos estes podem ser efetuados com a função `t.test`. Usando argumentos desta função definimos o tipo de teste desejado. No exemplo abaixo veremos um teste unilateral, para dois grupos com variâncias consideradas iguais.

Considere o seguinte exemplo:

Os dados a seguir correpondem a teores de um elemento indicador da qualidade de um certo produto vegetal. Foram coletadas 2 amostras referentes a 2 métodos de produção e deseja-se comparar as médias dos métodos fazendo-se um teste t bilateral, ao nível de 5% de significância e considerando-se as variâncias iguais.

Método 1	0.9	2.5	9.2	3.2	3.7	1.3	1.2	2.4	3.6	8.3
Método 2	5.3	6.3	5.5	3.6	4.1	2.7	2.0	1.5	5.1	3.5

```
> m1 <- c(0.9, 2.5, 9.2, 3.2, 3.7, 1.3, 1.2, 2.4, 3.6, 8.3)
> m2 <- c(5.3, 6.3, 5.5, 3.6, 4.1, 2.7, 2.0, 1.5, 5.1, 3.5)

 t.test(m1,m2, var.eq=T)

        Two Sample t-test

data:  m1 and m2
t = -0.3172, df = 18, p-value = 0.7547
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -2.515419  1.855419
sample estimates:
mean of x mean of y
     3.63      3.96
```

Os resultados mostram que não á evidências para rejeitar a hipótese de igualdade entre as médias.

18.4 Exercícios

1. Revisite os dados `milsa` visto na aula de estatística descritiva e selecione pares de variáveis adequadas para efetuar:
   (a) um teste χ^2
   (b) um teste para o coeficiente de correlação
   (c) um teste t
2. Inspecione o conjunto de dados humanos.txt, selecione variáveis a aplique os testes vistos nesta Seção.
3. Queremos verificar se machos e fêmeas de uma mesma espécie possuem o mesmo comprimento (em *mm*) Para isso, foram medidos 6 exemplares de cada sexo e obtivemos os seguintes comprimentos:

Machos	145	127	136	142	141	137
Fêmeas	143	128	132	138	142	132

Obtenha intervalos de confiança para a razão das variâncias e para a diferença das médias dos dois grupos.

Dica: Use as funções `var.test` e `t.test`

4. Carregue o conjunto de dados *iris* usando o comando `data(iris)`.
Veja a descrição dos dados em `help(iris)`.
Use a função `cor.test` para testar a correlação entre o comprimento de sépalas e pétalas.

19 Funções de verossimilhança

A função de verossimilhança é central na inferência estatística. Nesta sessão vamos ver como traçar gráficos de funções de verossimilhança de um parâmetro utilizando o programa R. Também veremos como traçar a função *deviance*, obtida a partir da função de verossimilhança e conveniente em certos casos para representações gráficas, cálculos e inferências.

19.1 Definições e notações

Seja $L(\theta; y)$ a função de verossimilhança. A notação indica que o argumento da função é θ que pode ser um escalar ou um vetor de parâmetros. Nesta sessão consideraremos que é um escalar. O termo y denota valores realizados de uma variável aleatória Y, isto é os valores obtidos em uma amostra.

O valor que maximiza $L(\theta; y)$ é chamado do estimador de máxima verossimilhança e denotado por $\hat{\theta}$. A função de verossimilhança *relativa* ou *normatizada* $R(\theta; y)$ é dada pela razão entre a função de verossimilhança e o valor maximizado desta função, portanto $R(\theta; y) = L(\theta; y)/L(\hat{\theta}; y)$, assumindo valores no intervalo $[0, 1]$. Esta função é útil para comparar todos dos modelos dados pelos diferentes valores de θ com o modelo mais plausível (verossível) para a amostra obtida.

O valor que maximiza a função de verossimilhança é também o que maximiza a a função obtida pelo logarítimo da função de verossimilhança, chamada função de log-verossimilhança, uma vez que a função logarítimo é uma função monotônica. Denotamos a função de log-verossimilhança por $l(\theta; y)$ sendo $l(\theta; y) = \log(L(\theta; y))$. A função de log-verossimilhança é mais adequada para cálculos computacionais e permite que modelos possam ser comparados aditivamente, ao invés de multiplicativamente.

Aplicando-se o logarítimo à função padronizada obtemos $\log\{R(\theta; y)\} = l(\theta; y) - l(\hat{\theta}; y)$, que tem portanto um valor sempre não-positivo. Desta forma esta função pode ser multiplicada por um número negativo arbitrário, e sendo este número -2 obtemos a chamada *função deviance*, $D(\theta; y) = -2\left[l(\theta; y) - l(\hat{\theta}; y)\right]$, onde lembramos que $\hat{\theta}$ é o estimador de máxima verossimilhança de θ. Esta função tem portanto o seu mínimo em zero e quanto maior o seu valor, maior a diferença de plausibilidade entre o modelo considerado e o modelo mais plausível para os dados obtidos na amostra. Esta função combina as vantagens da verossimilhança relativa e da log-verossimilhança sendo portanto conveniente para cálculos computacionais e inferência.

19.2 Exemplo 1: Distribuição normal com variância conhecida

Seja o vetor $(12, 15, 9, 10, 17, 12, 11, 18, 15, 13)$ uma amostra aleatória de uma distribuição normal de média μ e variância conhecida e igual a 4. O objetivo é fazer um gráfico da função de log-verossimilhança.

Solução:

Vejamos primeiro os passos da solução analítica:

1. Temos que $X_1, \ldots, X_n$ onde, neste exemplo $n = 10$, é uma a.a. de $X \sim N(\mu, 4)$,
2. a densidade para cada observação é dada por $f(x_i) = \frac{1}{2\sqrt{2\pi}} \exp\{-\frac{1}{8}(x_i - \mu)^2\}$,
3. a verossimilhança é dada por $L(\mu) = \prod_1^{10} f(\mu; x_i)$,

4. e a log-verossimilhança é dada por

$$\begin{aligned} l(\mu) &= \sum_{1}^{10} \log(f(x_i)) \\ &= -5\log(8\pi) - \frac{1}{8}(\sum_{1}^{10} x_i^2 - 2\mu \sum_{1}^{10} x_i + 10\mu^2), \end{aligned} \tag{4}$$

5. que é uma função de μ e portanto devemos fazer um gráfico de $l(\mu)$ *versus* μ tomando vários valores de μ e calculando os valores de $l(\mu)$.

Vamos ver agora uma primeira possível forma de fazer a função de verossimilhança no R.

1. Primeiro entramos com os dados que armazenamos no vetor `x`

```
> x <- c(12, 15, 9, 10, 17, 12, 11, 18, 15, 13)
```

2. e calculamos as quantidades $\sum_1^{10} x_i^2$ e $\sum_1^{10} x_i$

```
> sx2 <- sum(x^2)
> sx <- sum(x)
```

3. agora tomamos uma sequência de valores para μ. Sabemos que o estimador de máxima verossimilhança neste caso é $\hat{\mu} = 13.2$ (este valor pode ser obtido com o comando `mean(x)`) e portanto vamos definir tomar valores ao redor deste ponto.

```
> mu.vals <- seq(11, 15, l = 100)
```

4. e a seguir calculamos os valores de $l(\mu)$ de acordo com a equação acima

```
> lmu <- -5 * log(8 * pi) - (sx2 - 2 * mu.vals * sx + 10 * (mu.vals^2))/8
```

5. e finalmente fazemos o gráfico visto na Figura 38

```
> plot(mu.vals, lmu, type = "l", xlab = expression(mu), ylab = expression(l(mu)))
```

Entretanto podemos obter a função de verossimilhança no R de outras forma mais geral e menos trabalhosa. Sabemos que a função `dnorm()` calcula a densidade $f(x)$ da distribuição normal e podemos usar este fato para evitar a digitação da expressão acima.

- Primeiro vamos criar uma função que calcula o valor da log-verossimilhança para um certo valor do parâmetro e para um certo conjunto de dados,

```
> logvero <- function(mu, dados) {
+     sum(dnorm(dados, mean = mu, sd = 2, log = TRUE))
+ }
```

- a seguir criamos uma sequência adequada de valores de μ e calculamos $l(\mu)$ para cada um dos valores

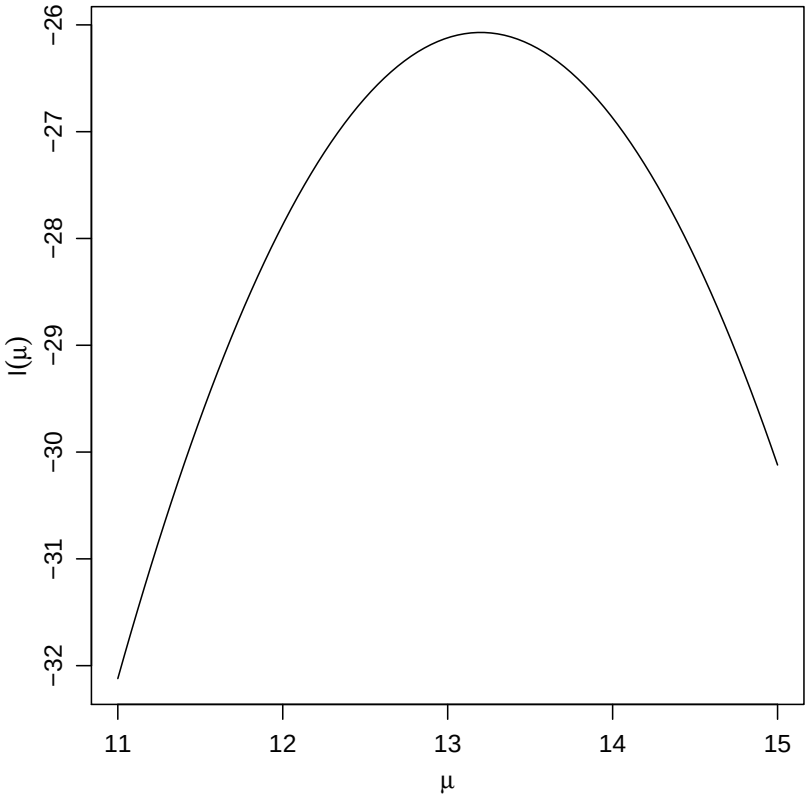


Figura 38: Função de verossimilhança para o parâmetro μ da distribuição normal com variância $\sigma^2 = 4$ com os dados do Exemplo 1.

```
> mu.vals <- seq(11, 15, l = 100)
> mu.vals

  [1] 11.00000 11.04040 11.08081 11.12121 11.16162 11.20202 11.24242 11.28283
  [9] 11.32323 11.36364 11.40404 11.44444 11.48485 11.52525 11.56566 11.60606
 [17] 11.64646 11.68687 11.72727 11.76768 11.80808 11.84848 11.88889 11.92929
 [25] 11.96970 12.01010 12.05051 12.09091 12.13131 12.17172 12.21212 12.25253
 [33] 12.29293 12.33333 12.37374 12.41414 12.45455 12.49495 12.53535 12.57576
 [41] 12.61616 12.65657 12.69697 12.73737 12.77778 12.81818 12.85859 12.89899
 [49] 12.93939 12.97980 13.02020 13.06061 13.10101 13.14141 13.18182 13.22222
 [57] 13.26263 13.30303 13.34343 13.38384 13.42424 13.46465 13.50505 13.54545
 [65] 13.58586 13.62626 13.66667 13.70707 13.74747 13.78788 13.82828 13.86869
 [73] 13.90909 13.94949 13.98990 14.03030 14.07071 14.11111 14.15152 14.19192
 [81] 14.23232 14.27273 14.31313 14.35354 14.39394 14.43434 14.47475 14.51515
 [89] 14.55556 14.59596 14.63636 14.67677 14.71717 14.75758 14.79798 14.83838
 [97] 14.87879 14.91919 14.95960 15.00000

> lmu <- sapply(mu.vals, logvero, dados = x)
> lmu

  [1] -32.12086 -31.90068 -31.68458 -31.47256 -31.26462 -31.06076 -30.86099 -30.66529
  [9] -30.47368 -30.28615 -30.10270 -29.92333 -29.74804 -29.57683 -29.40971 -29.24666
 [17] -29.08770 -28.93282 -28.78201 -28.63529 -28.49266 -28.35410 -28.21962 -28.08923
 [25] -27.96291 -27.84068 -27.72253 -27.60846 -27.49847 -27.39256 -27.29074 -27.19299
 [33] -27.09933 -27.00975 -26.92424 -26.84282 -26.76549 -26.69223 -26.62305 -26.55796
 [41] -26.49694 -26.44001 -26.38716 -26.33839 -26.29370 -26.25309 -26.21656 -26.18412
```

```
 [49] -26.15575 -26.13147 -26.11127 -26.09515 -26.08311 -26.07515 -26.07127 -26.07147
 [57] -26.07576 -26.08413 -26.09657 -26.11310 -26.13371 -26.15840 -26.18718 -26.22003
 [65] -26.25697 -26.29798 -26.34308 -26.39226 -26.44552 -26.50286 -26.56428 -26.62978
 [73] -26.69937 -26.77304 -26.85078 -26.93261 -27.01852 -27.10851 -27.20258 -27.30074
 [81] -27.40297 -27.50929 -27.61968 -27.73416 -27.85272 -27.97536 -28.10208 -28.23289
 [89] -28.36777 -28.50674 -28.64978 -28.79691 -28.94812 -29.10341 -29.26278 -29.42623
 [97] -29.59377 -29.76538 -29.94108 -30.12086
```

Note na sintaxe acima que a função `sapply` aplica a função `logvero` anteriormente definida em cada elemento do vetor `mu.vals`.

- Finalmente fazemos o gráfico.

```
> plot(mu.vals, lmu, type = "l", xlab = expression(mu), ylab = expression(l(mu)))
```

Para encerrar este exemplo vamos apresentar uma solução ainda mais genérica que consiste em criar uma função que vamos chamar de `vero.norm.v4` para cálculo da verossimilhança de distribuições normais com σ^2=4. Esta função engloba os comandos acima e pode ser utilizada para obter o gráfico da log-verossimilhança para o parâmetro μ para qualquer amostra obtida desta distribuição.

```
> vero.normal.v4 <- function(mu, dados) {
+     logvero <- function(mu, dados) sum(dnorm(dados, mean = mu, sd = 2,
+         log = TRUE))
+     sapply(mu, logvero, dados = dados)
+ }
> curve(vero.normal.v4(x, dados = x), 11, 15, xlab = expression(mu),
+     ylab = expression(l(mu)))
```

19.3 Exemplo 2: Distribuição Poisson

Considere agora a amostra armazenada no vetor `y`:

```
> y <- c(5, 0, 3, 2, 1, 2, 1, 1, 2, 1)
```

de uma distribuição de Poisson de parâmetro λ. A função de verossimilhança pode ser definida por:

```
> lik.pois <- function(lambda, dados) {
+     loglik <- function(l, dados) {
+         sum(dpois(dados, lambda = l, log = TRUE))
+     }
+     sapply(lambda, loglik, dados = dados)
+ }
```

E podemos usar esta função para fazer o gráfico da função de verossimilhança como visto à esquerda da Figura 39

```
> lambda.vals <- seq(0, 10, l = 101)
> loglik <- sapply(lambda.vals, lik.pois, dados = y)
> plot(lambda.vals, loglik, ty = "l")
```

E o comando para gerar o gráfico poderia incluir o texto do eixos:

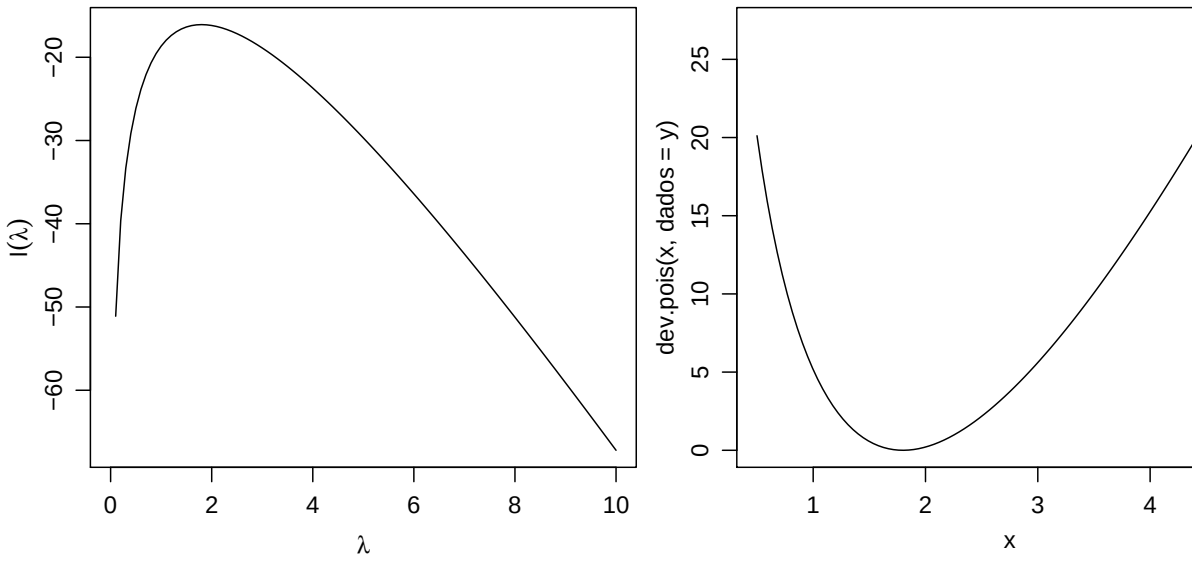


Figura 39: Função de verossimilhança (esquerda) e deviance (direita) para o parâmetro λ da distribuição Poisson.

```
> plot(lambda.vals, loglik, type = "l", xlab = expression(lambda),
+     ylab = expression(l(lambda)))
```

ou simplesmente usar:

```
> curve(lik.pois(x, dados = y), 0, 10)
```

Alternativamente pode-se fazer um gráfico da função deviance, como nos comandos abaixo.

```
> dev.pois <- function(lambda, dados) {
+     lambda.est <- mean(dados)
+     lik.lambda.est <- lik.pois(lambda.est, dados = dados)
+     lik.lambda <- lik.pois(lambda, dados = dados)
+     return(-2 * (lik.lambda - lik.lambda.est))
+ }
> curve(dev.pois(x, dados = y), 0, 10)
```

Ou fazendo novamente em um intervalo menor

```
> curve(dev.pois(x, dados = y), 0.5, 5)
```

O estimador de máxima verossimilhança é o valor que maximiza a função de verossimilhança que é o mesmo que minimiza a função deviance. Neste caso sabemos que o estimador tem expressão analítica fechada $\lambda = \bar{x}$ e portanto calculado com o comando.

```
> lambda.est <- mean(y)
> lambda.est
[1] 1.8
```

Caso o estimador não tenha expressão fechada pode-se usar maximização (ou minimização) numérica. Para ilustrar isto vamos encontrar a estimativa do parâmetro da Poisson e verificar que o valor obtido coincide com o valor dado pela expressão fechada do estimador. Usamos o função `optimise()` para encontrar o ponto de mínimo da função deviance.

```
> optimise(dev.pois, int = c(0, 10), dados = y)
$minimum
[1] 1.800004

$objective
[1] 1.075264e-10
```

A função `optimise()` é adequada para minimizações envolvendo um único parâmetro. Para dois ou mais parâmetros deve-se usar a função `optim()` ou `nlminb()`.

Finalmente os comandos abaixo são usados para obter graficamente o intervalo de confiança (a 95%) baseado na deviance.

```
> curve(dev.pois(x, dados = y), 1, 3.5, xlab = expression(lambda),
+     ylab = expression(l(lambda)))
> corte <- qchisq(0.95, df = 1)
> abline(h = corte)
```

Os limites (aproximados) do IC podem ser obtidos da forma:

```
> l.vals <- seq(0.5, 5, l = 1001)
> dev.l <- dev.pois(l.vals, dados = y)
> dif <- abs(dev.l - corte)
> ind <- l.vals < lambda.est
> ic2.lambda <- c(l.vals[ind][which.min(dif[ind])], l.vals[!ind][which.min(dif[!ind])])
> ic2.lambda
[1] 1.0895 2.7635
```

E adicionados ao gráfico com

```
> segments(ic2.lambda, 0, ic2.lambda, corte)
```

19.4 Exemplo 3: Distribuição normal com variância desconhecida

Vamos agora revisitar o Exemplo 1 desta seção, usando os mesmos dados porém agora sem assumir que a variância é conhecida. Portanto temos agora dois parâmetros sobre os quais queremos fazer inferência: μ e σ . O objetivo é fazer um gráfico 3-D da função de log-verossimilhança de dois argumentos $l(\mu, \sigma)$.

Solução:

Vejamos primeiro os passos da solução analítica:

1. Temos que $X_1, \ldots, X_n$ onde, neste exemplo $n = 10$, é uma a.a. de $X \sim N(\mu, \sigma^2)$,
2. a densidade para cada observação é dada por $f(x_i) = \frac{1}{\sigma\sqrt{2\pi}} \exp\{-\frac{1}{2\sigma^2}(x_i - \mu)^2\}$,
3. a verossimilhança é dada por $L(\mu, \sigma) = \prod_1^{10} f(\mu, \sigma; x_i)$,
4. e a log-verossimilhança é dada por

$$\begin{aligned} l(\mu, \sigma) &= \sum_1^{10} \log(f(x_i)) \\ &= -5\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}(\sum_1^{10} x_i^2 - 2\mu \sum_1^{10} x_i + 10\mu^2), \end{aligned} \tag{5}$$

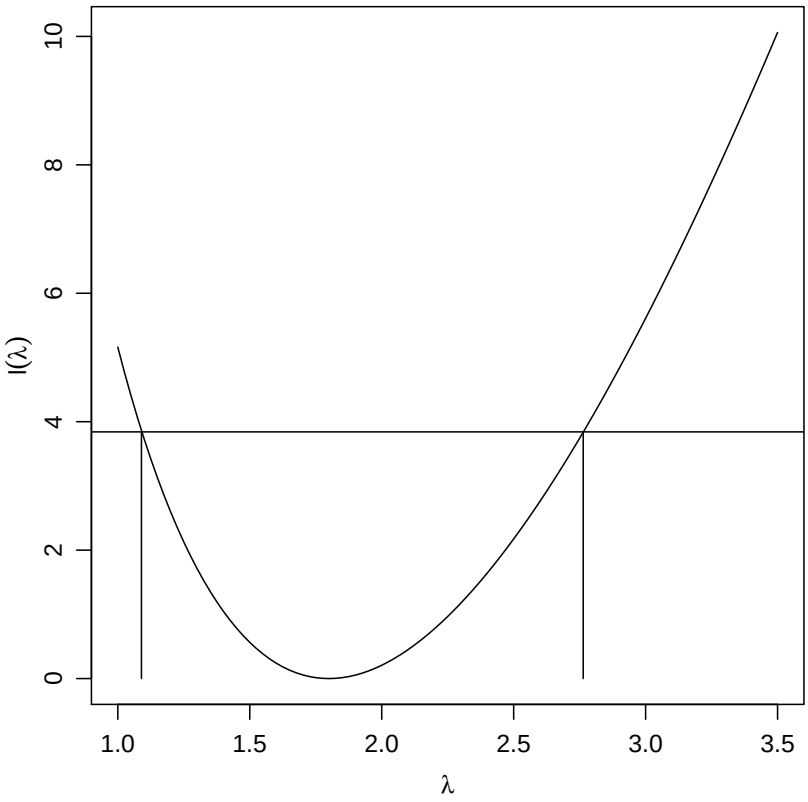


Figura 40: Intervalo de confiança baseado na deviance para o parâmetro λ da distribuição Poisson.

5. que é uma função de μ e σ e portanto devemos fazer um gráfico tridimensional de $l(\mu, \sigma)$ *versus* μ e σ tomando vários valores de pares (μ, σ) e calculando os valores correspondentes de $l(\mu, \sigma)$.

Assim como no Exemplo 1 poderíamos calcular a verossimilhança fazendo as contas "passo a passo"da função acima, ou então usando a função `dnorm()`. Neste exemplo vamos fazer apenas da segunda forma, ficando a primeira como exercício para o leitor.

1. Primeiro entramos com os dados que armazenamos no vetor `x`. Vamos também calcular as estimativas de máxima verossimilhança.

```
> x <- c(12, 15, 9, 10, 17, 12, 11, 18, 15, 13)
> pars.MV <- c(mu = mean(x), sd = sqrt(var(x) * (length(x) - 1)/length(x)))
> pars.MV

       mu        sd
13.200000  2.821347
```

2. a seguir vamos criar uma função que calcula o valor da log-verossimilhança para um certo par de valores dos parâmetros (média e desvio padrão, nesta ordem) e para um certo conjunto de dados,

```
> logveroN <- function(pars, dados) sum(dnorm(dados, mean = pars[1],
+     sd = pars[2], log = TRUE))
```

3. a seguir criamos uma sequência adequada de pares de valores de (μ, σ) e calculamos $l(\mu, \sigma)$ para cada um dos pares.

```
> par.vals <- expand.grid(mu = seq(11, 15, l = 100), sd = seq(1.5,
+     20, l = 100))
> dim(par.vals)
```

```
[1] 10000     2
```

```
> head(par.vals)
```

```
        mu  sd
1 11.00000 1.5
2 11.04040 1.5
3 11.08081 1.5
4 11.12121 1.5
5 11.16162 1.5
6 11.20202 1.5
```

```
> tail(par.vals)
```

```
            mu sd
9995  14.79798 20
9996  14.83838 20
9997  14.87879 20
9998  14.91919 20
9999  14.95960 20
10000 15.00000 20
```

```
> par.vals$logL <- apply(par.vals, 1, logveroN, dados = x)
> head(par.vals)
```

```
        mu  sd      logL
1 11.00000 1.5 -41.68848
2 11.04040 1.5 -41.29705
3 11.08081 1.5 -40.91287
4 11.12121 1.5 -40.53595
5 11.16162 1.5 -40.16628
6 11.20202 1.5 -39.80387
```

Note na sintaxe acima que a função `apply` aplica a função `logveroN` a cada par de valores em cada linha de `par.vals`. Ao final o objeto |par.vals| contém na terceira coluna os valores da log-verossimilhança correspondentes as valores dos parâmetros dados na primeira e segunda colunas.

4. O gráfico 3-D da função pode ser visualizado de três formas alternativas como mostrado na Figura 41: como uma superfície 3D gerada pela função `persp()`, como um mapa de curvas de isovalores obtido com `image()`, ou ainda como um mapa de cores correspondentes aos valores gerado por `image()`.

```
> with(par.vals, persp(unique(mu), unique(sd), matrix(logL, ncol = length(unique(sd))),
+     xlab = expression(mu), ylab = expression(sigma), zlab = expression(l(mu,
+         sigma)), theta = 30, phi = 30))
> with(par.vals, contour(unique(mu), unique(sd), matrix(logL, ncol = length(unique(sd))
+     xlab = expression(sigma), ylab = expression(mu), nlev = 20))
> points(pars.MV[1], pars.MV[2], pch = 4, cex = 1.5)
```

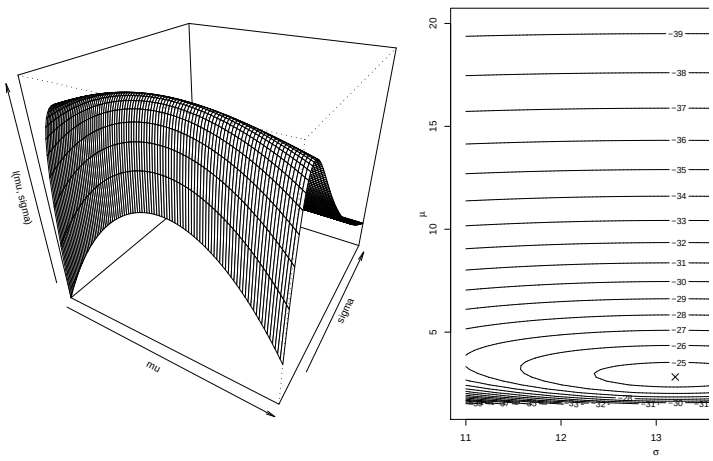


Figura 41: Função de verossimilhança para os parâmetros μ e σ da distribuição normal com os dados do Exemplo 1.

```
> with(par.vals, image(unique(mu), unique(sd), matrix(logL, ncol = length(unique(sd))),
+     xlab = expression(mu), ylab = expression(sigma), col = gray(seq(0,
+         1, length = 20))))
> points(pars.MV[1], pars.MV[2], pch = 4, cex = 1.5)
```

Notas:

- a obtenção da função foi necessário especificar faixas de valores para μ e σ. A definição desta faixa foi feita após várias tentativas pois depende do problema, em especial do número e variabilidade dos dados.
- as funções gráficas utilizadas requirem: dois vetores de tamanhos n_1 e n_2 com os valores dos argumentos da função e os valores da função em uma matrix de dimensão $n_1 \times n_2$. Por isto usamos `unique()` para extrair os valores dos argumentos, sem repeti-los e `matrix()` para os valores da função.
- na função `perp()` as argumentos `theta` e `phi` são utilizados para rotacionar o gráfico a fim de se obter uma melhor visualização.
- o valor das estimativas de máxima verossimilhança são indicados por `x` nos dois últimos gráficos. Neste caso eles foram encontrados facilmente como mostrado acima no objeto `pars.MV` pois podem ser obtidos analiticamente. De forma mais geral, a função `fitdistr()` do pacote **MASS** poide ser usada para encontrar estimativas de máxima verossimilhança.

```
> require(MASS)
> MV <- fitdistr(x, "normal")
> MV
```

```
      mean          sd
  13.2000000    2.8213472
 ( 0.8921883) ( 0.6308724)
```

19.5 Exercícios

1. Seja a amostra abaixo obtida de uma distribuição Poisson de parâmetro λ.
   5 4 6 2 2 4 5 3 3 0 1 7 6 5 3 6 5 3 7 2
   Obtenha o gráfico da função de log-verossimilhança.

2. Seja a amostra abaixo obtida de uma distribuição Binomial de parâmetro p e com $n = 10$.
   7 5 8 6 9 6 9 7 7 7 8 8 9 9 9
   Obtenha o gráfico da função de log-verossimilhança.

3. Seja a amostra abaixo obtida de uma distribuição χ^2 de parâmetro ν.
   8.9 10.1 12.1 6.4 12.4 16.9 10.5 9.9 10.8 11.4
   Obtenha o gráfico da função de log-verossimilhança.

20 Escrevendo textos com o LaTeX

O LaTeXé uma ferramenta para editoração que preduz textos com alta qualidade gráfica, em particular para documentos que incluem fórmulas e equações. Além disto a possibilidade de integração com o R, com o mecanismo do *Sweave* que veremos mais adiante, torna o uso deste editor ainda mais adequado a atrativo para estatísticos. Durante as aulas vamos explicar o funcionamento do LaTeXe da estrutura do documento, enquanto inspecionamos alguns arquivos com exemplos.

20.1 Documentos editados nas aulas

Durante as aulas estamos construindo documentos ilustrando o uso do LaTeX. Clique para copiar um arquivo básico com exemplos (exemplo.tex) e copie também o arquivo histograma.ps com a figura gerada no R e incluída neste documento.

20.2 Alguns links

Iniciantes e mesmo usuários acostumados com o LaTeXprecisam frequentemente recorrer à documentação para saber como obter o resultado esperado na editoração. Há vários textos disponíveis tanto no formato de livro quanto na WEB – vasculhe e escolha o(s) seu(s) preferido(s)!

Para começar aqui estão alguns links com documentação e introdução ao LaTeX:

- Um excelente site/wiki sobre o LaTeX em portuguêses
- Introdução ao LaTeX
- Curso de Introdução ao LaTeX
- Curso de Introdução ao LaTeX
- LaTeXpara iniciantes
- outro texto de Introdução ao LaTeX
- Uma breve introdução ao LaTeX
- Um Tutorial
- Dicas do Ricardo para uso do LaTeX

20.3 Uso e Interfaces

Para editar documentos em LaTeXvoce precisa ter instalado:

- LINUX: o programa tetex que usualmente é instalado automaticamente junto com o LINUX, caso contrário instale os pacotes tetex.
- WINDOWS: o programa em MiKTeX é livremente disponível para download.

Um documento LaTeXé um arquivo texto e portanto pode ser editado em qualquer editor. Depois de escrito pode ser compilado na "linha de comando do LINUX" ou no "PROMPT do DOS".

Existem algums programas/interfaces para facilitar a edição:

1. Recursos multiplataforma (disponíveis para diferentes sistemas operacionais tais como LINUX, WINDOWS e MAC). **Recomendamos fortemente o uso de ferramentas livres e multiplataforma.**

- Os editor xemacs ou emacs combinado com o pacote auctex facilitam e edição e compilação.
- O programa kile oferece uma interface amigável para edição de documentos LaTeX. Este programa está disponível nos terminais LINUX do LABEST/LM - C3SL. Para iniciá-lo basta digitar `kile`.
- O editor Lyx facilita e edição e compilação além de ser *semi-wysiwyg*, permitindo a pré-visualização durante a edição. Este programa está disponível nos terminais LINUX do LABEST/LM - C3SL. Para iniciá-lo basta digitar `lyx-qt`.

2. Recursos restritos ao ambiente WINDOWS

- O Tinn-R além de facilitar o uso do R também possui funcionalidades para trabalhar com documentos LaTeX.
- O programa TeXnicCenter é outra opção com interface amigável para edição de documentos LaTeXno ambiente WINDOWS.

21 Usando o Sweave

21.1 O que é e por que adotar o Sweave

O Sweave é uma funcionalidade do R implementada por algumas funções do pacote **tools** que permite a edição ágil de documentos combinando o LaTeXe o R.

Usando o Sweave o usuário pode ter comandos, saídas computacionais e/ou gráficos incluídos automaticamente no texto, sem a necessidade de fazer tal inclusão manualmente e passo a passo. Este macanismo também permite que o texto seja agilmente e automáticamente atualizado para qualquer mudança ou inclusão de dados e/ou nas análises, acelerando *muito* o processo de edição de textos.

Uma outra vantagem de extrema importância é a de que todo código usado para análise fica no arquivo texto (fonte) preservando a memória dos procedimentos usados e possibilitando a análise ser reproduzida e ou modificada facilmente e a qualquer tempo.

21.2 Usando o Sweave

Os passos básicos para uso do Sweave são:

1. Editar o arquivo `.Rnw`. Neste documento vamos supor que seu arquivo se chama `foo.Rnw`
2. Iniciar o R
3. Carregar o pacote **tools** com o comando:

   ```
   > require(tools)
   [1] TRUE
   ```

4. rodar a função `Sweave()` no seu documento com um comando do tipo: ■eval=F■ Sweave("foo.Rnw")

   Ao final destes passos, a função `Sweave()` irá imprimir uma mensagem na tela como a seguir dizendo que o documento `foo.tex` foi gerado.

   ```
   You can now run LaTeX on 'foo.tex'
   ```

   Caso outra mensagem que não esta apareça na tela algum problema deve ter ocorrido com o código R em seu documento. Leia a mensagem, identifique e corrija o erro e processe novamente com `Sweave()`.

5. Compile e visualize o documento LaTeX de forma usual.

21.3 Outras informações úteis para uso do Sweave

- O Sweave tem algumas dependências de outroas recursos no LaTeX. No caso do LINUX certifique-se que voce tem os seguintes pacotes instalados: `tetex-bin` e `tetex-extra`. No Windows a instalação do MiKTeX deve prover as ferramentas necessárias.
- A página do Sweave contém o manual, artigos, exemplos, FAQ ("Frequantly asked questions") e informações adicionais.

- Versões mais recentes do R incorporaram o comando Sweave de tal forma que é possível processar o documento `.Rnw` para gerar o `.tex` diretamente da linha de comando do LINUX sem a necessidade de iniciar o R. Para isto basta digitar o comando a seguir na linha de comando do LINUX (ou o comando análogo em outros sistemas operacionais).

  ```
  R CMD Sweave foo.Rnw
  ```

- O mecanismo descrito anteriormente substitui uma versão anterior que recomendava o uso do script Sweave.sh que também permitia rodar o Sweave no seu documento `.Rnw` diretamente da linha de comando do LINUX, sem a necessidade de iniciar o R, bastando digitar:

  ```
  Sweave.sh foo.Rnw
  ```

  Note que para o comando acima funcionar o "script"*Sweave.sh* deve estar como arquivo executável e disponível no seu `PATH`.
  Alternativamente voce pode copiá-lo para o seu diretório de trabalho e rodar com:

  ```
  ./Sweave.sh foo.Rnw
  ```

  Este arquivo deve estar em formato executável e para assegurar isto no LINUX digita-se:

  ```
  chmod +x Sweave.sh
  ```

  O *script* `Sweave.sh` foi portanto substituído pelo comando `R CMD Sweave`, mas permanece de interesse caso deseje-se modificar para adaptar à alguma necessidade específica do usuário.

- Uma outra função útil é `Stangle()` que extrai o código R de um documento `.Rnw`. Por exemplo, rodando `Stangle("foo.Rnw")` vai ser gerado um arquivo `foo.R` que contém apenas o codigo R do arquivo.

- O arquivo sweave-site.el contém as instruções necessárias para fazer o Xemacs reconhecer arquivos `.Rnw`. Isto é muito útil na preparação dos documentos pois permite também que o código em R dentro dos *chunks* seja enviado para processamento no R.

- O Sweave for concebido por Frederich Leisch da Universidade Técnica de Viena e membro do *R Core Team.*

21.4 Exemplos de arquivos em Sweave

1. Um exemplo de um arquivo `.Rnw`.
2. Arquivo com o conteúdo da seção sobre distribuições de probabilidades deste material. Para compilar este exemplo voce poderá precisar copiar também os seguintes arquivos: `Sweave.sty`, `Rd.sty` e `upquote.sty`,
3. Documento mostrando como obter tabelas estatísticas a partir do R.

21.5 Links

- Página do Sweave
- Texto sobre o Sweave por Fritz Leisch, o criador do Sweave
- Um tutotial em Espanhol
- Dicas de uso por Fernando Ferraz
- Dicas de uso por Fábio R. Mathias

Sobre este texto

Este material é produzido e disponibilizado usando exclusivamente recursos de ***SOFTWARE LIVRE***.

O texto foi editado em LaTeX e combinado com código R usando o recurso do Sweave.

A versão para WEB foi obtida convertendo o documento LaTeXpara XHTML usando o programa TeX4ht. A opção de conversão utilizada produz documentos em formato .XML que utilizam MATHML para impressão de fórmulas, equações e símbolos matemáticos.

Para visualização pela WEB sugerimos o uso do navegador Mozilla Firefox. Este documento pode não ser bem visualizado em alguns navegadores que não possuam suporte a MATHML.

Se seu navegador não suporta MATHML (por exemplo Internet Explorer) voce pode habilitar este suporte instalando o MathPlayer.

Todo o material foi produzido em ambiente Debian-Linux e/ou Ubuntu-Linux. A página WEB é disponibilizada usando um servidor APACHE rodando em um Debian-Linux.