# Introdução ao sistema estatístico R
# Mini-curso EMBRAPA

Paulo Justiniano Ribeiro Junior

Brasília, 30/05 a 03/06 de 2005 (última revisão: 17 de julho de 2007)

Estas notas estão disponíveis em formato HTML em `http://www.leg.ufpr.br/~paulojus/embrapa/Rembrapa` e também em no arquivo arquivo em formato PDF.

Este curso foi montado visando uma *introdução ao sistema estatístico* R para profissionais da EMBRAPA. O objetivo é ilustrar aspectos básicos do sistema com ênfase na compreensão de aspectos básicos da linguagem, a estrutura e a forma de operar o programa. O curso não tem o objetivo de discutir em detalhe nenhum método e/ou modelo estatístico em particular. Métodos estatísticos básicos são usados ao longo do texto simplesmente para ilustrar o uso da linguagem.

Será assumida apenas familiaridade com conceitos e métodos estatísticos básicos. Não será assumido nenhum conhecimento prévio do R. O curso foi preparado e será ministrado em ambiente LINUX porém não fará uso de nenhum recurso específico deste sistema operacional e participantes poderão acompanhar usando outro sistema operacional, tal como Windows®.

Vamos começar "experimentando o R", para ter uma idéia de seus recursos e a forma de trabalhar com este programa. Para isto vamos rodar e estudar os comandos mostrados no texto e seus resultados para nos familiarizar com aspectos básicos do programa. Ao longo deste curso iremos ver com mais detalhes o uso do programa R.

Siga os seguintes passos:

1. inicie o R em seu computador;
2. voce verá uma janela de comandos com o símbolo >,
   este é o *prompt* do R indicando que o programa está pronto para receber comandos;
3. a seguir digite (ou "recorte e cole") os comandos mostrados ao longo deste material.

No restante deste texto vamos seguir as seguintes convenções.

- comandos do R são mostrados em fontes do tipo ***`slanted verbatim como esta`***, e precedidas pelo símbolo >,
- saídas do R são sempre exibidas em fontes do tipo `verbatim como esta`,
- linhas iniciadas pelo símbolo # são comentários e são ignoradas pelo R.

# 1 Uma primeira sessão com o R

Esta é uma primeira sessão com o R visando dar aos participantes uma idéia geral da aparência e forma de operação do programa. Os comandos abaixo serão reproduzidos e comentados durante o curso.

Vamos começar gerando dois vetores `x` e `y` de coordenadas geradas a partir de números pseudo-aleatórios e depois inspecionar os valores gerados.

```
> x <- rnorm(5)
> x

[1]  1.8614407 -1.0874200 -0.5615027 -2.3187178  0.3776864

> print(x)

[1]  1.8614407 -1.0874200 -0.5615027 -2.3187178  0.3776864

> print(x, dig = 3)

[1]  1.861 -1.087 -0.562 -2.319  0.378

> y <- rnorm(x)
> y

[1]  0.1432350  0.5101738 -0.2760532 -0.2362307  1.1996061

> args(rnorm)

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

No exemplo acima primeiro geramos um *vetor* `x` com 5 elementos. Note que ao fazermos `y <- rnorm(x)` nao especificamos o tamanho da amostra explicitamente como anteriormente mas estamos definindo um vetor `y` que tem o mesmo tamanho de `x`, por isto `y` foi gerado com também 5 elementos. Note que se voce tentar reproduzir este exemplo deve obter valores simulados diferentes dos mostrados aqui.

Ao digitar o nome do objeto `x` os elementos deste objetos são exibidos. O comando `print(x)` também exibe os elementos do objeto porém é mais flexível pois oferece opções extras de visualização. O comando `print(x, dig=3)` exibe este particular objeto `x` com no mínimo 3 dígitos significativos. Para controlar o número de dígitos globalmente, isto é, para impressão de qualquer objeto, por exemplo com 4 dígitos, usamos `options(digits=4)`.

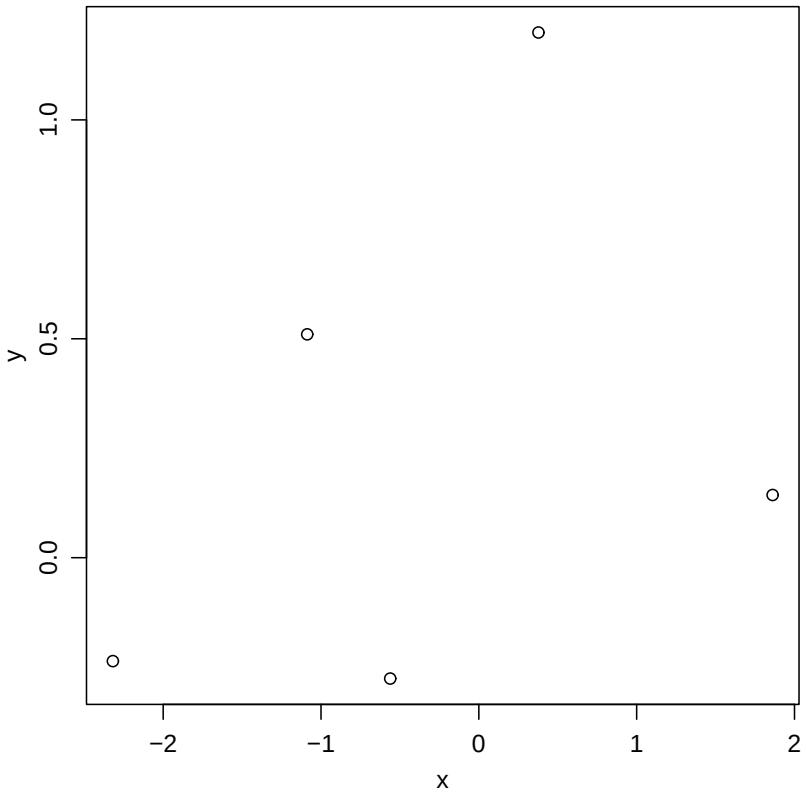
Neste simples exemplo introduzimos várias idéias e conceitos: *objeto*, *atribuição de valores*, *vetores*, *impressão de objetos*, *função*, *argumentos de funções*, *"defaults"*, *geração de números aleatórios* e *controle de semente*.

Agora vamos colocar num gráfico os pontos gerados usando o comando

```
> plot(x, y)
```

Note que a janela gráfica se abrirá automaticamente e exibirá o gráfico. Há muitas opções de controle e configuração da janela gráfica que são especidicadas usando-se a função `par()`. Algumas destas opções serão vistas ao longo deste material.

A função `plot()` oferece através de seus argumentos várias opções para visualização dos gráficos. As argumentos e básicos são mostrados a seguir.




```
> args(plot.default)

function (x, y = NULL, type = "p", xlim = NULL, ylim = NULL,
    log = "", main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
    ann = par("ann"), axes = TRUE, frame.plot = axes, panel.first = NULL,
    panel.last = NULL, asp = NA, ...)
NULL
```

Para ilustração, no exemplo a seguir mostramos o uso do argumento `type`. Para facilitar esta ilustração vamos primeiro ordenar os valores de `x` e `y` na sequência crescente dos valores de `x`.

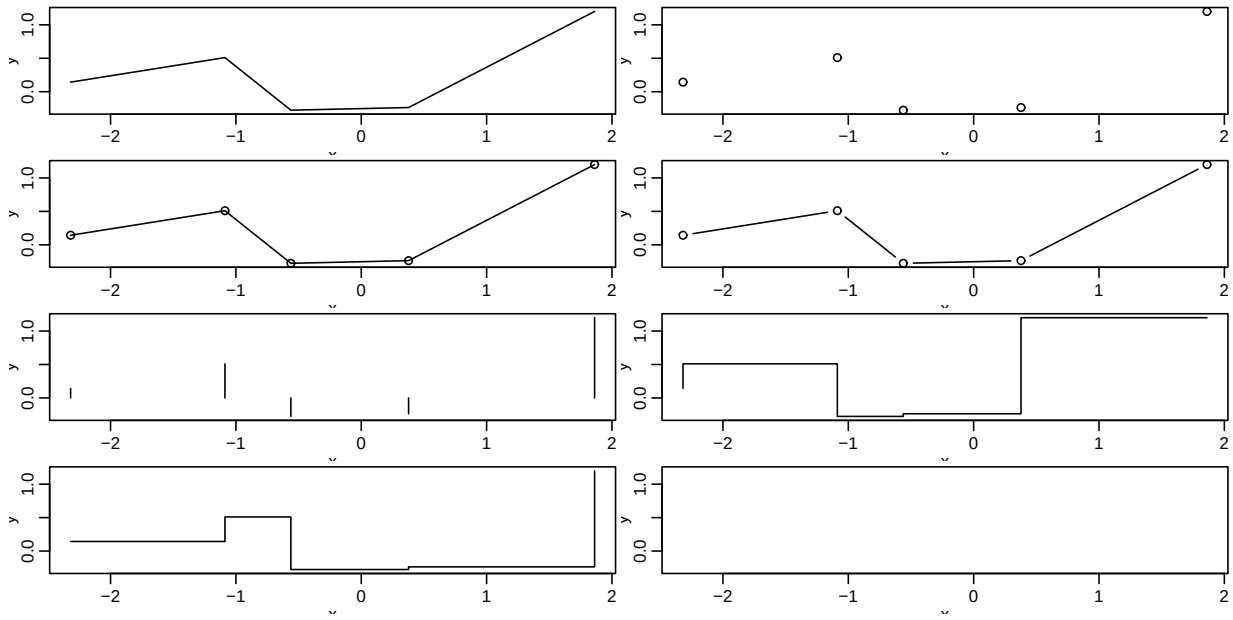
```
> x <- sort(x)
> y <- y[order(x)]
```

Nos comandos abaixo iniciamos dividindo a janela gráfica em 8 partes e reduzindo as margens do gráfico. A seguir produzimos diversos gráficos com diferentes opções para o argumento `type`. Ao final retornamos a configuração original de apenas um gráfico na janela gráfica.

Um pouco mais sobre manipulação de vetores. Note que os colchetes `[]` são usados para selecionar elementos e há funções para arredondar valores.

```
> x

[1] -2.3187178 -1.0874200 -0.5615027  0.3776864  1.8614407

> x[1]

[1] -2.318718

> x[3]
```

```
> par(mfrow = c(4, 2), mar = c(2, 2, 0.3, 0.3), mgp = c(1.5, 0.6,
+     0))
> plot(x, y, type = "l")
> plot(x, y, type = "p")
> plot(x, y, type = "o")
> plot(x, y, type = "b")
> plot(x, y, type = "h")
> plot(x, y, type = "S")
> plot(x, y, type = "s")
> plot(x, y, type = "n")
> par(mfrow = c(1, 1))
```



```
[1] -0.5615027

> x[2:4]

[1] -1.0874200 -0.5615027  0.3776864

> round(x, dig = 1)

[1] -2.3 -1.1 -0.6  0.4  1.9

> ceiling(x)

[1] -2 -1  0  1  2

> floor(x)

[1] -3 -2 -1  0  1

> trunc(x)

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

Os objetos exixtentes na área de trabalho pode ser listados usando a função `ls()` e objetos podem ser removidos com a função `rm()`. Nos comandos a seguir estamos verificando os objetos existentes na área de trabalho e removendo objetos que julgamos não mais necessários.

```
> ls()

[1] "x" "y"

> rm(x, y)
```

A seguir vamos criar um vetor que chamaremos de `x` com uma sequência de números de 1 a 20. Depois criamos um vetor `w` de pesos com os desvios padrões de cada observação. Na sequência montamos um *data-frame* de 3 colunas com variáveis que chamamos de `x`, `y` e `w`. Inspecionando o conteúdo do objeto criado digitando o seu nome. A terminamos apagando objetos que não são mais necessários.

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

 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20

> w <- 1 + sqrt(x)/2
> w

 [1] 1.500000 1.707107 1.866025 2.000000 2.118034 2.224745 2.322876 2.414214 2.500000
[10] 2.581139 2.658312 2.732051 2.802776 2.870829 2.936492 3.000000 3.061553 3.121320
[19] 3.179449 3.236068

> dummy <- data.frame(x = x, y = x + rnorm(x) * w, w = w)
> dummy

    x         y        w
1   1  2.148754 1.500000
2   2  1.659649 1.707107
3   3  1.711935 1.866025
4   4  3.111563 2.000000
5   5  5.342233 2.118034
6   6  4.383622 2.224745
7   7  3.954104 2.322876
8   8  7.896386 2.414214
9   9 10.505363 2.500000
10 10 10.535822 2.581139
11 11 12.522613 2.658312
12 12 11.747249 2.732051
13 13 15.556417 2.802776
14 14 10.148046 2.870829
15 15 14.245631 2.936492
16 16 17.722934 3.000000
17 17 19.053369 3.061553
18 18 25.597813 3.121320
19 19 17.851351 3.179449
20 20 26.432684 3.236068
```

```
> rm(x, w)
```

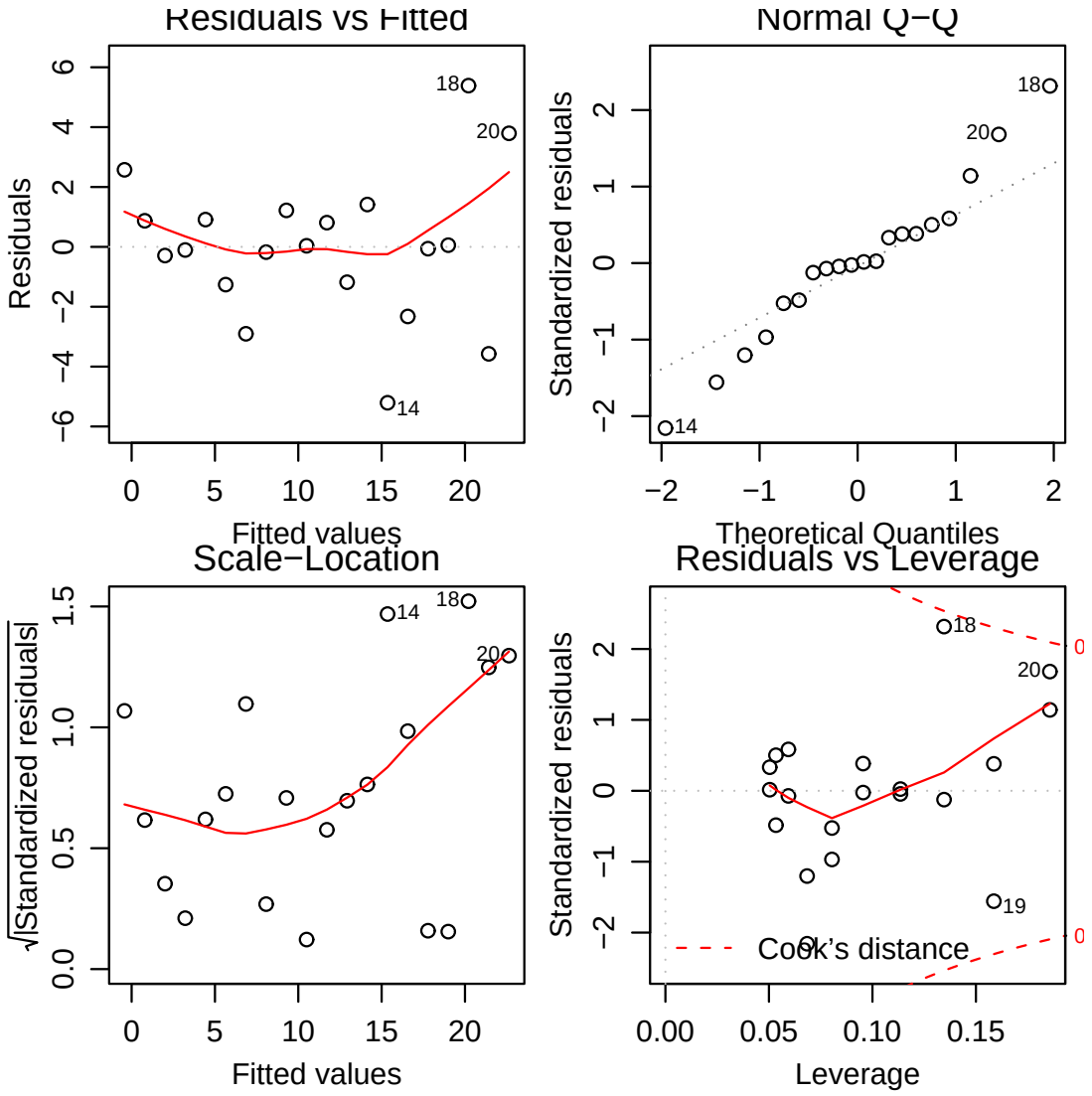
Nos comandos a seguir estamos ajustando uma regressão linear simples de `y` em `x` e examinando os resultados. Na sequência, uma vez que temos valores dos pesos, podemos fazer uma regressão ponderada e comparar os resultados.

```
> fm <- lm(y ~ x, data = dummy)
> summary(fm)

Call:
lm(formula = y ~ x, data = dummy)

Residuals:
     Min       1Q   Median       3Q      Max
-5.20702 -1.20003 -0.01178  0.98924  5.38711

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.63969    1.16188  -1.411    0.175
x            1.21391    0.09699  12.516 2.56e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.501 on 18 degrees of freedom
Multiple R-Squared: 0.8969,        Adjusted R-squared: 0.8912
F-statistic: 156.6 on 1 and 18 DF,  p-value: 2.556e-10

> fm1 <- lm(y ~ x, data = dummy, weight = 1/w^2)
> summary(fm1)

Call:
lm(formula = y ~ x, data = dummy, weights = 1/w^2)

Residuals:
     Min       1Q   Median       3Q      Max
-1.74545 -0.50251  0.03886  0.33719  1.87258

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.92001    0.82522  -1.115    0.280
x            1.14849    0.08414  13.649 6.18e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.9119 on 18 degrees of freedom
Multiple R-Squared: 0.9119,        Adjusted R-squared: 0.907
F-statistic: 186.3 on 1 and 18 DF,  p-value: 6.185e-11
```

Gráficos de resíduos são produzidos com `plot()`. Como a função produz 4 gráficos dividiremos a tela gráfica,

Note que o comando acima `par(mfrow=c(2,2))` dividiu a janela gráfica em 4 partes para acomodar os 4 gráficos. Para restaurar a configuração original usamos

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



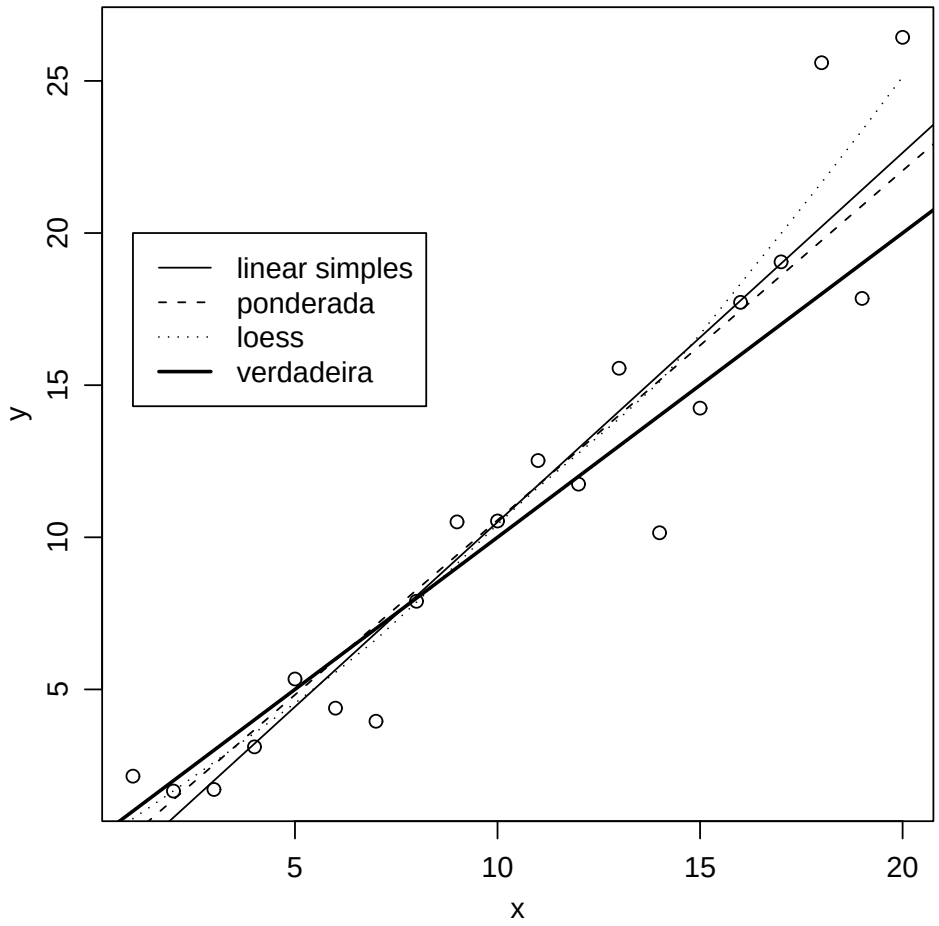

```
> par(mfrow = c(1, 1))
```

Tornando visíveis as colunas do data-frame.

```
> search()

 [1] ".GlobalEnv"        "package:tools"     "package:stats"     "package:graphics"
 [5] "package:grDevices" "package:utils"     "package:datasets"  "package:methods"
 [9] "Autoloads"         "package:base"

> attach(dummy)
> search()

 [1] ".GlobalEnv"        "dummy"             "package:tools"     "package:stats"
 [5] "package:graphics"  "package:grDevices" "package:utils"     "package:datasets"
 [9] "package:methods"   "Autoloads"         "package:base"
```

Fazendo uma regressão local não-paramétrica, e visualizando o resultado. Depois adicionamos a linha de regressão verdadeira (intercepto 0 e inclinação 1), a linha da regressão sem ponderação e a linha de regressão ponderada.

```
> lrf <- lowess(x, y)
> plot(x, y)
> lines(lrf, lty = 3)
> abline(coef(fm))
> abline(coef(fm1), lty = 2)
> abline(0, 1, lwd = 2)
> legend(1, 20, c("linear simples", "ponderada", "loess", "verdadeira"),
+     lty = c(1, 2, 3, 1), lwd = c(1, 1, 1, 2))
```



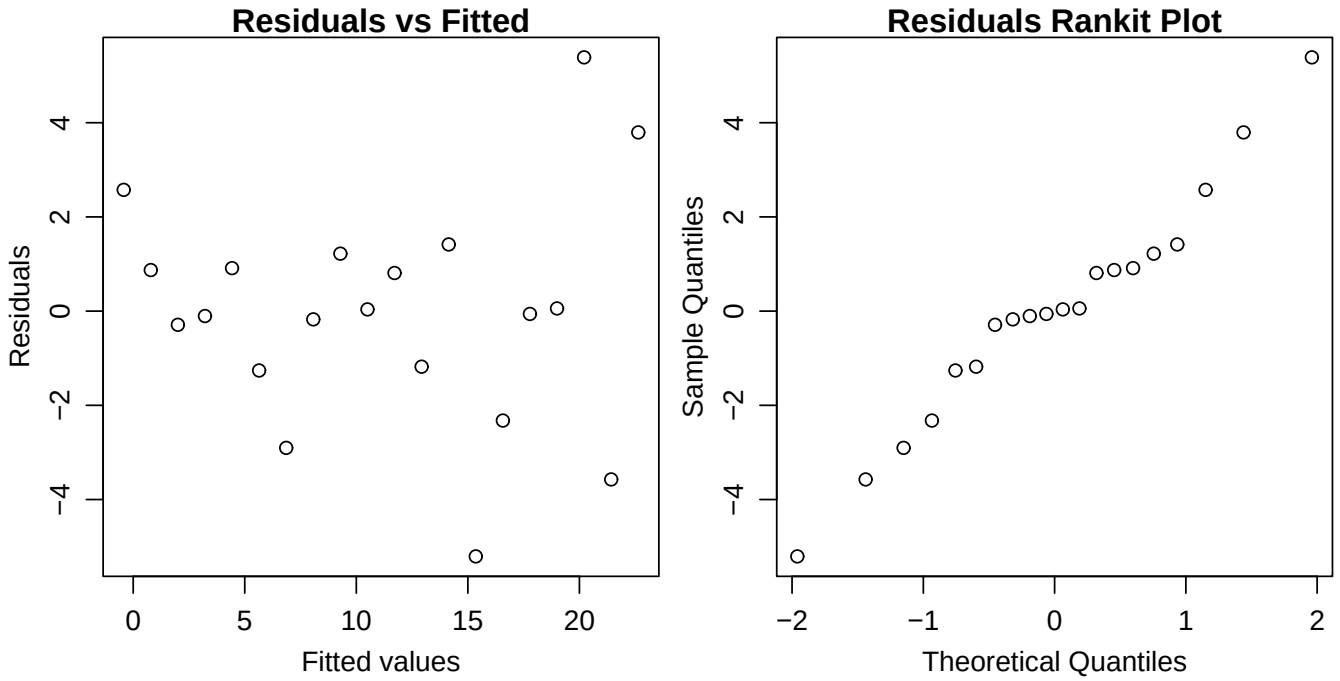

Ao final destas análises removemos o objeto `dummy` do caminho de procura.

```
> detach()
```

Agora vamos fazer um gráfico diagnóstico padrão para checar ajuste e pressupostos: o gráfico de resíduos por valores preditos e gráfico de escores normais para checar assimetria, curtose e outliers (não muito útil aqui).

```
> par(mfrow = c(1, 2))
> plot(fitted(fm), resid(fm), xlab = "Fitted values", ylab = "Residuals",
+     main = "Residuals vs Fitted")
> qqnorm(resid(fm), main = "Residuals Rankit Plot")
```

E ao final retornamos ao gráfico padrão e ”limpamos”novamente o *workspace*, ou seja, apagando objetos.




```
> par(mfrow = c(1, 1))
> rm(fm, fm1, lrf, dummy)
```

Agora vamos inspecionar dados do experimento clássico de Michaelson e Morley para medir a velocidade da luz. Clique para ver o arquivo morley.tab de dados no formato texto. Se quiser voce pode ainda fazer o *download* deste arquivo para o seu micro. Pode-se visualizar um arquivo externo dentro do próprio R utilizando `file.show()` e note que no comando abaixo assume-se que o arquivo está na área de trabalho do R, caso contrário deve ser precedido do caminho para o diretório adequado.

```
> file.show("morley.tab")
```

Lendo dados como um "data-frame"e inspecionando seu conteúdo. Há 5 experimentos (coluna Expt) e cada um com 20 "rodadas"(coluna Run) e sl é o valor medido da velocidade da luz numa escala apropriada

```
> mm <- read.table("http://www.leg.ufpr.br/~paulojus/embrapa/morley.tab")
> mm
```

```
    Expt Run Speed
001    1   1   850
002    1   2   740
003    1   3   900
004    1   4  1070
005    1   5   930
006    1   6   850
007    1   7   950
008    1   8   980
009    1   9   980
010    1  10   880
011    1  11  1000
012    1  12   980
```

| | | | |
|---|---|---|---|
| 013 | 1 | 13 | 930 |
| 014 | 1 | 14 | 650 |
| 015 | 1 | 15 | 760 |
| 016 | 1 | 16 | 810 |
| 017 | 1 | 17 | 1000 |
| 018 | 1 | 18 | 1000 |
| 019 | 1 | 19 | 960 |
| 020 | 1 | 20 | 960 |
| 021 | 2 | 1 | 960 |
| 022 | 2 | 2 | 940 |
| 023 | 2 | 3 | 960 |
| 024 | 2 | 4 | 940 |
| 025 | 2 | 5 | 880 |
| 026 | 2 | 6 | 800 |
| 027 | 2 | 7 | 850 |
| 028 | 2 | 8 | 880 |
| 029 | 2 | 9 | 900 |
| 030 | 2 | 10 | 840 |
| 031 | 2 | 11 | 830 |
| 032 | 2 | 12 | 790 |
| 033 | 2 | 13 | 810 |
| 034 | 2 | 14 | 880 |
| 035 | 2 | 15 | 880 |
| 036 | 2 | 16 | 830 |
| 037 | 2 | 17 | 800 |
| 038 | 2 | 18 | 790 |
| 039 | 2 | 19 | 760 |
| 040 | 2 | 20 | 800 |
| 041 | 3 | 1 | 880 |
| 042 | 3 | 2 | 880 |
| 043 | 3 | 3 | 880 |
| 044 | 3 | 4 | 860 |
| 045 | 3 | 5 | 720 |
| 046 | 3 | 6 | 720 |
| 047 | 3 | 7 | 620 |
| 048 | 3 | 8 | 860 |
| 049 | 3 | 9 | 970 |
| 050 | 3 | 10 | 950 |
| 051 | 3 | 11 | 880 |
| 052 | 3 | 12 | 910 |
| 053 | 3 | 13 | 850 |
| 054 | 3 | 14 | 870 |
| 055 | 3 | 15 | 840 |
| 056 | 3 | 16 | 840 |
| 057 | 3 | 17 | 850 |
| 058 | 3 | 18 | 840 |
| 059 | 3 | 19 | 840 |
| 060 | 3 | 20 | 840 |
| 061 | 4 | 1 | 890 |
| 062 | 4 | 2 | 810 |

```
063    4   3   810
064    4   4   820
065    4   5   800
066    4   6   770
067    4   7   760
068    4   8   740
069    4   9   750
070    4  10   760
071    4  11   910
072    4  12   920
073    4  13   890
074    4  14   860
075    4  15   880
076    4  16   720
077    4  17   840
078    4  18   850
079    4  19   850
080    4  20   780
081    5   1   890
082    5   2   840
083    5   3   780
084    5   4   810
085    5   5   760
086    5   6   810
087    5   7   790
088    5   8   810
089    5   9   820
090    5  10   850
091    5  11   870
092    5  12   870
093    5  13   810
094    5  14   740
095    5  15   810
096    5  16   940
097    5  17   950
098    5  18   800
099    5  19   810
100    5  20   870
```

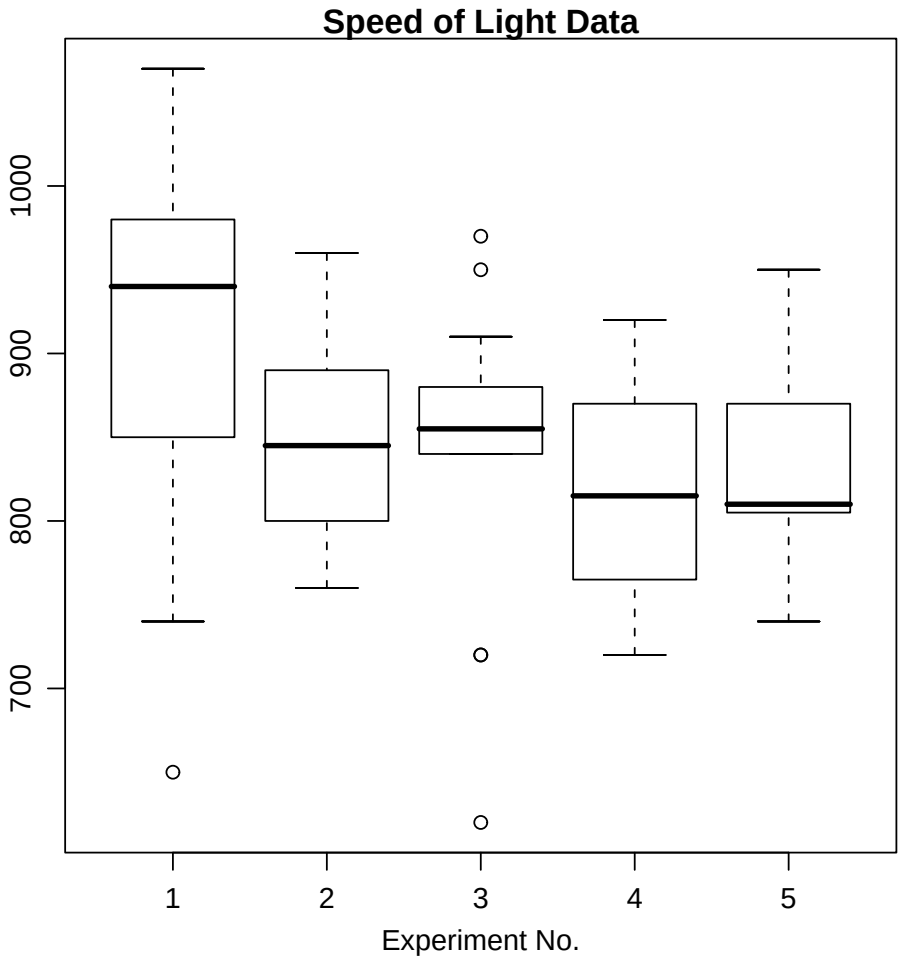
Devemos definir `Expt` e `Run` como fatores tornar o data-frame visível na posição 2 do caminho de procura.

```
> mm$Expt <- factor(mm$Expt)
> mm$Run <- factor(mm$Run)
> attach(mm)
```

Podemos fazer um gráfico para comparar visualmente os 5 experimentos

```
> plot(Expt, Speed, main = "Speed of Light Data", xlab = "Experiment No.")
```

Depois analisamos como um experimento em blocos ao acaso com `Run` e `Expt` como fatores e inspecionamos os resultados.




```
> fm <- aov(Speed ~ Run + Expt, data = mm)
> summary(fm)

            Df Sum Sq Mean Sq F value   Pr(>F)
Run         19 113344    5965  1.1053 0.363209
Expt         4  94514   23629  4.3781 0.003071 **
Residuals   76 410166    5397
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

> names(fm)

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

> fm$coef

  (Intercept)          Run2          Run3          Run4          Run5          Run6
 9.506000e+02 -5.200000e+01 -2.800000e+01  6.000000e+00 -7.600000e+01 -1.040000e+02
         Run7          Run8          Run9         Run10         Run11         Run12
-1.000000e+02 -4.000000e+01 -1.000000e+01 -3.800000e+01  4.000000e+00 -1.737634e-13
        Run13         Run14         Run15         Run16         Run17         Run18
-3.600000e+01 -9.400000e+01 -6.000000e+01 -6.600000e+01 -6.000000e+00 -3.800000e+01
```

```
        Run19         Run20         Expt2         Expt3         Expt4         Expt5
-5.000000e+01 -4.400000e+01 -5.300000e+01 -6.400000e+01 -8.850000e+01 -7.750000e+01
```

Podemos redefinir o modelo, por exemplo ajustando um sub-modelo sem o fator "runs" e comparar os dois modelos lineares via uma análise de variância.

```
> fm0 <- update(fm, . ~ . - Run)
> anova(fm0, fm)

Analysis of Variance Table

Model 1: Speed ~ Expt
Model 2: Speed ~ Run + Expt
  Res.Df    RSS Df Sum of Sq      F Pr(>F)
1     95 523510
2     76 410166 19    113344 1.1053 0.3632
```

É importante saber interpretar os coeficientes segunda a parametrização utilizada. Por *default* a parametrização é feita tomando o primeiro grupo como referência.

```
> fm0$coef

(Intercept)       Expt2       Expt3       Expt4       Expt5
      909.0       -53.0       -64.0       -88.5       -77.5

> mds <- tapply(Speed, Expt, mean)
> mds

    1     2     3     4     5
909.0 856.0 845.0 820.5 831.5

> mds[-1] - mds[1]

    2     3     4     5
-53.0 -64.0 -88.5 -77.5
```

E este comportamento é controlado por `options()`. Por exemplo, contrastes de Helmert são definidos como se segue.

```
> options()$contrast

        unordered           ordered
"contr.treatment"      "contr.poly"

> options(contrasts = c("contr.helmert", "contr.poly"))
> fm0 <- update(fm, . ~ . - Run)
> fm0$coef

(Intercept)       Expt1       Expt2       Expt3       Expt4
    852.400     -26.500     -12.500     -12.375      -5.225

> mean(Speed)
```

```
[1] 852.4

> (mds[2] - mds[1])/2

    2
-26.5

> (2 * mds[3] - mds[1] - mds[2])/6

    3
-12.5

> (3 * mds[4] - mds[1] - mds[2] - mds[3])/12

      4
-12.375

> (4 * mds[5] - mds[1] - mds[2] - mds[3] - mds[4])/20

     5
-5.225
```

Enquanto que contrastes de cada tratamento contra a média geral são obtidos da forma:

```
> options(contrasts = c("contr.sum", "contr.poly"))
> fm0 <- update(fm, . ~ . - Run)
> fm0$coef

(Intercept)       Expt1       Expt2       Expt3       Expt4
      852.4        56.6         3.6        -7.4       -31.9

> mds - mean(Speed)

    1     2     3     4     5
 56.6   3.6  -7.4 -31.9 -20.9
```

Há algumas opções de contrastes implementadas no R e além disto o usuário pode implementar contrastes de sua preferência. Para entender melhor os resultados acima analise as saídas dos comandos abaixo.

```
> contr.treatment(5)

  2 3 4 5
1 0 0 0 0
2 1 0 0 0
3 0 1 0 0
4 0 0 1 0
5 0 0 0 1

> contr.helmert(5)
```

```
  [,1] [,2] [,3] [,4]
1   -1   -1   -1   -1
2    1   -1   -1   -1
3    0    2   -1   -1
4    0    0    3   -1
5    0    0    0    4
```

```
> contr.sum(5)

  [,1] [,2] [,3] [,4]
1    1    0    0    0
2    0    1    0    0
3    0    0    1    0
4    0    0    0    1
5   -1   -1   -1   -1
```

```
> contr.poly(5)

                .L         .Q            .C         ^4
[1,] -6.324555e-01  0.5345225 -3.162278e-01  0.1195229
[2,] -3.162278e-01 -0.2672612  6.324555e-01 -0.4780914
[3,] -3.287978e-17 -0.5345225  1.595204e-16  0.7171372
[4,]  3.162278e-01 -0.2672612 -6.324555e-01 -0.4780914
[5,]  6.324555e-01  0.5345225  3.162278e-01  0.1195229
```

Se ainda não estivar claro experimente para cada uma destas examinar a matrix do modelo com os comandos abaixo (saídas não são mostradas aqui).

```
> options(contrasts = c("contr.treatment", "contr.poly"))
> model.matrix(Speed ~ Expt)
> options(contrasts = c("contr.helmert", "contr.poly"))
> model.matrix(Speed ~ Expt)
> options(contrasts = c("contr.sum", "contr.poly"))
> model.matrix(Speed ~ Expt)
```

Ao final desanexamos o objeto e limpamos novamente o *workspace*.

```
> detach()
> rm(fm, fm0)
```

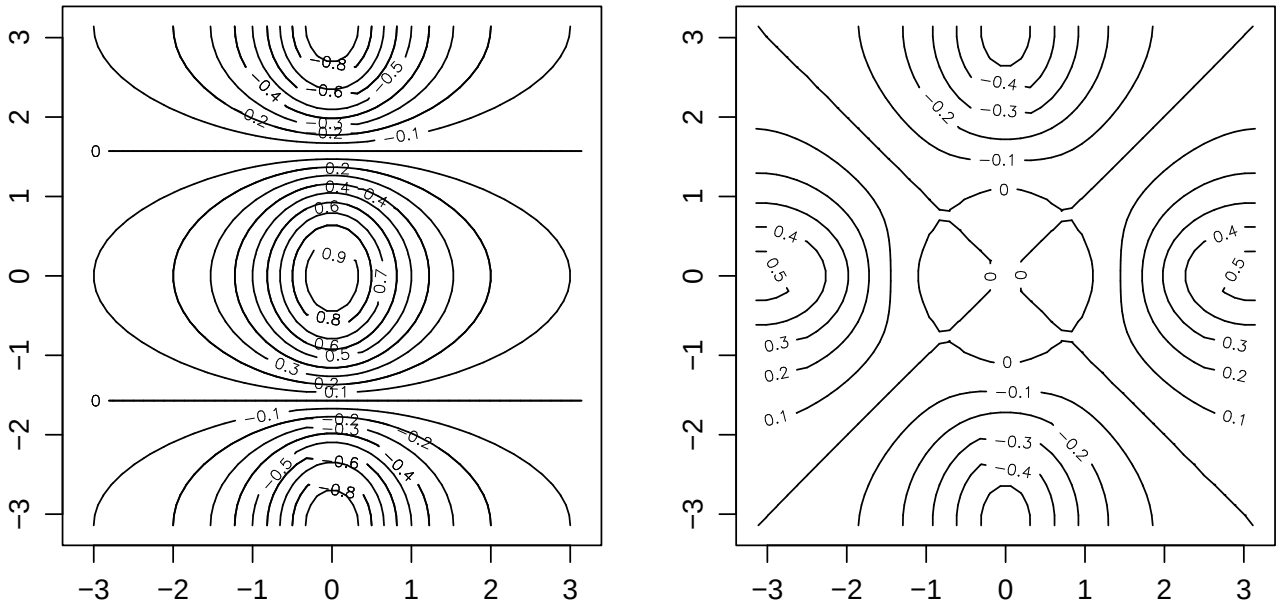
Vamos agora ver alguns gráficos gerados pelas funções `contour()` e `image()`.

No próximo exemplo `x` é um vetor de 50 valores igualmente espaçados no intervalo [-pi pi]. `y` idem. O objeto `f` é uma matrix quadrada com linhas e colunas indexadas por `x` e `y` respectivamente com os valores da função $cos(y)/(1 + x^2)$.

```
> x <- seq(-pi, pi, len = 50)
> y <- x
> f <- outer(x, y, function(x, y) cos(y)/(1 + x^2))
```

Agora gravamos parâmetros gráficos e definindo a região gráfica como quadrada e fazemos um mapa de contorno de `f`. Depois adicionamos mais linhas para melhor vizualização. `fa` é a "parte assimétrica" e `t()` é transposição. Ao final e restauramos os parâmetros gráficos iniciais.

```
> oldpar <- par(no.readonly = TRUE)
> par(pty = "s", mfrow = c(1, 2))
> contour(x, y, f)
> contour(x, y, f, nlevels = 15, add = TRUE)
> fa <- (f - t(f))/2
> contour(x, y, fa, nlevels = 15)
> par(oldpar)
```



Fazendo um gráfico de imagem

```
> oldpar <- par(no.readonly = TRUE)
> par(pty = "s", mfrow = c(1, 2))
> image(x, y, f)
> image(x, y, fa)
> par(oldpar)
```

E apagando objetos novamente antes de prosseguir.

```
> objects()
```

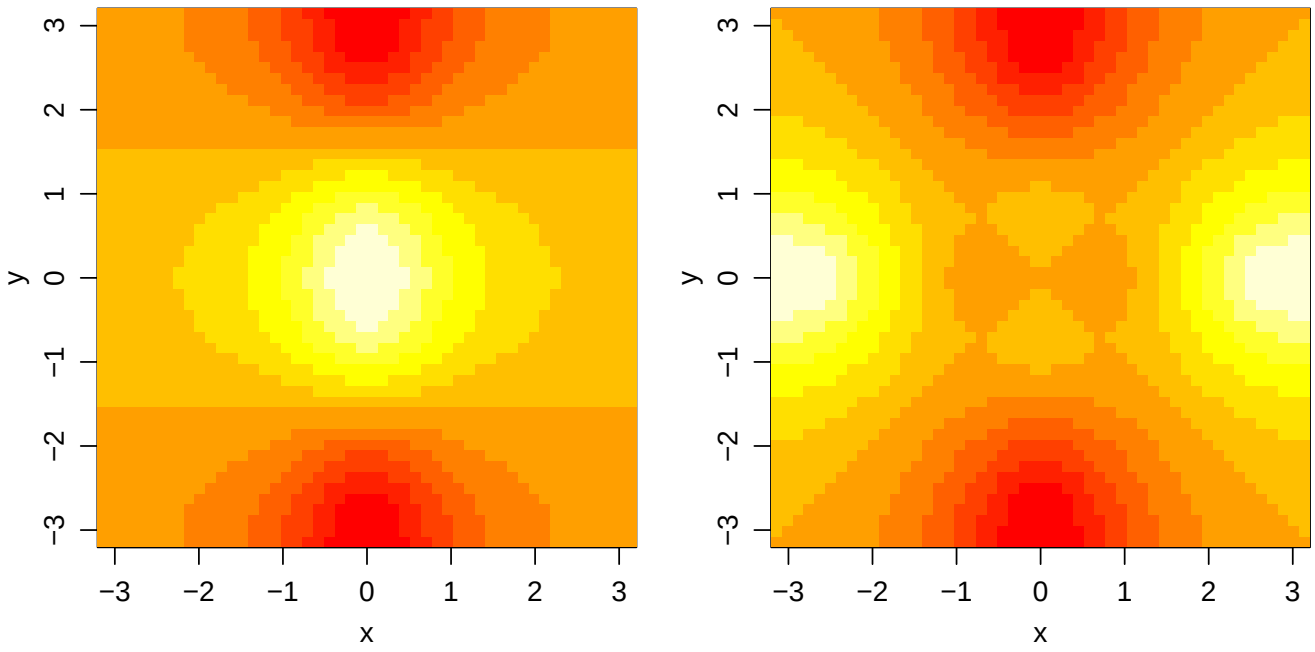
```
[1] "f"      "fa"     "mds"    "mm"     "oldpar" "x"      "y"
```

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

Para encerrar esta sessão vejamos mais algumas funcionalidades do R. O R pode fazer operação com complexos, note que 1i denota o número complexo i.

```
> th <- seq(-pi, pi, len = 100)
> z <- exp((0+1i) * th)
```

Plotando complexos significa parte imaginária versus real Isto deve ser um círculo: Suponha que desejamos amostrar pontos dentro do círculo de raio unitário. uma forma simples de fazer isto é tomar números complexos com parte real e imaginária padrão. E depois mapeamos qualquer externo ao círculo no seu recíproco:




```
> par(pty = "s")
> plot(z, type = "l")
> w <- rnorm(100) + rnorm(100) * (0+1i)
> w <- ifelse(Mod(w) > 1, 1/w, w)
```

Desta forma todos os pontos estão dentro do círculo unitário, mas a distribuição não é uniforme. Um segundo método usa a distribuição uniforme. os pontos devem estar melhor distribuídos sobre o círculo

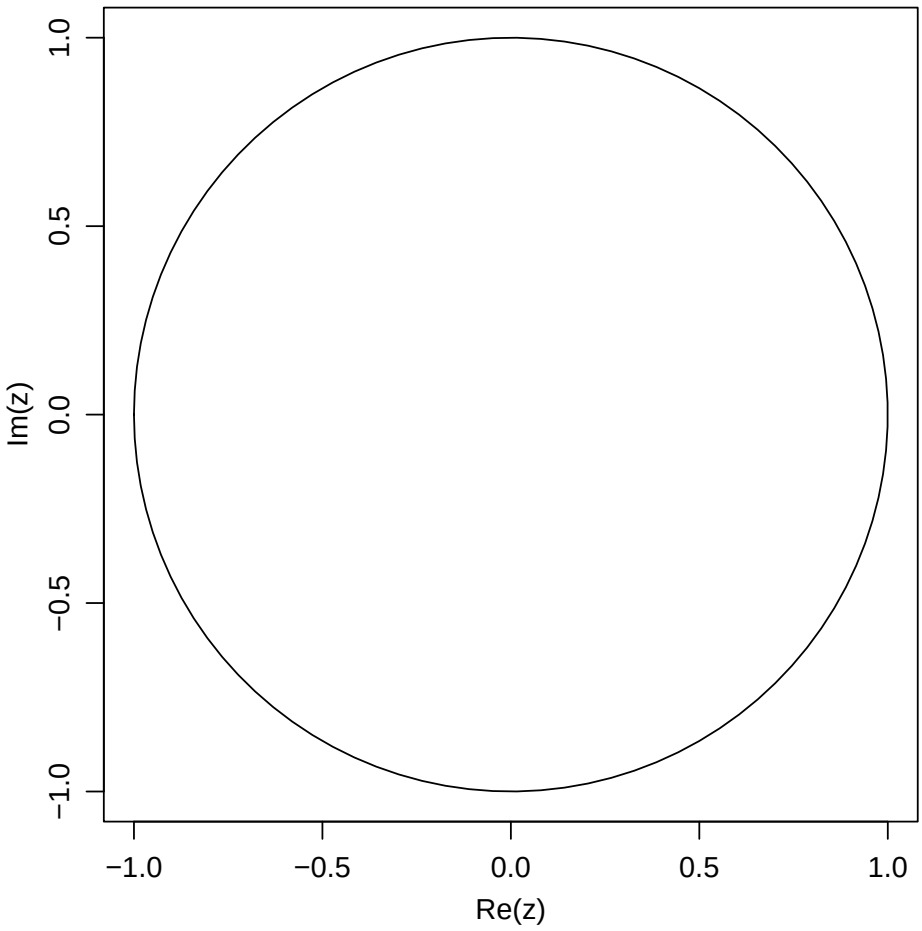
```
> plot(w, xlim = c(-1, 1), ylim = c(-1, 1), pch = "+", xlab = "x",
+     ylab = "y")
> lines(z)
> w <- sqrt(runif(100)) * exp(2 * pi * runif(100) * (0+1i))
> plot(w, xlim = c(-1, 1), ylim = c(-1, 1), pch = "+", xlab = "x",
+     ylab = "y")
> lines(z)
```

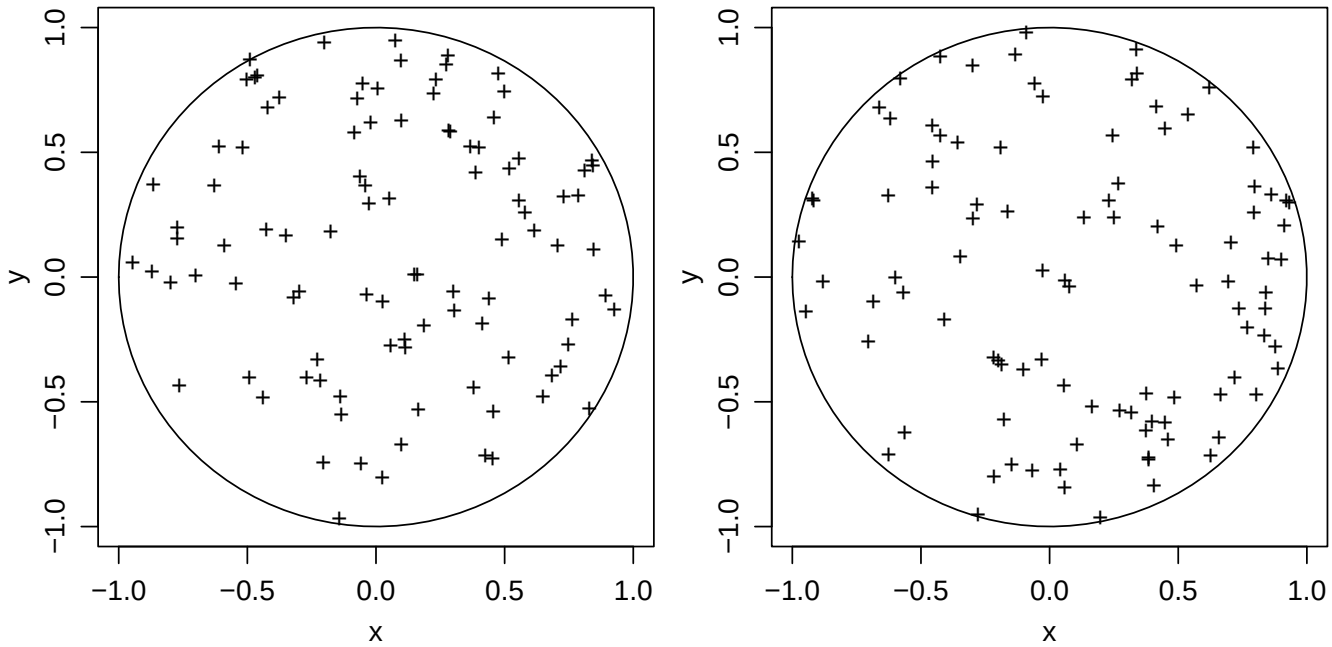
Apagamos novamente os objetos ...

```
> rm(th, w, z)
```

... e saímos do R.

```
q()
```




# 2 Estatística computacional e o sistema R

Nesta seção iremos seguir a apresentação disponível no arquivo `estcompR.pdf`

# 3 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

# 4 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.

## 4.1 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.

## 4.2 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.

## 4.3 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.

## 4.4 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.

## 4.5 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**.

# 5 Aritmética e Objetos

## 5.1 Operações aritméticas

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

```
> 1 + 2 + 3

[1] 6

> 2 + 3 * 4

[1] 14

> 3/2 + 1

[1] 2.5

> 4 * 3^3

[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)

[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
```

| | |
|---|---|
| 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 | logarítmo base-10 |

## 5.2 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
```

## 5.3 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)
> x

[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)
> y + x

[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 $\pi$) 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
```

# 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 concaternar) 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] 78 76 75 86 80 88 69 73 69 66
```

```
> sum(x9)

[1] 760

> mean(x9)

[1] 76

> var(x9)

[1] 52.44444

> min(x9)

[1] 66

> max(x9)

[1] 88

> summary(x9)

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   66.0    70.0    75.5    76.0    79.5    88.0

> fivenum(x9)

[1] 66.0 69.0 75.5 80.0 88.0
```

**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] 78 76 75

> x9[2 * (1:5)]
```

```
[1] 76 86 88 73 66
```

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  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE
> as.numeric(ind.72)
 [1] 1 1 1 1 1 1 0 1 0 0
> x10 <- x9[ind.72]
> x10
[1] 78 76 75 86 80 88 73
```

**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] 46 80 63 75 60 73 76

> rev(vec)

[1] 76 73 60 75 63 80 46

> sort(vec)

[1] 46 60 63 73 75 76 80

> order(vec)

[1] 1 5 3 6 4 7 2

> vec[order(vec)]

[1] 46 60 63 73 75 76 80

> rank(vec)

[1] 1 7 3 5 2 4 6

> vec%%5

[1] 1 0 3 0 0 3 1

> vec%/%5

[1]  9 16 12 15 12 14 15
```

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

```
> which(vec > 70)
```

```
[1] 2 4 6 7

> which.max(vec)

[1] 2

> which.min(vec)

[1] 1
```

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

```
> vec

[1] 46 80 63 75 60 73 76

> vec[-5]

[1] 46 80 63 75 73 76

> vec[-(2:4)]

[1] 46 60 73 76
```

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:

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

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.

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.

Tabela 1: 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 |

```
> 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   8.027908   1
2   7.333096   1
3  11.680884   1
4  10.060119   1
5  10.121508   1
6  17.967366   2
7  18.493815   2
8  16.060282   2
9  15.738959   2
10 18.925635   2
11 11.790096   3
12 16.055760   3
13 13.611893   3
14 13.838306   3
15 14.370944   3

> by(d2$Y, d2$lev, summary)

INDICES: 1
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  7.333   8.028  10.060   9.445  10.120  11.680
------------------------------------------------------------
INDICES: 2
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  15.74   16.06   17.97   17.44   18.49   18.93
------------------------------------------------------------
INDICES: 3
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  11.79   13.61   13.84   13.93   14.37   16.06

> 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  160.50881  30.09839
Deg. of Freedom         2        12

Residual standard error: 1.583730
Estimated effects may be unbalanced

> summary(lis3)

            Df  Sum Sq Mean Sq F value    Pr(>F)
lev          2 160.509  80.254  31.997 1.550e-05 ***
Residuals   12  30.098   2.508
---
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 $3 \times 2$.

```
> 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 $10 \times 10$ 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) $1 + 1/2 + 1/3 + 1/4 + \ldots$
   - (b) $1 + 1/22 + 1/42 + 1/62 + 1/82 + \ldots$
   - (c) $1/(1+1/1!)2 + 1/(1+1/2!)2 + 1/(1+1/3!)2 + \ldots$
   - (d) $1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + \ldots$

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 Miscelânia de funcionalidades do R

## 7.1 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
```

## 7.2 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 7.2 e 7.2.

(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")
```

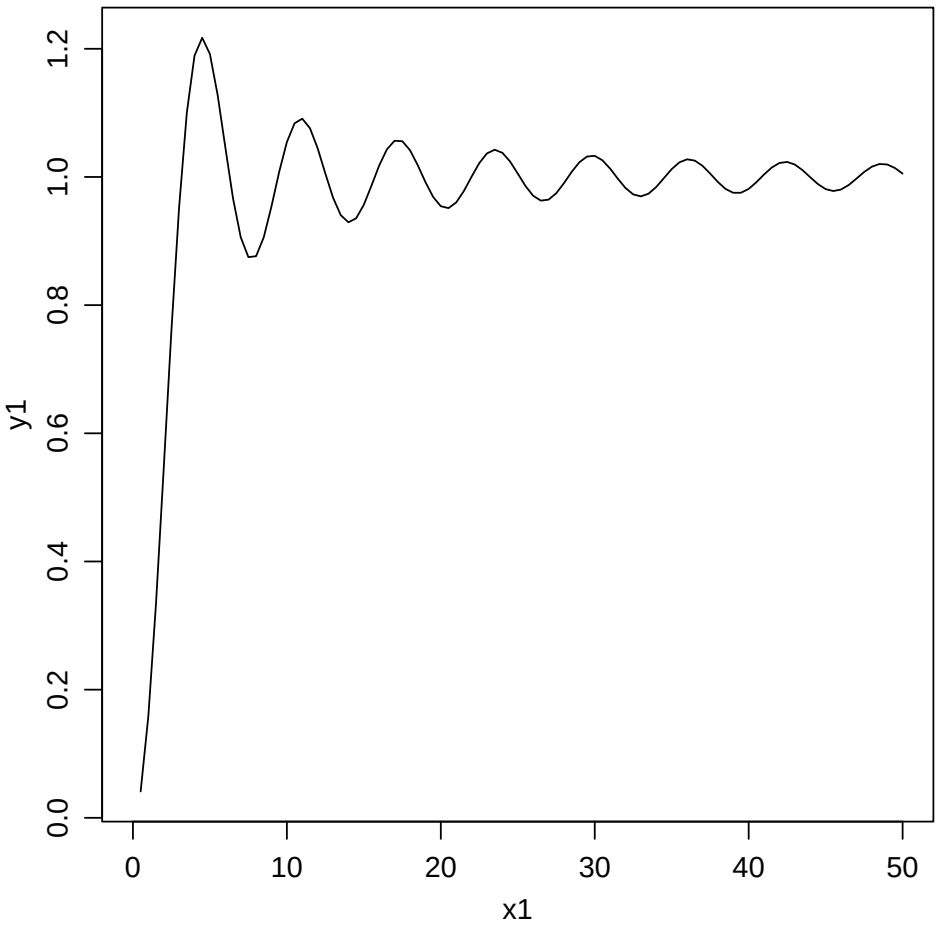



Figura 1: Gráfico da função dada em (a).

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)
```

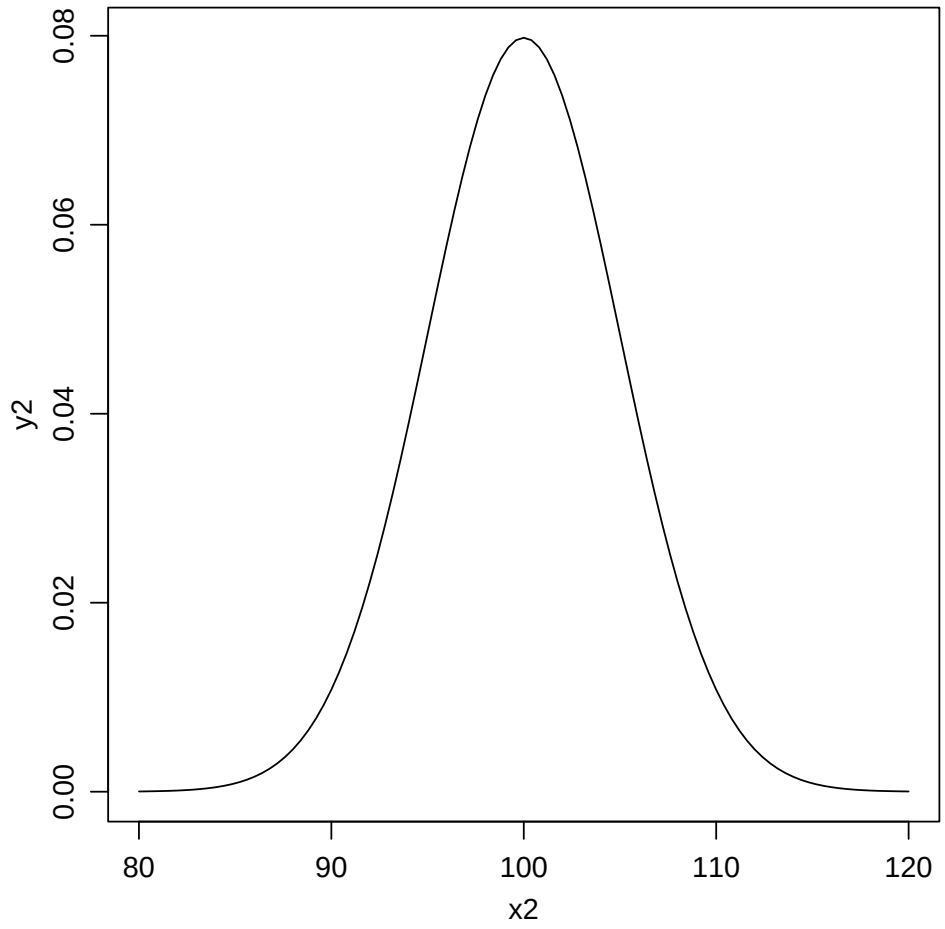



Figura 2: Gráfico da função dada em (b).

## 7.3 Integração numérica

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

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

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 7.3. 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")
```

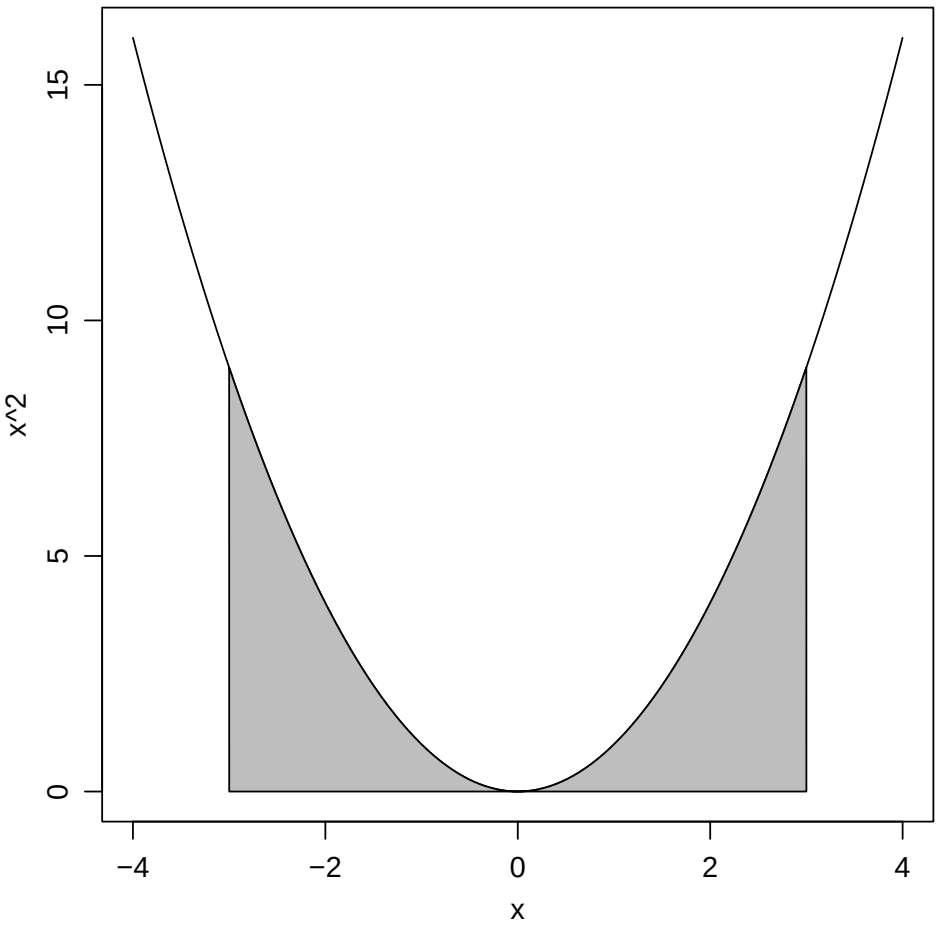



Figura 3: Gráfico onde a área indicada corresponde à integral definida na equação Equation 2.

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
```

## 7.4 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 $y$, aonde $x$ 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 + ...

# 8 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.

## 8.1 Entrando com dados diretamento no *R*

### 8.1.1 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
```

### 8.1.2 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
```

### 8.1.3 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.

## 8.2 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")
```

## 8.3 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*.

## 8.4 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")
```

## 8.5 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.

# 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 2: 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 3: 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)[civil.tb == max(civil.tb)]
> civil.mo

[1] "casado"
```

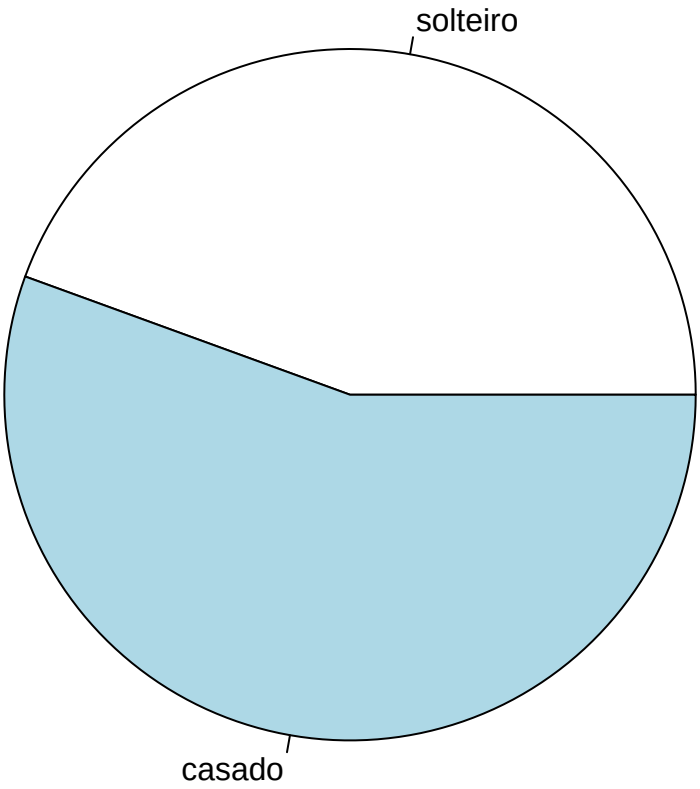



Figura 4: Gráfico de setores para variável `civil`.

**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

> 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)
```

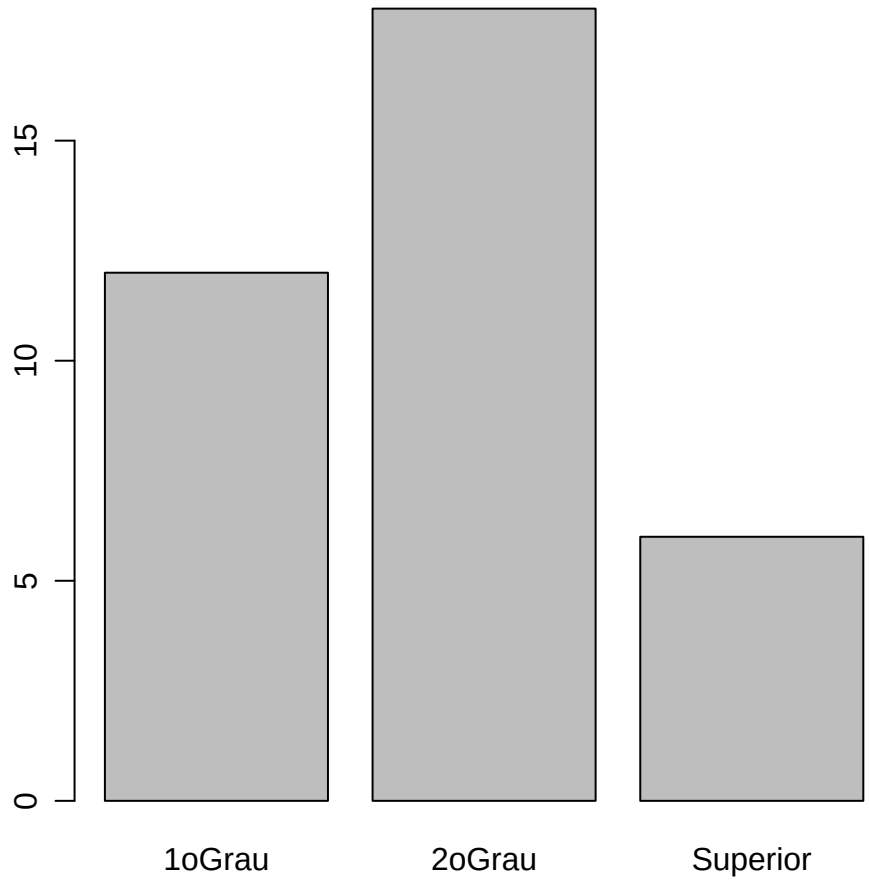



Figura 5: Gráfico de barras para variável `instrucao`.

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)[instrucao.tb == max(instrucao.tb)]
> instrucao.mo

[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)
```

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

   0    1    2    3    5
0.20 0.45 0.80 0.95 1.00

> 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 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.

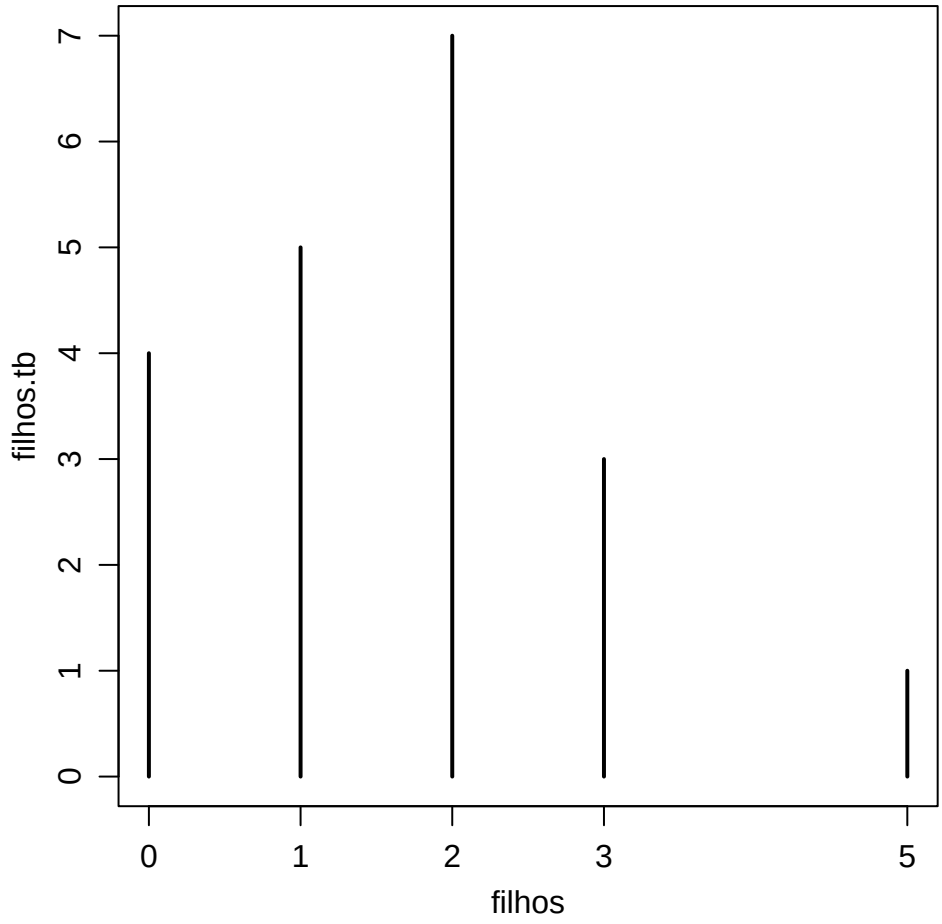



Figura 6: Gráfico de frequências absolutas para variável `filhos`.

```
> filhos.mo <- names(filhos.tb)[filhos.tb == 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)
```

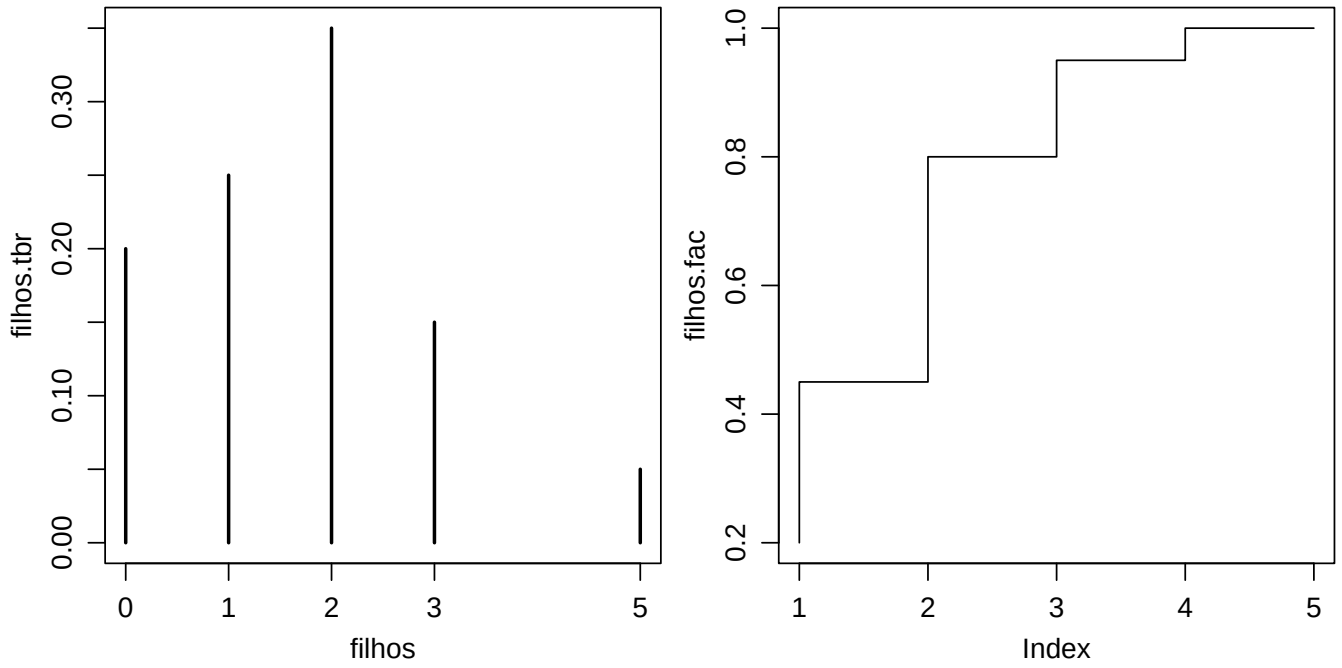



Figura 7: Gráfico de frequências relativas (esquerda) e frequências acumuladas para variável `filhos`.

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)

function (x, ...)
NULL

> args(cut.default)
```

Figura 8: Histograma (esquerda) e boxplot (direita) para a variável `salario`.

```
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)
```

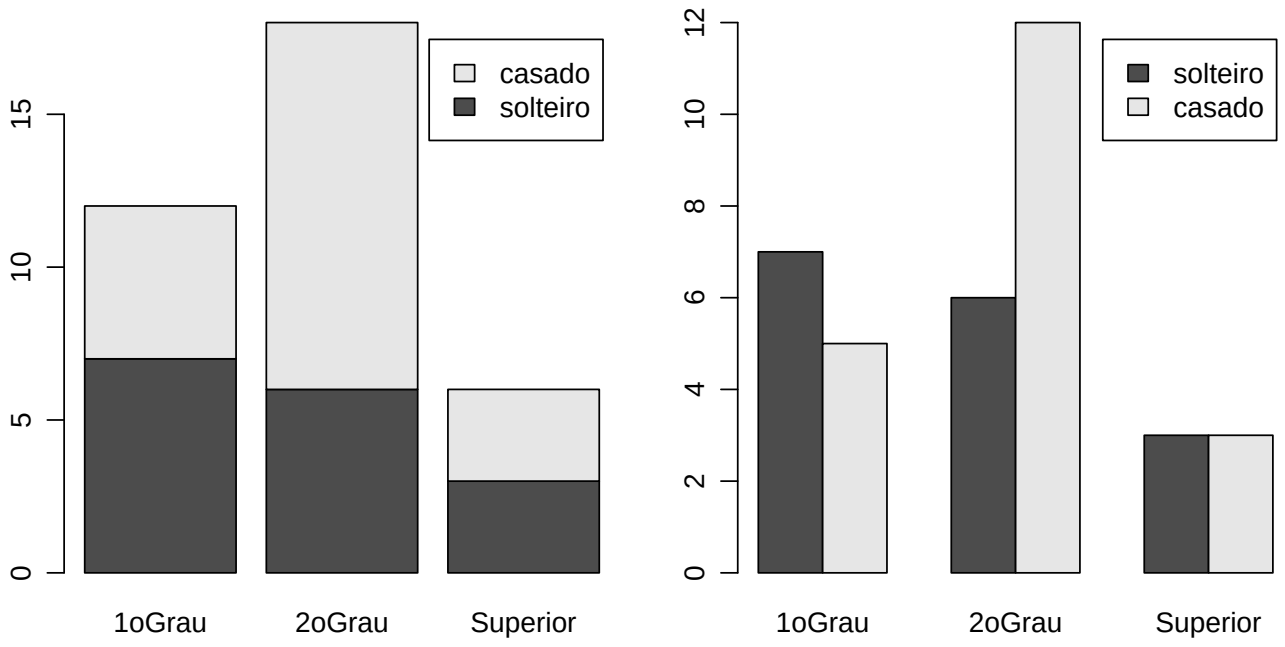



Figura 9: Dois tipos de gráficos de barras ilustrando o cruzamento das variáveis `civil` e `instrucao`.

```
          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)
```

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 2º 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

          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)
```

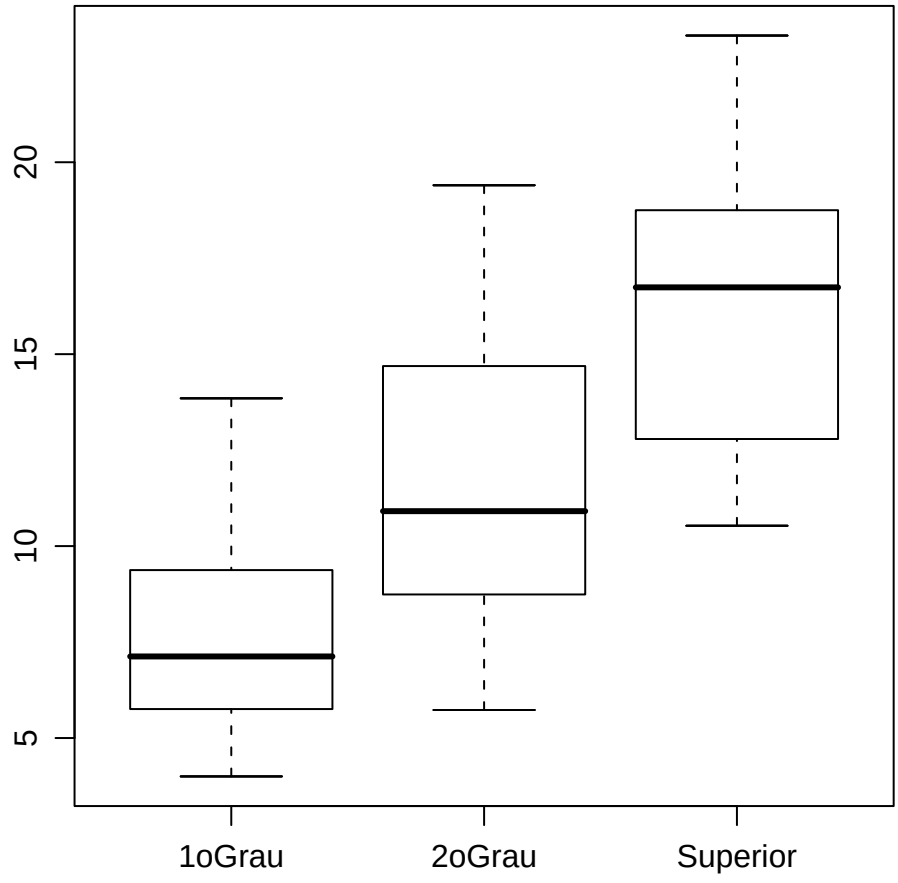



Figura 10: Boxplot da variável `salario` para cada nível da variável `instrucao`.

```
  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 $4 \times 4$.

```
> 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 $3 \times 3$.

```
> 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)
```

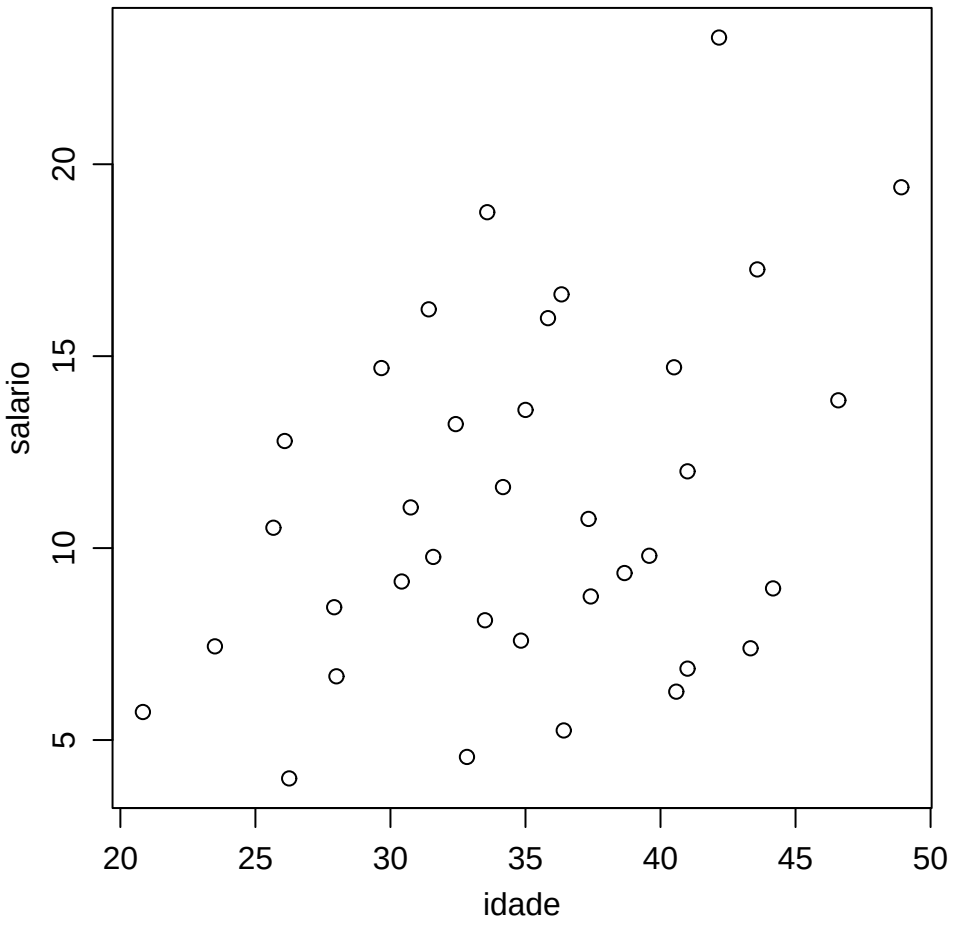



Figura 11: Diagrama de dispersão para as variáveis `salario` e `idade`.

## 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)
```

## 9.5 Mais detalhes sobre o uso de funções

As funções do R são documentadas e o uso é explicado e ilustrado usando a `help()`. Por exemplo, o comando `help(mean)` vai exibir e documentação da função `mean()`. Note que no final da documentação há exemplos de uso da função que voce pode reproduzir para entendê-la melhor.

## 9.6 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.

# 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)
```

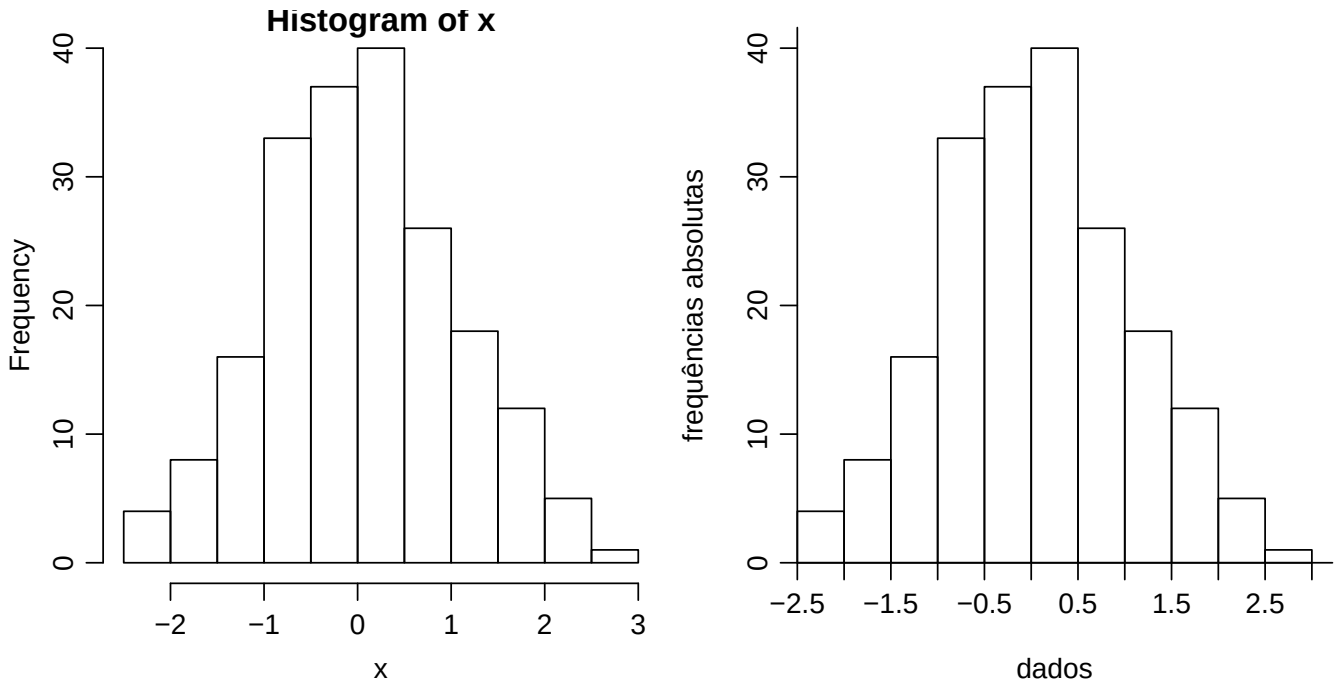



Figura 12: Histograma gerado com opções padrão (esquerda) e modificadas (direita).

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 Análise descritiva de tabelas de contingência

## 11.1 Tabelas para dois ou mais fatores

Vamos utilizar aqui os dados *milsa* de Bussab & Morettin discutidos em **??** e que podem ser obtidos conforme comando abaixo. Repetimos aqui o preparo inicial dos dados convertendo as variáveis categóricas em *fatores* do R e criando a variável *idade*.

```
> milsa <- read.table("http://www.leg.ufpr.br/~paulojus/aulasR/dados/milsa.dat",
+     head = T)
> 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)))
> milsa <- transform(milsa, idade = ano + mes/12)
> names(milsa)
```

Tabelas de contingência podem ser obtidas com as frequências de occorrência dos cruzamentos das variáveis. A seguir mostramos algumas opções da vizualização dos resultados usando a função `table()` e a função `ftable()`. As funções retornam as tabelas de contingência em um objeto que pode ser uma *matrix*, no caso do cruzamento de duas variáveis, ou de forma mais geral, na forma de um *array*, onde o número de dimensões é igual ao número de variáveis. Entretanto a classe do objeto resultante vai depender da função utilizada. Neste caso, para o cruzamento de apenas duas variáveis, os resultados são exibidos de forma semelhante. No exemplo consideram-se as variáveis *civil* e *instrucao* que situadas nas colunas 2 e 3 do *data-frame*.

```
> t1 <- table(milsa[c(2, 3)])
> t1

          instrucao
civil      1oGrau 2oGrau Superior
  solteiro      7      6        3
  casado        5     12        3

> t1f <- ftable(milsa[c(2, 3)])
> t1f

         instrucao 1oGrau 2oGrau Superior
civil
solteiro               7      6        3
casado                 5     12        3

> sapply(list(t1, t1f), class)

[1] "table"  "ftable"

> sapply(list(t1, t1f), is.matrix)

[1] TRUE TRUE

> sapply(list(t1, t1f), is.array)
```

```
[1] TRUE TRUE
```

Ambas funções possuem o argumento `dnn` que pode ser usado para sobrescrever os nomes das dimensões do objeto resultante.

```
> dimnames(t1)

$civil
[1] "solteiro" "casado"

$instrucao
[1] "1oGrau"   "2oGrau"   "Superior"

> t1 <- table(milsa[c(2, 3)], dnn = c("Estado Civil", "Nível de Instrução"))
> dimnames(t1)

$`Estado Civil`
[1] "solteiro" "casado"

$`Nível de Instrução`
[1] "1oGrau"   "2oGrau"   "Superior"

> t1f <- table(milsa[c(2, 3)], dnn = c("Estado Civil", "Nível de Instrução"))
```

As diferenças na forma de exibir os resultados são mais claras considerando-se o cruzamento de três ou mais variáveis. Enquanto `table()` vai exibir um *array* da forma usual, mostrando as várias camadas separadamente, `ftable()` irá arranjar a tabela de forma plana, em uma visualização mais adequada para a leitura dos dados. Vamos considerar o cruzamento das variáveis *civil*, *instrucao* e *regiao* situadas nas colunas 2, 3 e 8 do *data-frame*.

```
> t2 <- with(milsa, table(civil, instrucao, regiao))
> t2

, , regiao = capital

          instrucao
civil      1oGrau 2oGrau Superior
  solteiro      2      1        1
  casado        2      4        1

, , regiao = interior

          instrucao
civil      1oGrau 2oGrau Superior
  solteiro      2      1        1
  casado        1      6        1

, , regiao = outro

          instrucao
civil      1oGrau 2oGrau Superior
  solteiro      3      4        1
  casado        2      2        1
```

```
> t2f <- with(milsa, ftable(civil, instrucao, regiao))
> t2f

                  regiao capital interior outro
civil    instrucao
solteiro 1oGrau                2        2     3
         2oGrau                1        1     4
         Superior              1        1     1
casado   1oGrau                2        1     2
         2oGrau                4        6     2
         Superior              1        1     1
```

Enquanto que o objeto retornado por `table()` não é uma matrix, mas sim um *array* de três dimensões, por serem três variáveis. A dimensão do *array* é de $2 \times 3 \times 3$ por haver 2 estados civis, 3 níveis de instrução e 3 regiões. Já o objeto retornado por `ftable()` ainda é uma matriz, neste caso de dimensão $6 \times 3$ onde $6 = 2 \times 3$ indicando o produto do número de nívies das duas primeiras variáveis.

```
> sapply(list(t2, t2f), is.matrix)

[1] FALSE  TRUE

> sapply(list(t2, t2f), is.array)

[1] TRUE TRUE

> sapply(list(t2, t2f), dim)

[[1]]
[1] 2 3 3

[[2]]
[1] 6 3
```

Com `ftable()` é possível ainda criar outras visualizações da tabela. Os argumentos `row.vars` e `col.vars` podem ser usados para indicar quais variáveis serão colocadas nas linhas e colunas, e em que ordem. No exemplo a seguir colocamos o estado civil e região de procedência (variáveis 1 e 3) nas colunas da tabela e também modificamos o nome das dimensões da tabela com o argumento `dnn`. O objeto resultante é uma matrix de dimensão $6 \times 3$.

```
> with(milsa, ftable(civil, instrucao, regiao, dnn = c("Estado Civil:",
+     "Nível de Instrução", "Procedência:"), col.vars = c(1, 3)))

                  Estado Civil: solteiro                casado
                  Procedência:   capital interior outro capital interior outro
Nível de Instrução
1oGrau                                2        2     3       2        1     2
2oGrau                                1        1     4       4        6     2
Superior                              1        1     1       1        1     1
```

## 11.2 Extensões: frequências relativas e gráficos

As funções `table()` e `ftable()` retornam objetos das classes `table` e `ftable`, respectivamente. A partir de tais objetos, outras funções podem ser utilizadas tais como `prop.table()` para obtenção de frequências relativas, ou `barplot()` para gráficos de barras. A distinção entre as classes não é importante no caso de cruzamento entre duas variáveis. Entretanto para três ou mais variáveis os resultados são bem diferentes, devido ao fato já mencionado de que `table()` retorna um array de dimensão igual ao número de variáveis, enquanto que `ftable()` retorna sempre uma matriz.

Considerando os exemplos da Seção anterior, vejamos primeiro os resultados de frequências relativas para duas variáveis, que não diferem entre as clases. Da mesma forma, no caso de duas variáveis, as *margens* da tabelas obtidas de uma ou outra forma são as mesmas.

```
> prop.table(t1)

            Nível de Instrução
Estado Civil     1oGrau     2oGrau   Superior
    solteiro 0.19444444 0.16666667 0.08333333
    casado   0.13888889 0.33333333 0.08333333

> prop.table(t1f)

            Nível de Instrução
Estado Civil     1oGrau     2oGrau   Superior
    solteiro 0.19444444 0.16666667 0.08333333
    casado   0.13888889 0.33333333 0.08333333

> prop.table(t1, margin = 1)

            Nível de Instrução
Estado Civil 1oGrau 2oGrau Superior
    solteiro 0.4375 0.3750   0.1875
    casado   0.2500 0.6000   0.1500

> prop.table(t1f, margin = 1)

            Nível de Instrução
Estado Civil 1oGrau 2oGrau Superior
    solteiro 0.4375 0.3750   0.1875
    casado   0.2500 0.6000   0.1500

> margin.table(t1, mar = 1)

Estado Civil
solteiro   casado
      16       20

> margin.table(t1f, mar = 1)

Estado Civil
solteiro   casado
      16       20

> margin.table(t1, mar = 2)
```

```
Nível de Instrução
  1oGrau   2oGrau Superior
      12       18        6

> margin.table(t1f, mar = 2)

Nível de Instrução
  1oGrau   2oGrau Superior
      12       18        6
```

Da mesma forma os gráficos obtidos são os mesmos. A Figura 11.2 mostra dois tipos de gráficos. Acima os gráficos mostram retângulos cojas áreas são proporcionais às frequências e abaixo um possível gráfico de barras.

```
> plot(t1, main = "")
> plot(t1f, main = "")
> barplot(t1, beside = T, legend = T)
> barplot(t1f, beside = T, legend = T)
```

Já para três os mais variáveis os resultados são bem diferentes em particular para as frequências marginais, uma vez que `ftable()` vai sempre retornar uma matriz e portanto só possuirá margens 1 e 2.

```
> prop.table(t2)

, , regiao = capital

          instrucao
civil          1oGrau     2oGrau   Superior
  solteiro 0.05555556 0.02777778 0.02777778
  casado   0.05555556 0.11111111 0.02777778

, , regiao = interior

          instrucao
civil          1oGrau     2oGrau   Superior
  solteiro 0.05555556 0.02777778 0.02777778
  casado   0.02777778 0.16666667 0.02777778

, , regiao = outro

          instrucao
civil          1oGrau     2oGrau   Superior
  solteiro 0.08333333 0.11111111 0.02777778
  casado   0.05555556 0.05555556 0.02777778

> prop.table(t2f)

                   regiao    capital   interior      outro
civil    instrucao
solteiro 1oGrau           0.05555556 0.05555556 0.08333333
         2oGrau           0.02777778 0.02777778 0.11111111
```

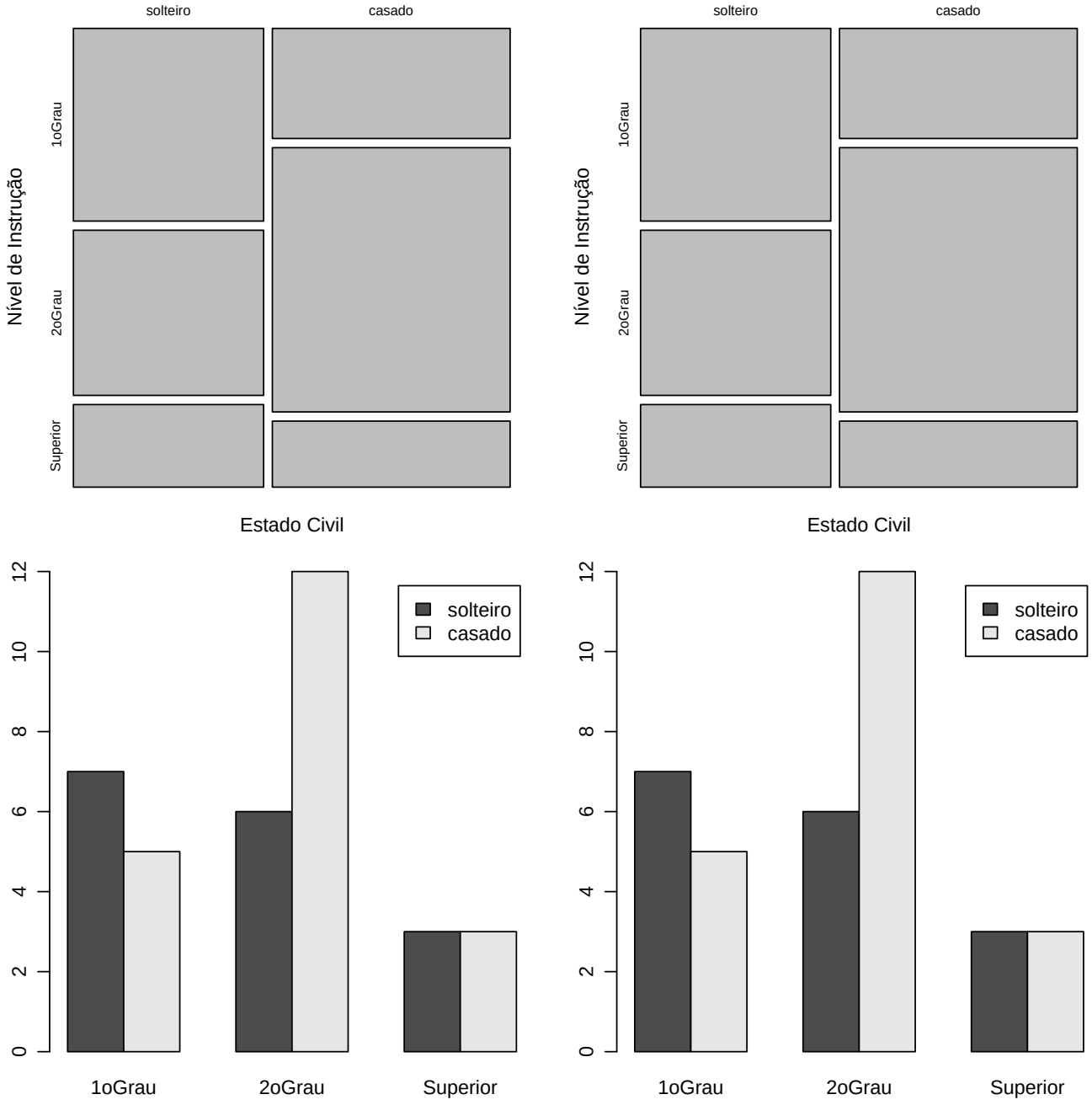



Figura 13: Representações gráficas de tabelas de contingência de duas variáveis obtidas pelas funções `table()` e `ftable()`.

```
         Superior         0.02777778 0.02777778 0.02777778
casado   1oGrau           0.05555556 0.02777778 0.05555556
         2oGrau           0.11111111 0.16666667 0.05555556
         Superior         0.02777778 0.02777778 0.02777778
```

```
> prop.table(t2, margin = 1)

, , regiao = capital

          instrucao
civil      1oGrau 2oGrau Superior
  solteiro 0.1250 0.0625   0.0625
  casado   0.1000 0.2000   0.0500

, , regiao = interior

          instrucao
civil      1oGrau 2oGrau Superior
  solteiro 0.1250 0.0625   0.0625
  casado   0.0500 0.3000   0.0500

, , regiao = outro

          instrucao
civil      1oGrau 2oGrau Superior
  solteiro 0.1875 0.2500   0.0625
  casado   0.1000 0.1000   0.0500
```

```
> prop.table(t2f, margin = 1)

                   regiao   capital  interior     outro
civil    instrucao
solteiro 1oGrau           0.2857143 0.2857143 0.4285714
         2oGrau           0.1666667 0.1666667 0.6666667
         Superior         0.3333333 0.3333333 0.3333333
casado   1oGrau           0.4000000 0.2000000 0.4000000
         2oGrau           0.3333333 0.5000000 0.1666667
         Superior         0.3333333 0.3333333 0.3333333
```

```
> prop.table(t2, margin = 3)

, , regiao = capital

          instrucao
civil          1oGrau     2oGrau   Superior
  solteiro 0.18181818 0.09090909 0.09090909
  casado   0.18181818 0.36363636 0.09090909

, , regiao = interior

          instrucao
```

```
civil          1oGrau     2oGrau   Superior
  solteiro 0.16666667 0.08333333 0.08333333
  casado   0.08333333 0.50000000 0.08333333

, , regiao = outro

          instrucao
civil          1oGrau     2oGrau   Superior
  solteiro 0.23076923 0.30769231 0.07692308
  casado   0.15384615 0.15384615 0.07692308
```

```
> prop.table(t2f, margin=3)

Error in sweep(x, margin, margin.table(x, margin), "/") :
        índice fora de limites
```

É possível obter totais marginais com `margin.table()` a partir de um objeto resultante de `table()` mas **não** para um objeto resultante de par`ftable()`!

```
> margin.table(t2, mar = 1)

civil
solteiro   casado
      16       20

> margin.table(t2, mar = 2)

instrucao
  1oGrau   2oGrau Superior
      12       18        6

> margin.table(t2, mar = 3)

regiao
 capital interior    outro
      11       12       13
```

Para gráficos nem todos os resultados são mais possíveis, `plot()` vai funcionar para a classe `table` mas o resultado é inapropriado para `ftable`. Já `barplot()` irá funcionar apenas para `ftable`, mas o resultado pode não ser satisfatório pois as barras irão mostrar as combinações de duas variáveis.

```
> plot(t2, main = "")
> barplot(t2f, beside = T)
```

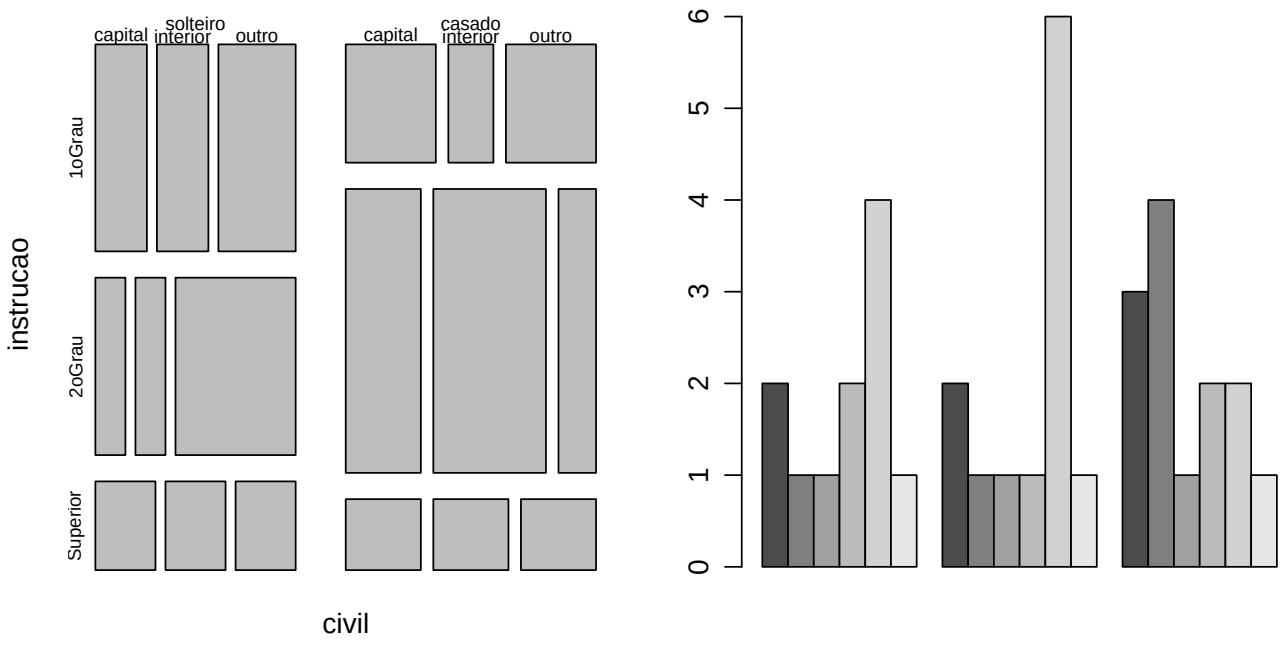



Figura 14: Representações gráficas de tabelas de contingência de três variáveis obtidas pelas funções `table()` (esquerda) e `ftable()` (direita).

# 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*. 4ª edição. Atual Editora. 1987.
Note que há uma edição mais nova: (5ª 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 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 15 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)
```

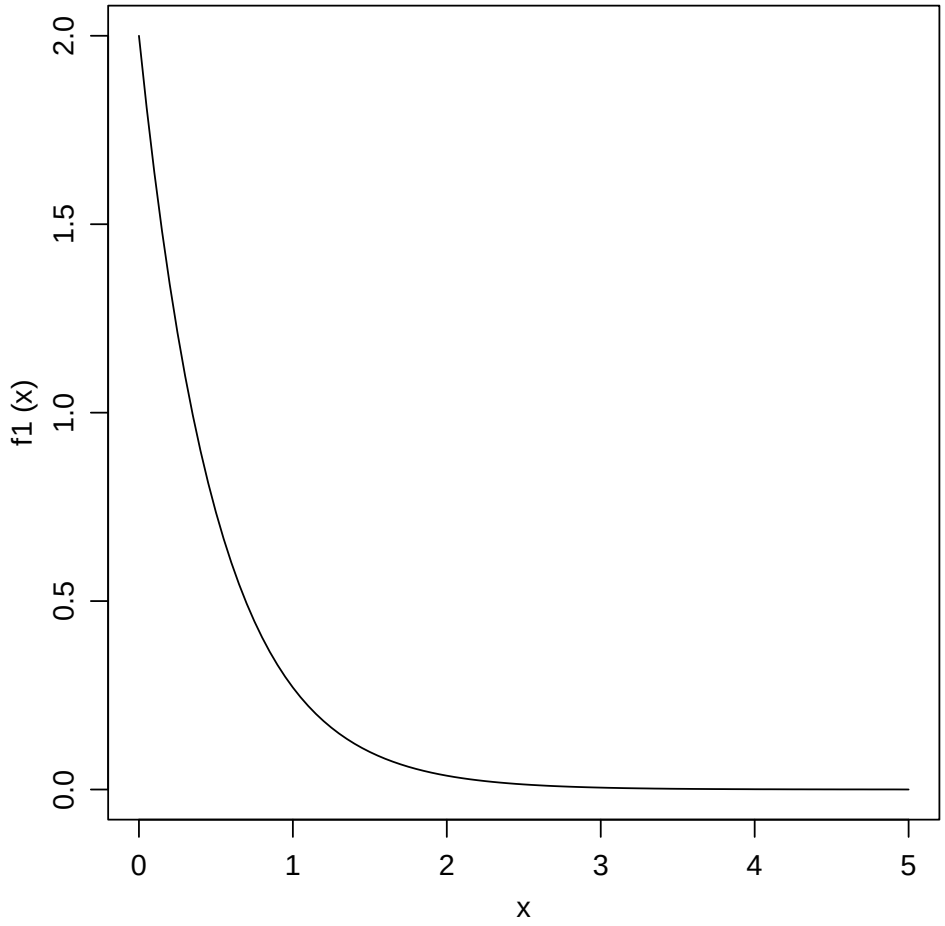



Figura 15: Gráfico da função de probabilidade do Exemplo 1.

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
```

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

$$\begin{aligned} p_b &= P(X > 1) = \int_1^{\infty} f(x)dx = \int_1^{\infty} 2\,e^{-2x}dx \\ p_c &= P(0,2 < X < 0,8) = \int_{0,2}^{0,8} f(x)dx = \int_{0.2}^{0.8} 2\,e^{-2x}dx \end{aligned}$$

cuja representação gráfica é mostrada na Figura 16. 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])))
```

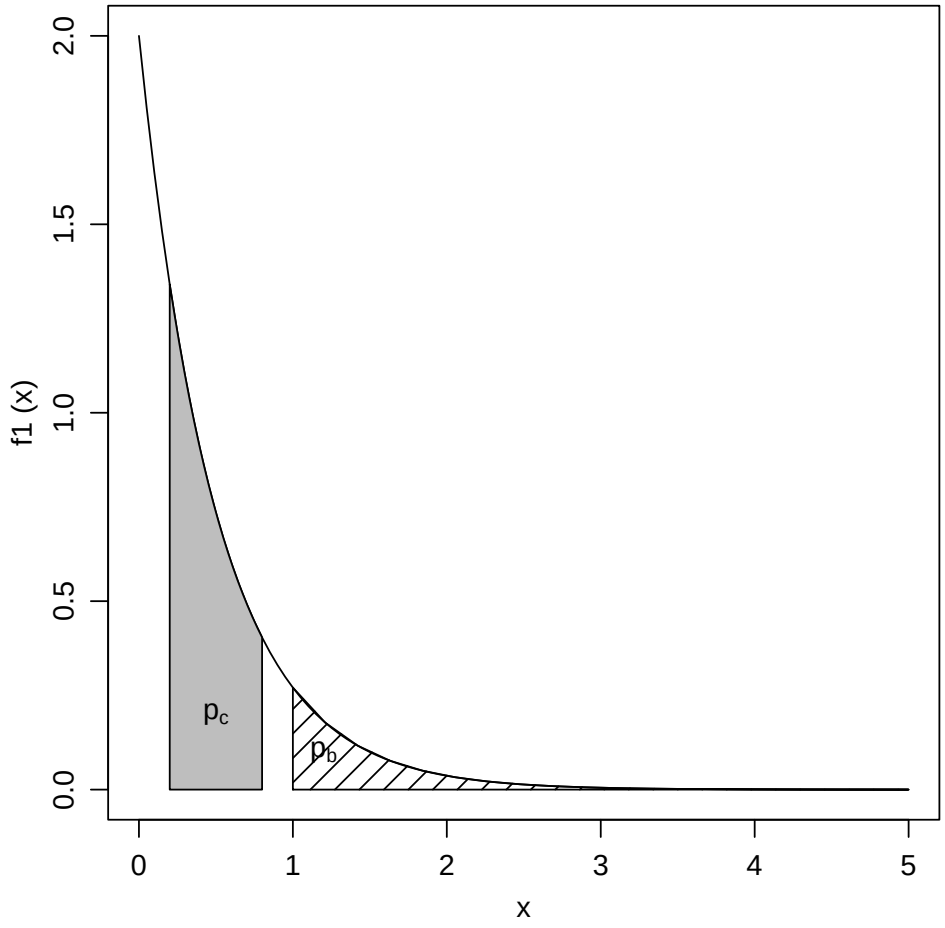



Figura 16: Probabilidades pedidas nos itens (b) e (c) do Exemplo 1.

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.

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

(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 $x$.

```
> 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 17.

```
> 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
```

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 with absolute error < 1.1e-15
```

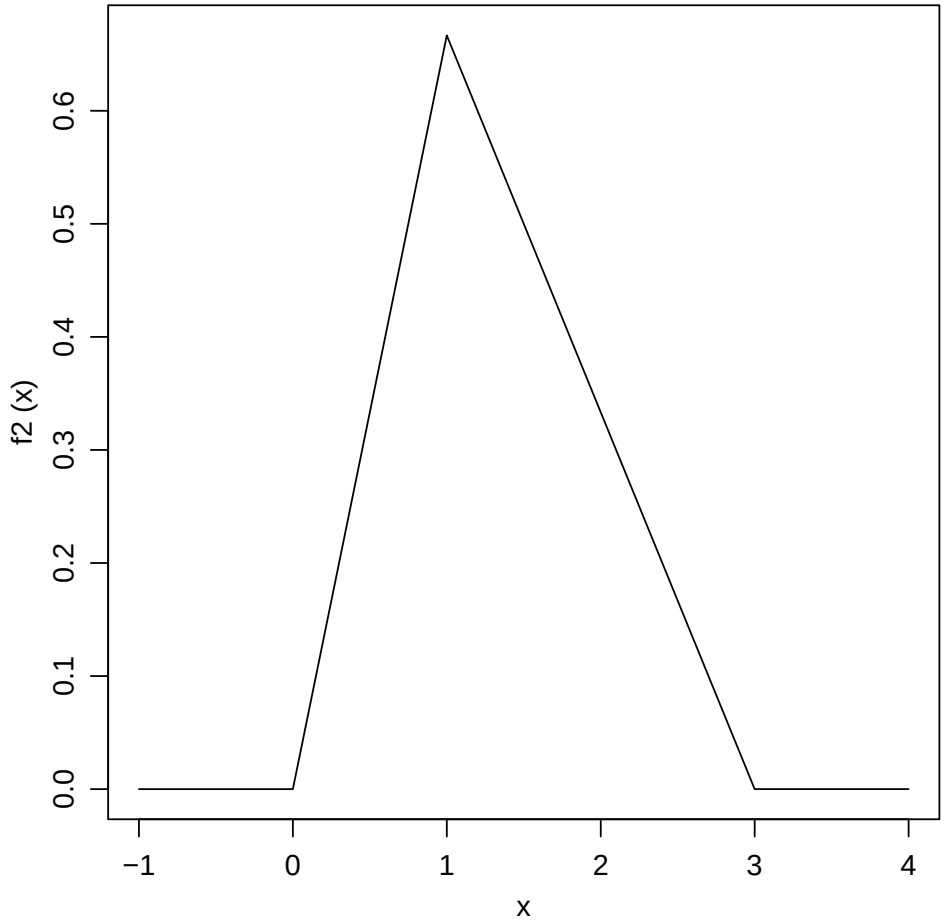



Figura 17: Gráfico da função densidade de probabilidade do Exemplo 2.

```
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 18 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)
```

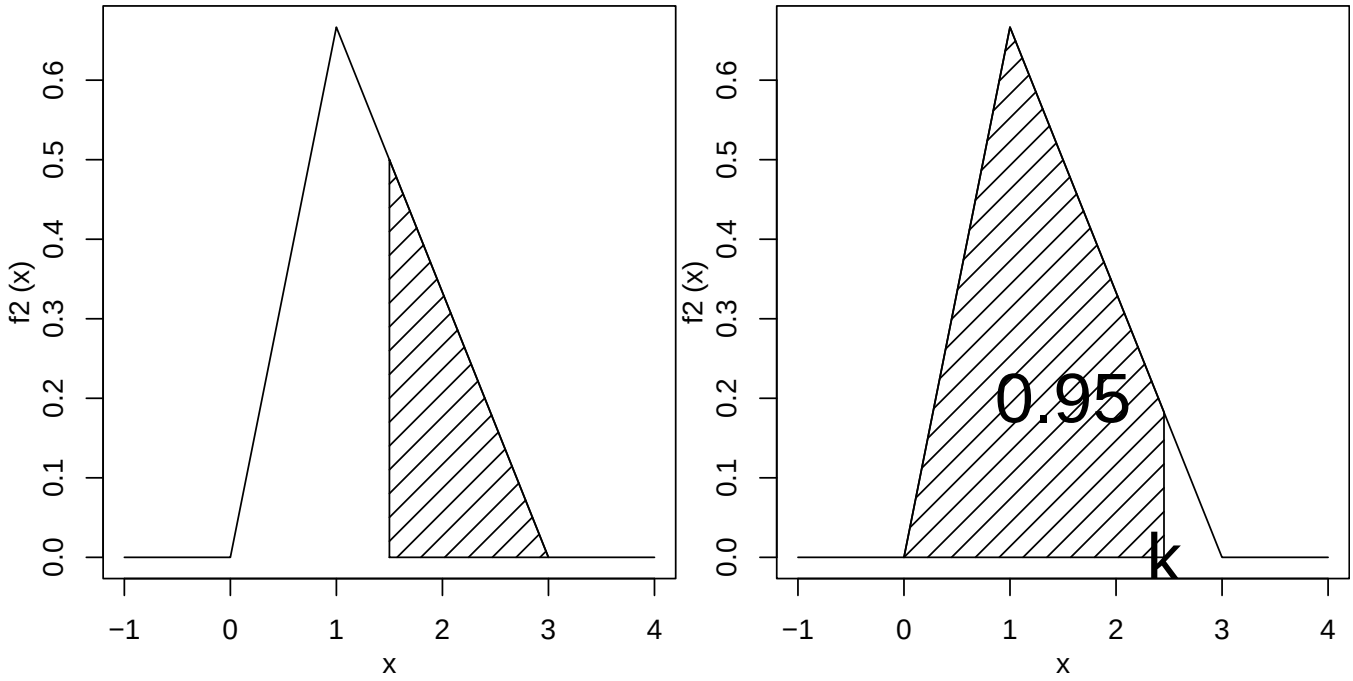



Figura 18: Gráficos indicando as soluções dos itens (a) e (c) do Exemplo 2.

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] -0.4791233  1.1333368 -0.4629392  1.0880655 -1.3480392  0.2646055 -0.6607288
 [8] -0.6265252 -0.4226680 -0.3349461
```

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
```

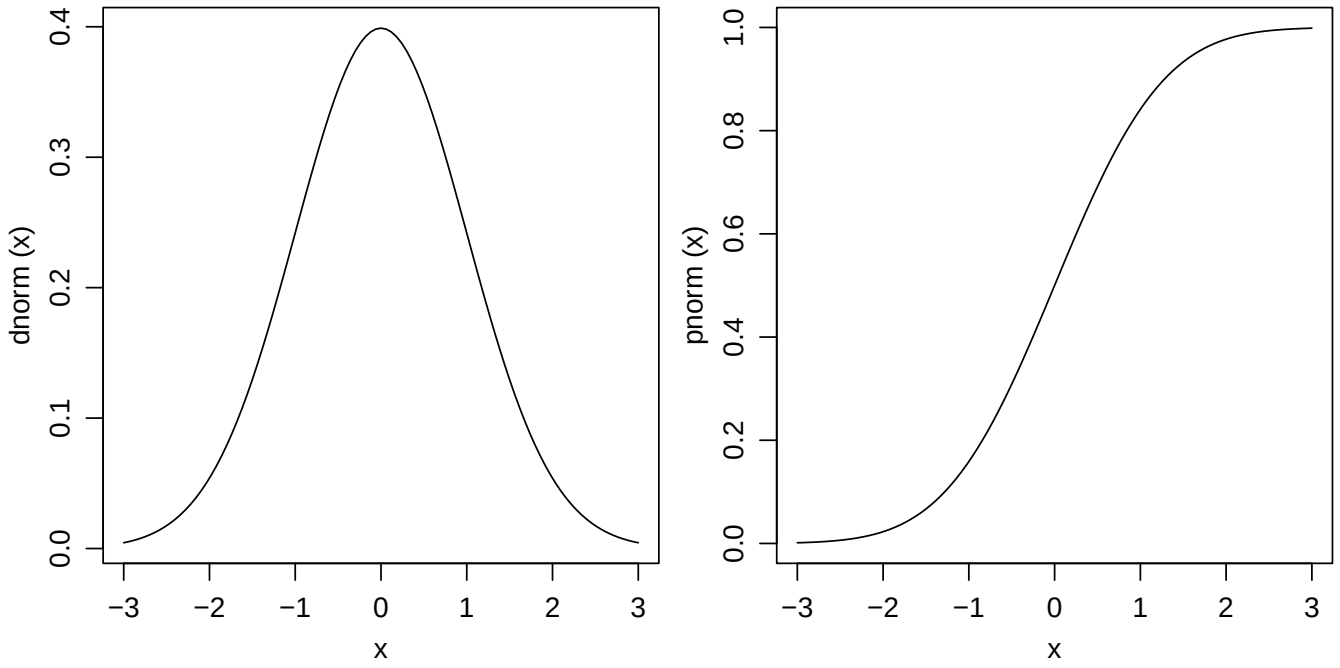



Figura 19: Funções de densidade e probabilidade da distribuição normal padrão.

```
> 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
```

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 19 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 20 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")
```

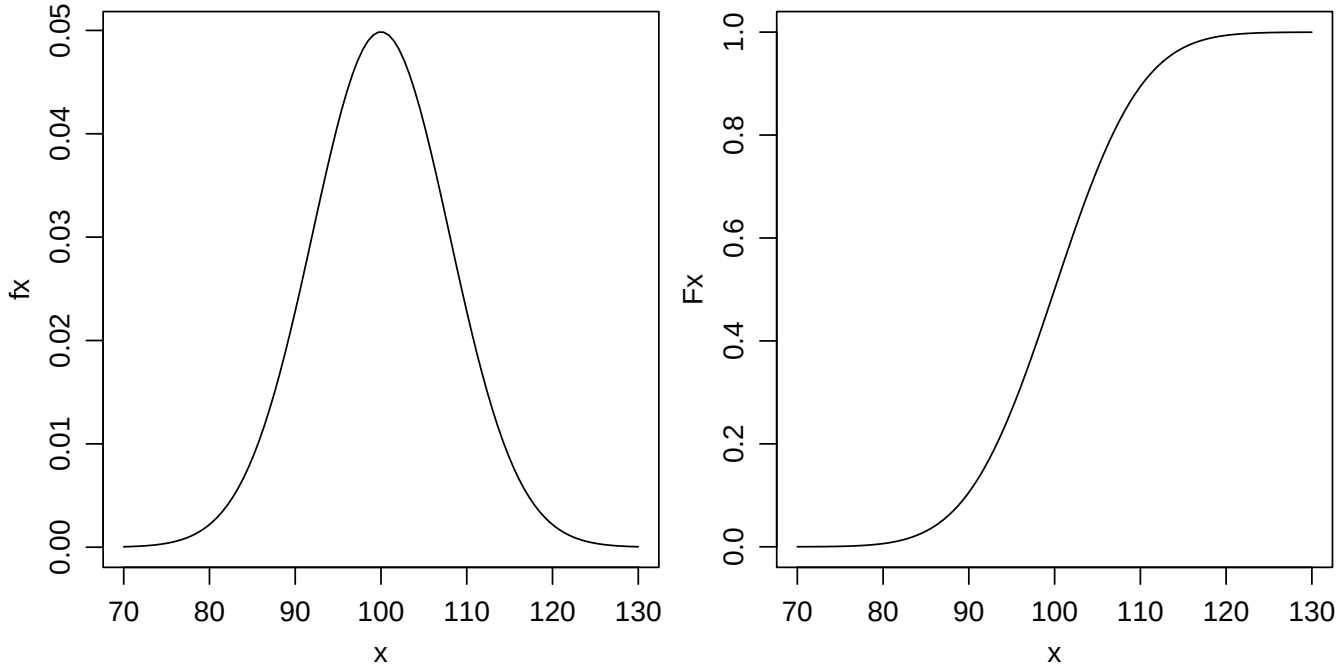



Figura 20: Funções de densidade de probabilidade (esquerda) e função de distribuição acumulada (direita) da $N(100, 64)$.

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 21 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)
```

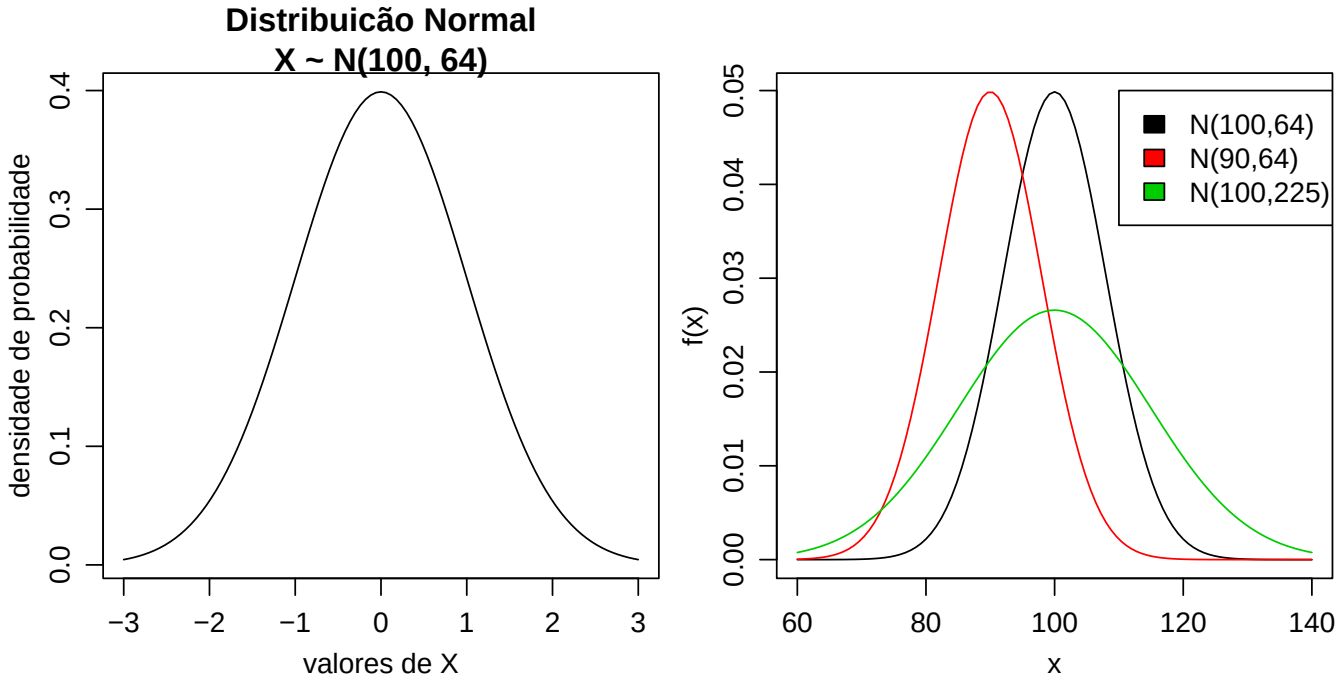



Figura 21: Gráfico com texto nos eixos e título (esquerda) e várias distribuições em um mesmo gráfico (direita).

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
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 22.

```
> 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)
```

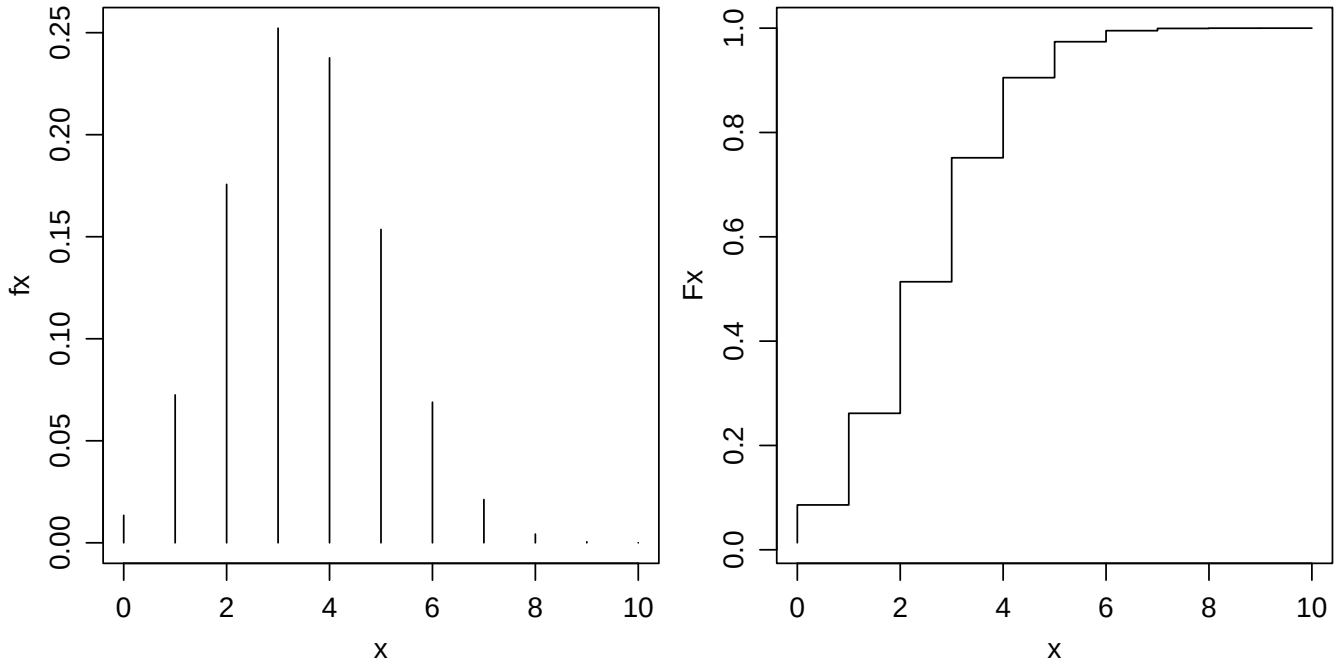



Figura 22: Funções de probabilidade (esquerda) e distribuição acumulada (direita) da $B(10, 0.35)$.

```
[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)

[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] 0.12165534 0.06732587 0.85313599 0.19932909 0.06736856

> 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]  5.172409  6.342121 11.629578  5.188892 14.846619
```

### 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]  8  8  6  4 10  1  8  9  4  7 10 10  9  8  2
```

## 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] 17 13  5
  ```

- 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] 18 17 13 12 19
```

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

```
> sample(1:6, 10, rep = T)

 [1] 5 4 5 2 3 1 4 1 4 6
```

- 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] 6 4 5 5 5 1 2 4 2 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 $\chi^2$ 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.

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.

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

# 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 23 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
```

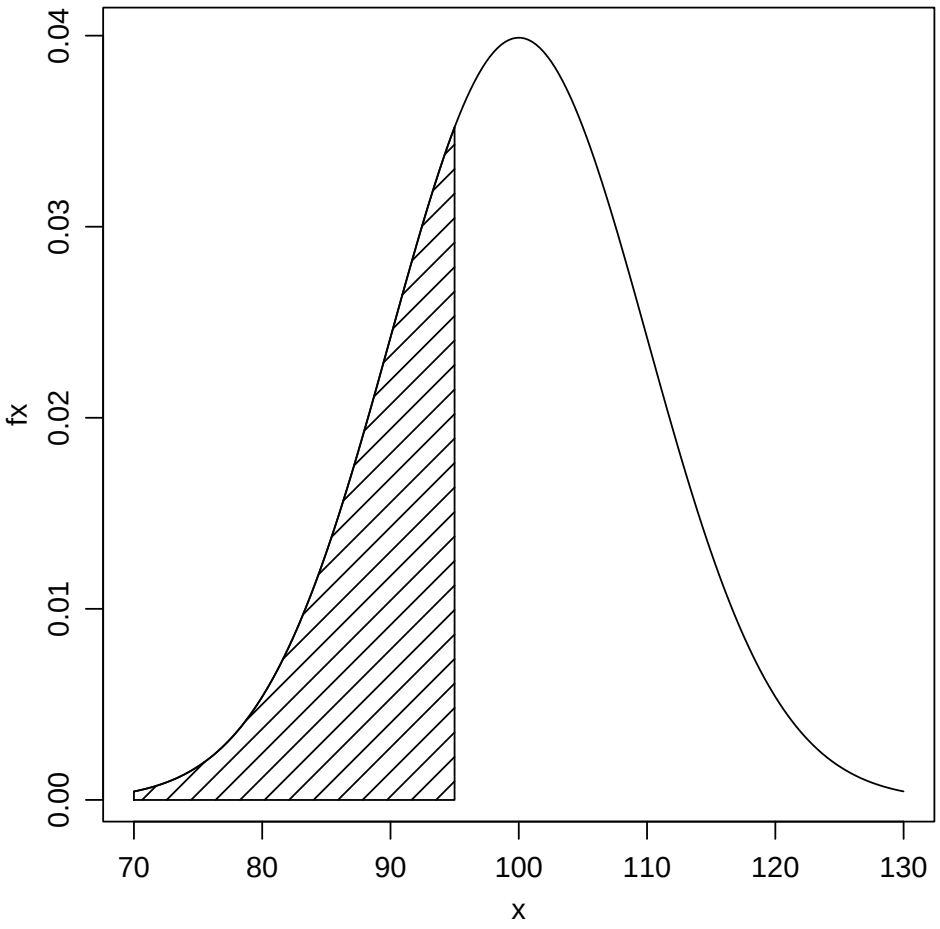



Figura 23: Funções de densidade da $N(100, 100)$ com a área correspondente à $P[X \leq 95]$.

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()`.

## 14.2 Distribuição exponencial

A função de densidade de probabilidade da distribuição exponencial com parâmetro $\lambda$ 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 24 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 $E[X] = \int_0^{\infty} xf(x)dx = \lambda$ 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
```

```
> 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)
```

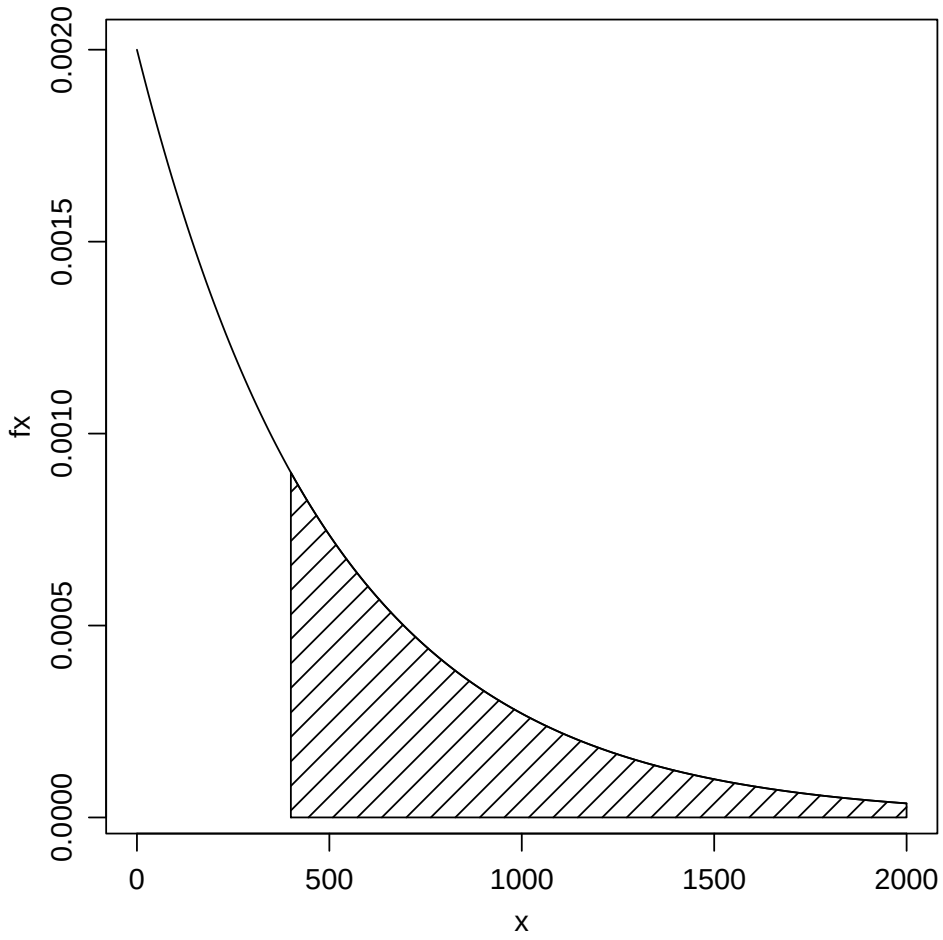



Figura 24: Função de densidade da $Exp(500)$ com a área correspondente à $P[X \geq 400]$.

```
[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()` 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 25 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)
```

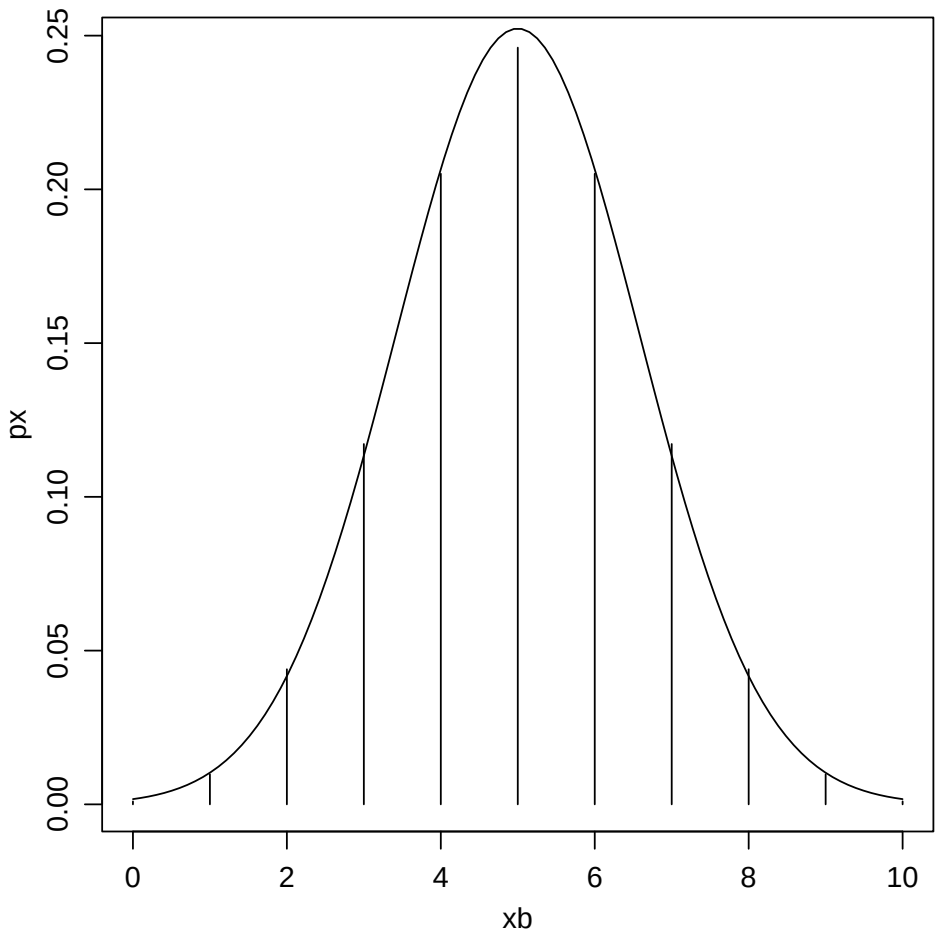



Figura 25: Função de probabilidade da $B(10, 1/2)$ e a aproximação pela $N(5, 2.5)$.

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)

[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:

   - $P[X > 2]$
   - $P[1.5 < X < 6]$
   - $P[X < 8]$

# 15 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.

## 15.1 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 26.

```
> 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 27 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.

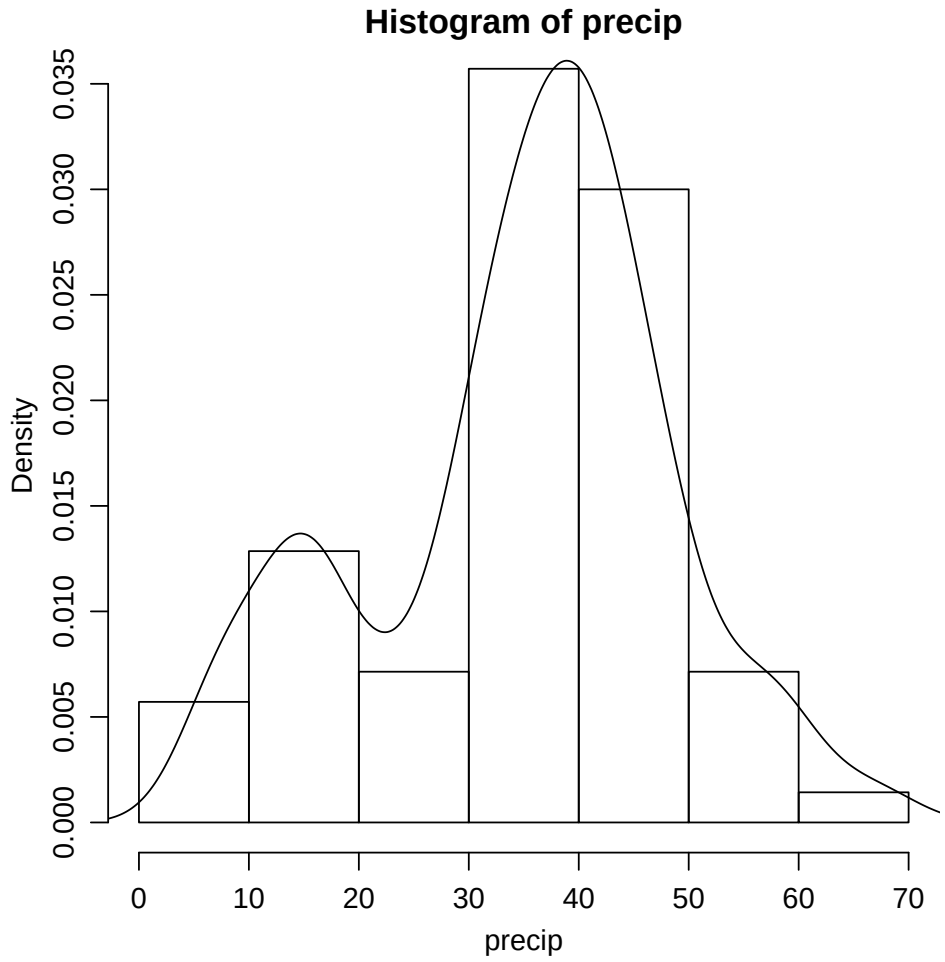



Figura 26: Histograma para os dados `precip` e a densidade estimada usando a função `density`.

```
> 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 28. 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 29.

```
> (kernels <- eval(formals(density.default)$kernel))
> plot(density(0, bw = 1), xlab = "", main = "kernels com bw = 1")
```

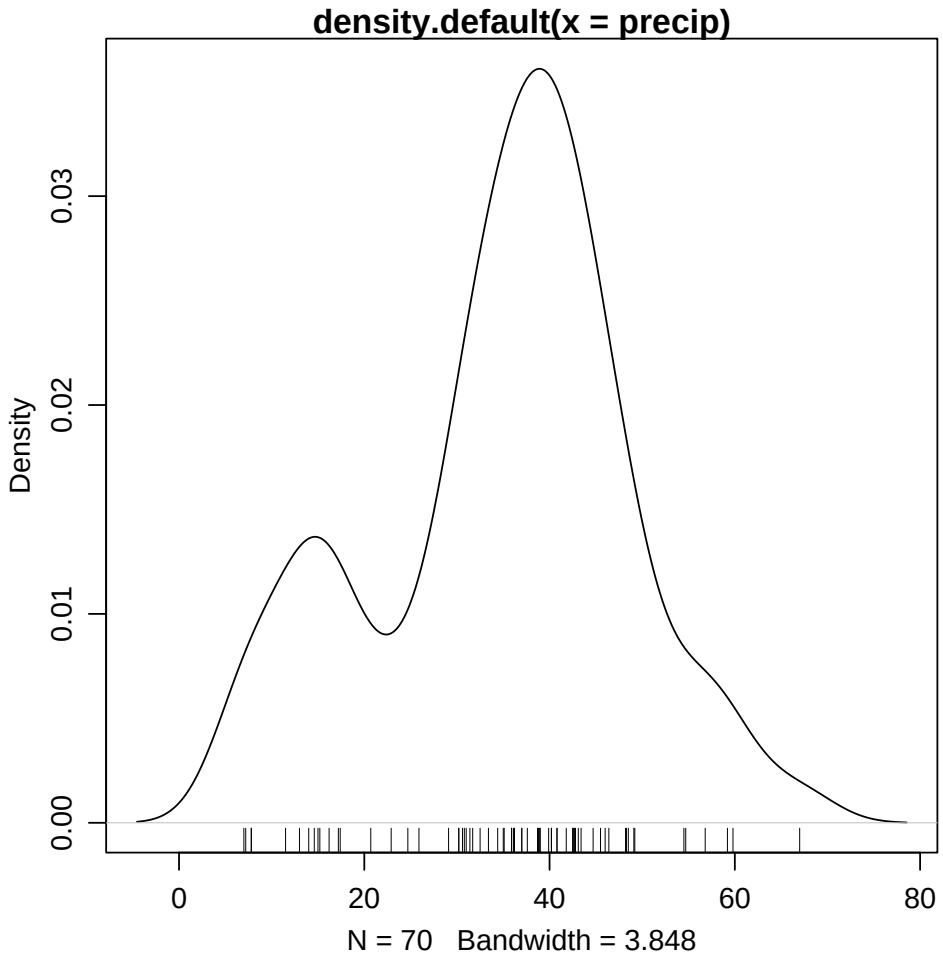



Figura 27: Dados `precip` e a densidade estimada usando a função `density`.

```
> for (i in 2:length(kernels)) lines(density(0, bw = 1, kern = kernels[i]),
+     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 30. 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.

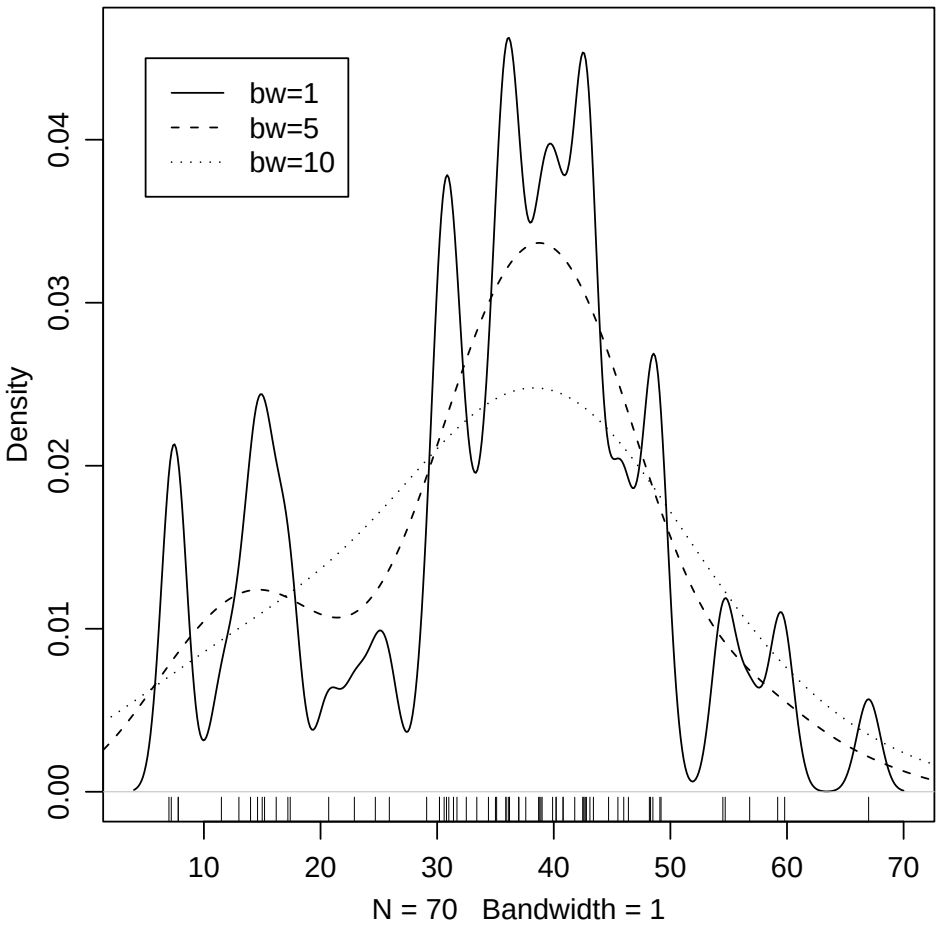



Figura 28: Densidade estimada usando a função `density` com diferentes valores para o argumento `bw`.

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 31.

```
> 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 32 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"
```

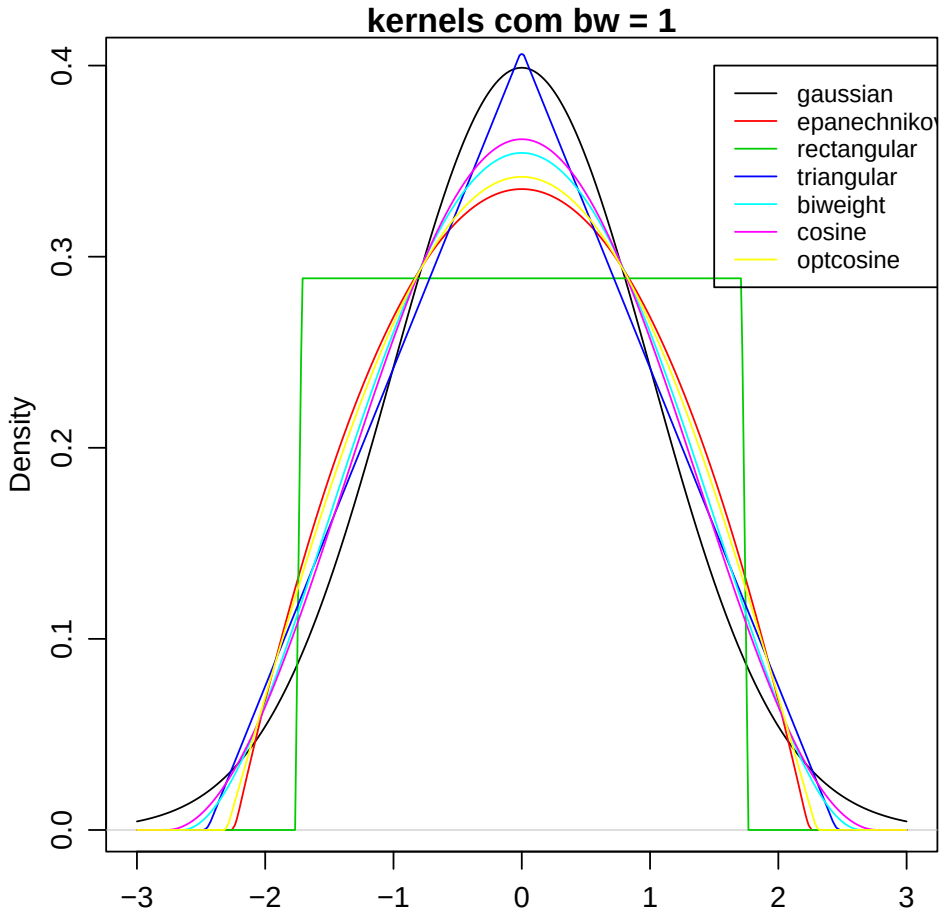



Figura 29: Diferentes núcleos implementados pela função `density`.

```
> 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)
```

## 15.2 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`.

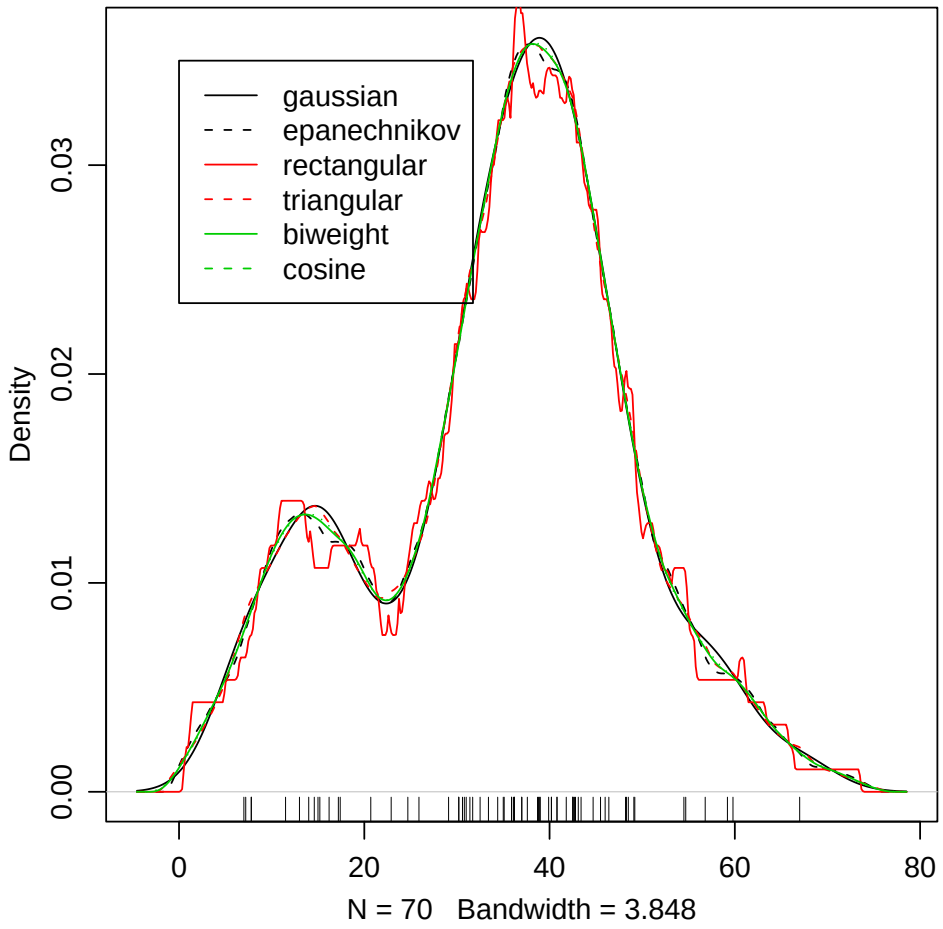



Figura 30: Densidade estimada usando a função `density` com diferentes valores para o argumento `kernel`.

# 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

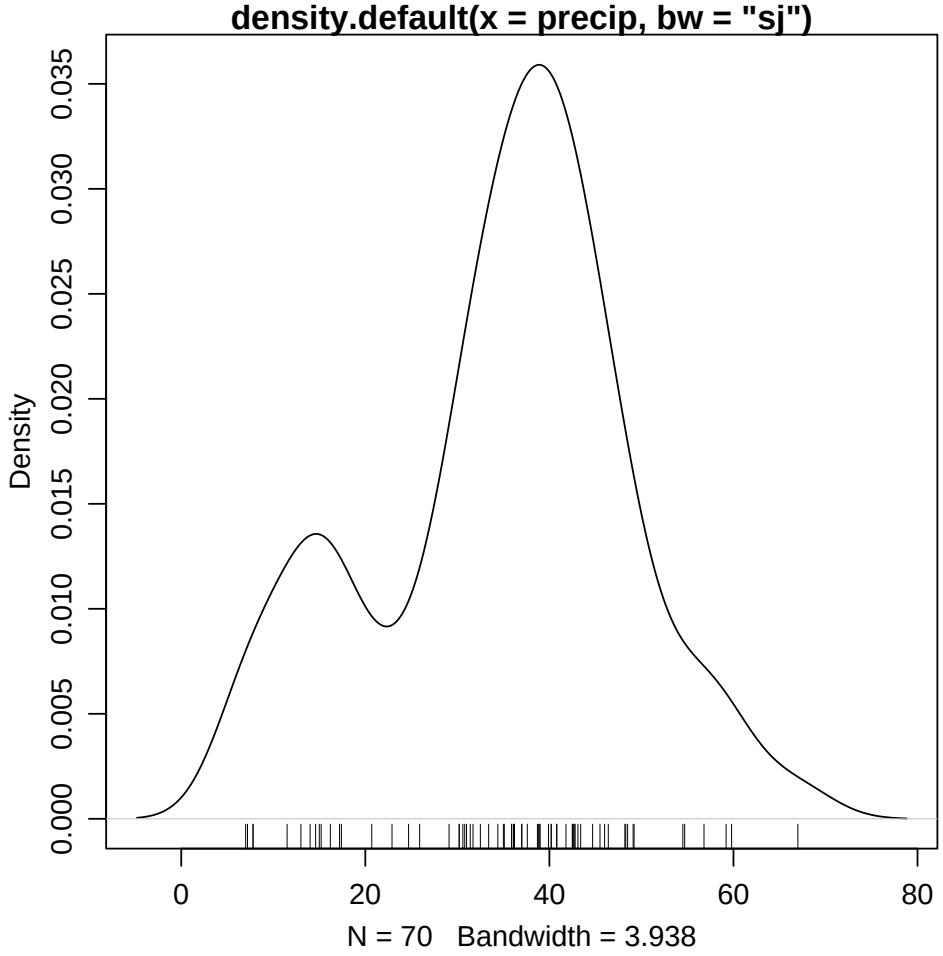



Figura 31: Densidade estimada para os dados `precip` usando a função `density` com critério de Sheather & Jones para seleção da largura de banda.

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.

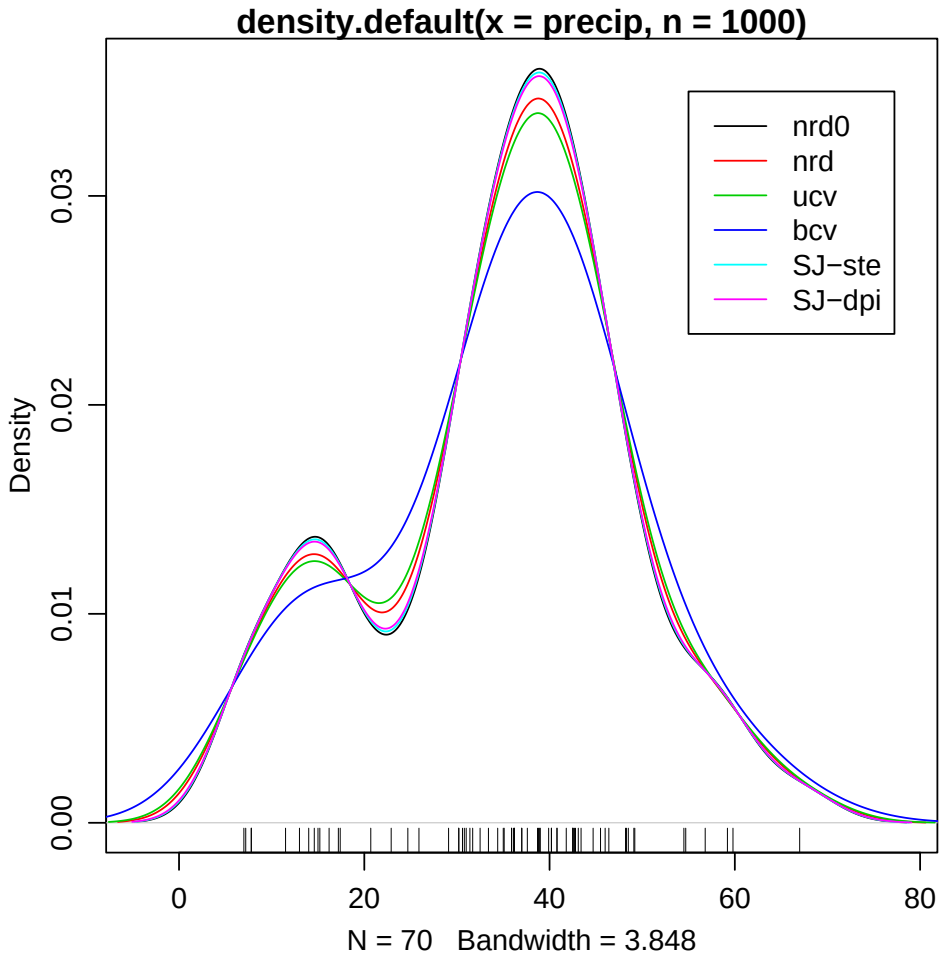



Figura 32: Diferentes métodos para largura de banda implementados pela função `density`.

### 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 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.

## 17.1 Definições e notações

Seja $L(\theta; y)$ a função de verossimilhança. A notação indica que o argumento da função é $\theta$ 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 $\theta$ 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 $\theta$. 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.

## 17.2 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 $\mu$ 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

$$\begin{aligned} l(\mu) &= \sum_1^{10} \log(f(x_i)) \\ &= -5\log(8\pi) - \frac{1}{8}(\sum_1^{10} x_i^2 - 2\mu \sum_1^{10} x_i + 10\mu^2), \end{aligned} \tag{4}$$

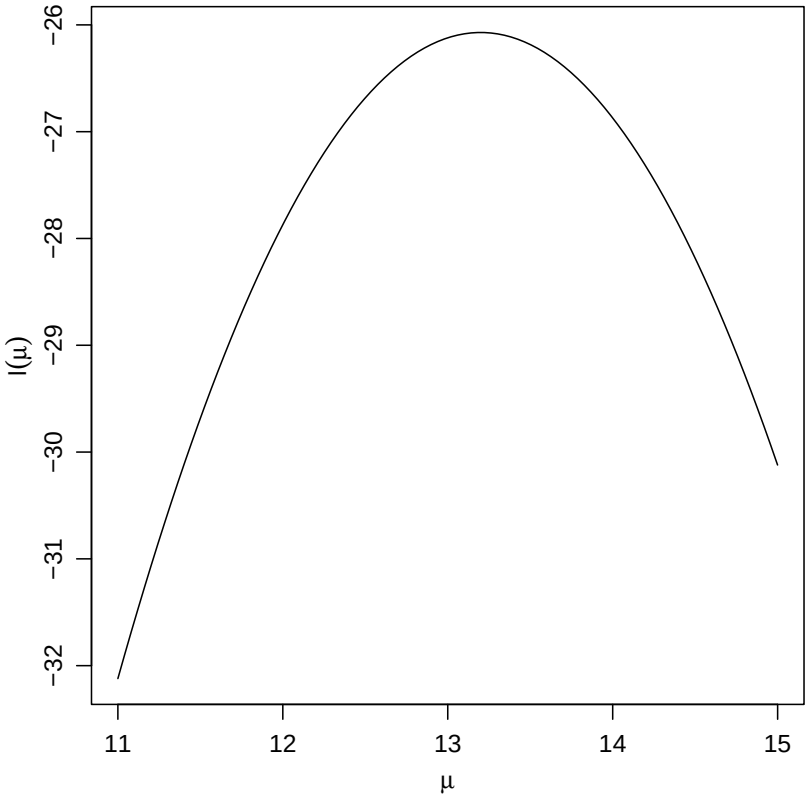



Figura 33: Função de verossimilhança para o parâmetro $\mu$ da distribuição normal com variância $\sigma^2 = 4$ com os dados do Exemplo 1.

5. que é uma função de $\mu$ e portanto devemos fazer um gráfico de $l(\mu)$ *versus* $\mu$ tomando vários valores de $\mu$ 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 $\mu$. 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 33

```
> 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 trabalhosas. 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 $\mu$ e calculamos $l(\mu)$ para cada um dos valores

```
> 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 $\sigma^2$=4. Esta função engloba os comandos acima e pode ser utilizada para obter o gráfico da log-verossimilhança para o parâmetro $\mu$ 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)))
```

## 17.3 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 $\lambda$. 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 34

```
> 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:

```
> 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)
```

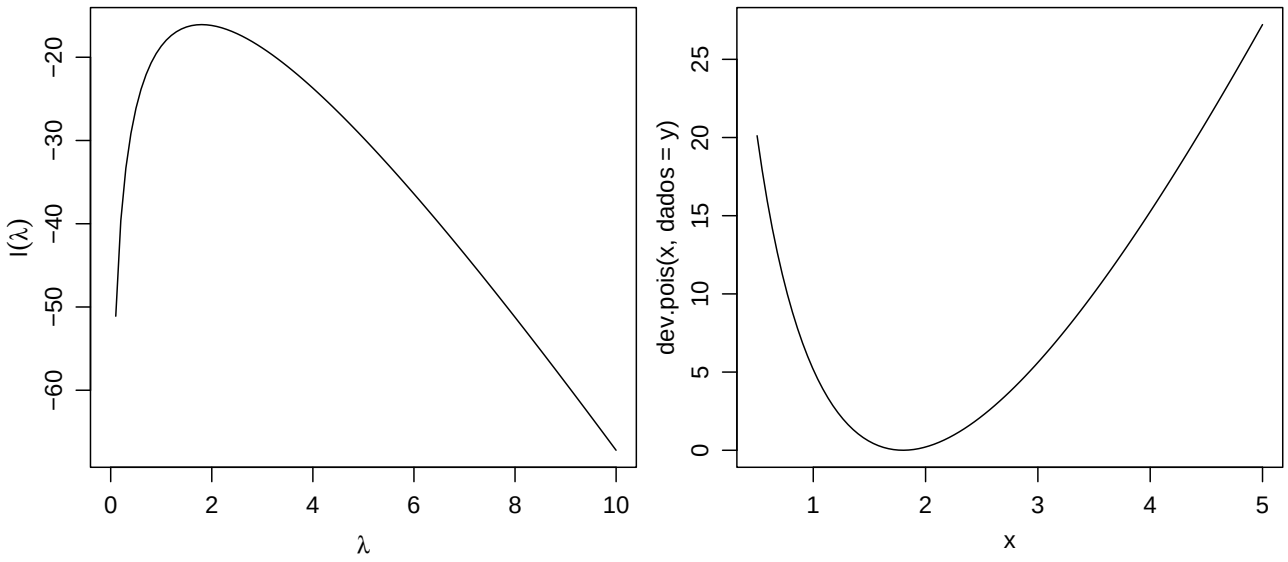



Figura 34: Função de verossimilhança (esquerda) e deviance (direita) para o parâmetro $\lambda$ da distribuição Poisson.

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] 1.075335e-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)
```

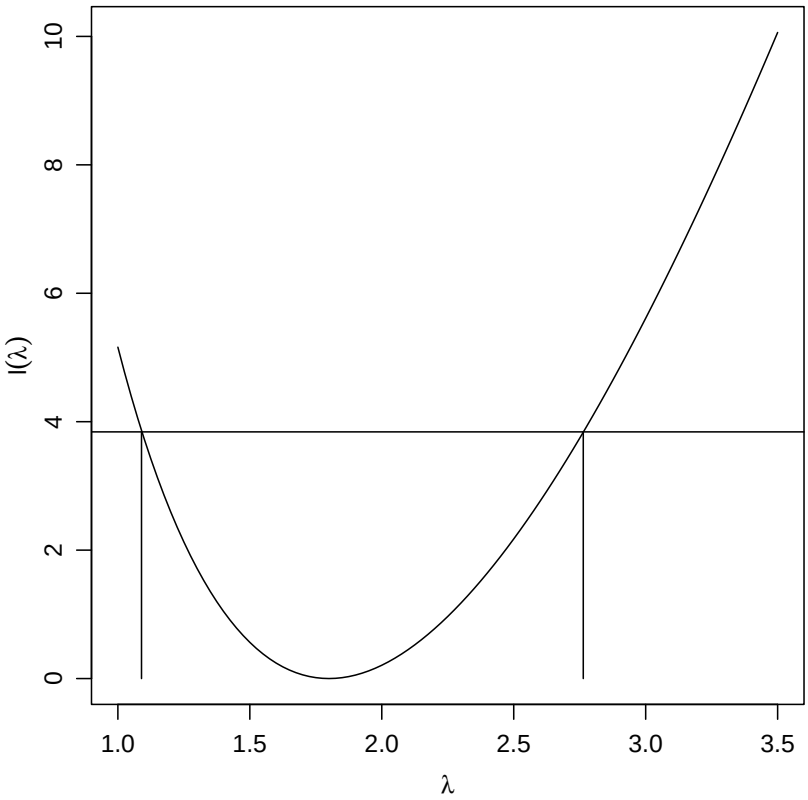



Figura 35: Intervalo de confiança a 95% baseado na deviance para o parâmetro $\lambda$ da distribuição Poisson.

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)
```

## 17.4 Exercícios

1. Seja a amostra abaixo obtida de uma distribuição Poisson de parâmetro $\lambda$.
   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 $\chi^2$ de parâmetro $\nu$.

   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.

# 18 Intervalos de confiança e função de verossimilhança

Nesta aula vamos nos aprofundar um pouco mais na teoria de intervalos de confiança. São ilustrados os conceitos de:

- obtenção de intervalos de confiança pelo método da quantidade pivotal,
- resultados diversos da teoria de verossimilhança,
- intervalos de cobertura.

Voce vai precisar conhecer de conceitos do método da quantidade pivotal, a propriedade de normalidade assintótica dos estimadores de máxima verossimilhança e a distribuição limite da função deviance.

## 18.1 Inferência para a distribuição Bernoulli

Os dados abaixo são uma amostra aleatória da distribuição $Bernoulli(p)$.

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

Desejamos obter:

(a) o gráfico da função de verossimilhança para $p$ com base nestes dados

(b) o estimador de máxima verossimilhança de $p$, a informação observada e a informação de Fisher

(c) um intervalo de confiança de 95% para $p$ baseado na normalidade assintótica de $\hat{p}$

(d) compare o intervalo obtido em (b) com um intervalo de confiança de 95% obtido com base na distribuição limite da função deviance

(e) a probabilidade de cobertura dos intervalos obtidos em (c) e (d). (O verdadeiro valor de $p$ é 0.8)

Primeiramente vamos entrar com os dados na forma de um vetor.

```
> y <- c(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)
```

*(a)*
Vamos escrever uma "função em Rpara obter a função de verossimilhança.

```
> vero.binom <- function(p, dados) {
+     n <- length(dados)
+     x <- sum(dados)
+     return(dbinom(x, size = n, prob = p, log = TRUE))
+ }
```

Esta função exige dados do tipo 0 ou 1 da distribuição Bernoulli. Entretanto às vezes temos dados Binomiais do tipo $n$ e $x$ (número $x$ de sucessos em $n$ observações). Por exemplo, para os dados acima teríamos $n = 25$ e $x = 18$. Vamos então escrever a função acima de forma mais geral de forma que possamos utilizar dados disponíveis tanto em um quanto em ou outro formato.

```
> vero.binom <- function(p, dados, n = length(dados), x = sum(dados)) {
+     return(dbinom(x, size = n, prob = p, log = TRUE))
+ }
```

Agora vamos obter o gráfico da função de verossimilhança para estes dados. Uma forma de fazer isto é criar uma sequência de valores para o parâmetro $p$ e calcular o valor da verossimilhança para cada um deles. Depois fazemos o gráfico dos valores obtidos contra os valores do parâmetro. No R isto pode ser feito com os comandos abaixo que produzem o gráfico mostrado na Figura 36.

```
> p.vals <- seq(0.01, 0.99, l = 99)
> logvero <- sapply(p.vals, vero.binom, dados = y)
> plot(p.vals, logvero, type = "l")
```

Note que os três comandos acima podem ser substituídos por um único que produz o mesmo resultado:

```
> curve(vero.binom(x, dados = y), from = 0, to = 1)
```

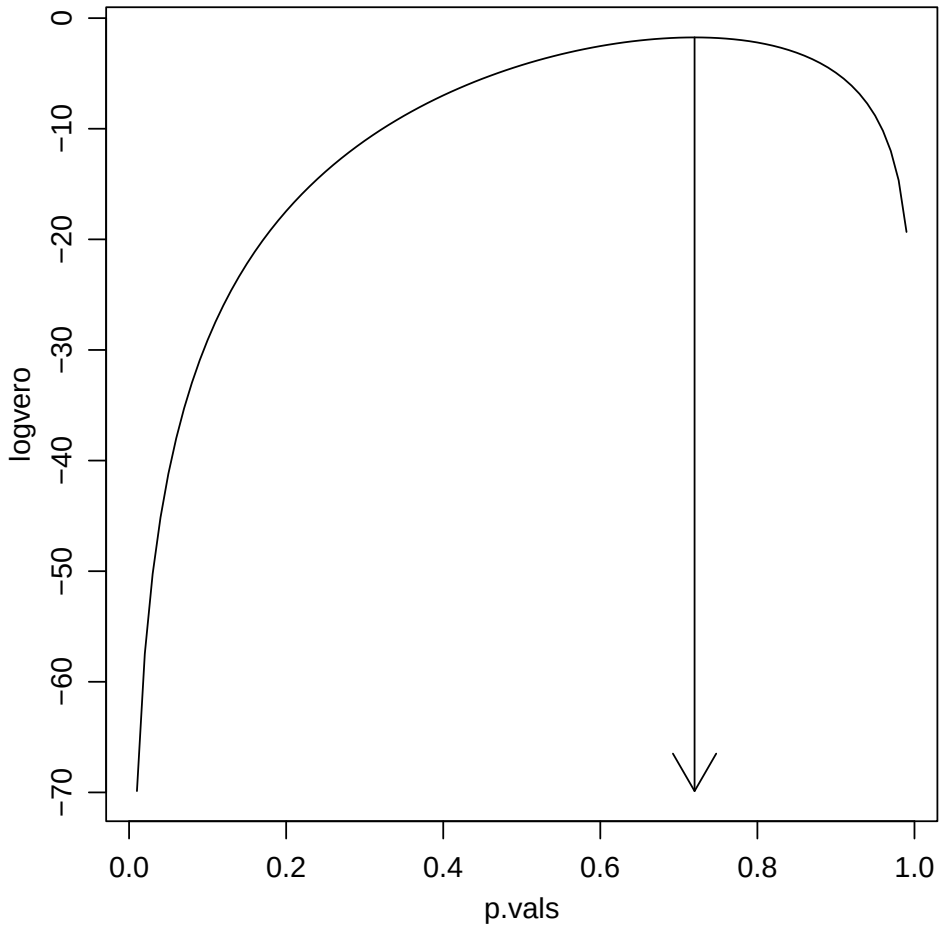



Figura 36: Função de verossimilhança para o parâmetro $p$ da distribuição Bernoulli.

*(b)*
Dos resultados para distribuição Bernoulli sabemos que o estimador de máxima verossimilhança é dado por

$$\hat{p} = \frac{\sum_{i=1}^{n} y_i}{n}$$

e que a informação esperada coincide com a esperança observada e sendo iguais a:

$$I(\hat{p}) = \frac{n}{\hat{p}(1-\hat{p})}$$

. Para indicar o estimador de MV o gráfico poderíamos usar `arrows()` w para obter os valores numéricos para a amostra dada utilizamos os comandos a seguir.

```
> p.est <- mean(y)
> arrows(p.est, vero.binom(p.est, dados = y), p.est, min(logvero))
> io <- ie <- length(y)/(p.est * (1 - p.est))
> io

[1] 124.0079

> ie

[1] 124.0079
```

*(c)*

O intervalo de confiança baseado na normalidade assintótica do estimador de máxima verossimilhança é dado por:

$$\left(\hat{p} - z_{\alpha/2}\sqrt{I(\hat{p})} \ , \ \hat{p} + z_{\alpha/2}\sqrt{I(\hat{p})}\right)$$

e para obter o intervalo no R usamos os comandos a seguir.

```
> ic1.p <- p.est + qnorm(c(0.025, 0.975)) * sqrt(1/ie)
> ic1.p

[1] 0.5439957 0.8960043
```

*(d)*

Vamos agora obter o intervalo baseado na função deviance graficamente. Primeiro vamos escrever uma função para calcular a deviance que vamos chamar de `dev.binom()`, lembrando que a deviance é definida pela expressão:

$$D(p) = 2\{(\hat{p}) - l(p)\}.$$

```
> dev.binom <- function(p, dados, n = length(dados), x = sum(dados)) {
+     p.est <- x/n
+     vero.p.est <- vero.binom(p.est, n = n, x = x)
+     dev <- 2 * (vero.p.est - vero.binom(p, n = n, x = x))
+     dev
+ }
```

E agora vamos fazer o gráfico de forma similar ao que fizemos para função de verossimilhança, definindo uma sequência de valores, calculando as valores da deviance e traçando a curva.

```
> p.vals <- seq(0.3, 0.95, l = 101)
> dev.p <- dev.binom(p.vals, dados = y)
> plot(p.vals, dev.p, typ = "l")

[1] 0.5275 0.8655
```

Agora usando esta função vamos obter o intervalo gráficamente. Para isto definimos o ponto de corte da função usando o fato que a função deviance $D(p)$ tem distribuição assintótica $\chi^2$. Nos comandos a seguir primeiro encontramos o ponto de corte para o nível de confiança de 95%. Depois traçamos a linha de corte com `abline()`. Os comandos seguintes consistem em uma forma simples e aproximada para encontrar os pontos onde a linha conta a função, que definem o intervalo de confiança.

```
[1] 0.5275 0.8655
```

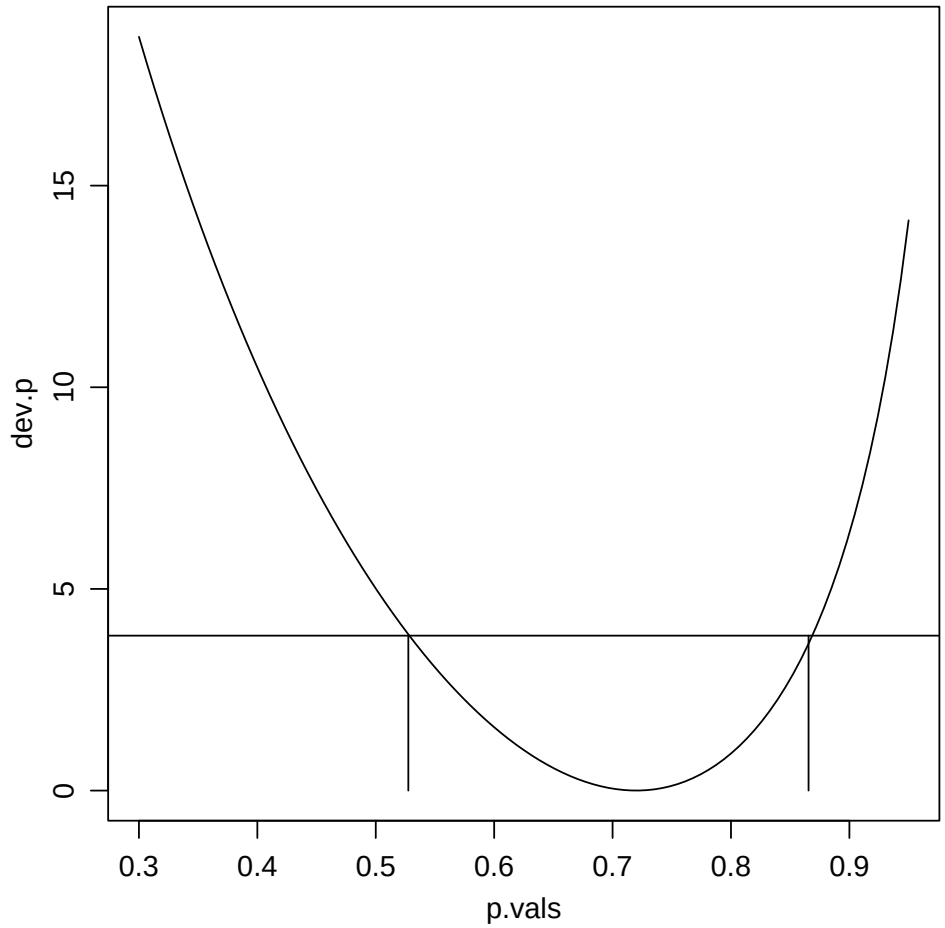



Figura 37: Função deviance para o parâmetro $p$ da distribuição Bernoulli.

```
> corte <- qchisq(0.95, df = 1)
> abline(h = corte)
> dif <- abs(dev.p - corte)
> inf <- ifelse(p.est == 0, 0, p.vals[p.vals < p.est][which.min(dif[p.vals <
+     p.est])])
> sup <- ifelse(p.est == 1, 1, p.vals[p.vals > p.est][which.min(dif[p.vals >
+     p.est])])
> ic2.p <- c(inf, sup)
> ic2.p

[1] 0.5275 0.8655

> segments(ic2.p, 0, ic2.p, corte)
```

Agora que já vimos as duas formas de obter o IC passo a passo vamos usar os comandos acima para criar uma função geral para encontrar IC para qualquer conjunto de dados e com opções para os dois métodos.

```
> ic.binom <- function(dados, n = length(dados), x = sum(dados), nivel = 0.95,
+     tipo = c("assintotico", "deviance")) {
```

```
+     tipo <- match.arg(tipo)
+     alfa <- 1 - nivel
+     p.est <- x/n
+     if (tipo == "assintotico") {
+         se.p.est <- sqrt((p.est * (1 - p.est))/n)
+         ic <- p.est + qnorm(c(alfa/2, 1 - (alfa/2))) * se.p.est
+     }
+     if (tipo == "deviance") {
+         p.vals <- seq(0, 1, l = 1001)
+         dev.p <- dev.binom(p.vals, n = n, x = x)
+         corte <- qchisq(nivel, df = 1)
+         dif <- abs(dev.p - corte)
+         inf <- ifelse(p.est == 0, 0, p.vals[p.vals < p.est][which.min(dif[p.vals <
+             p.est])])
+         sup <- ifelse(p.est == 1, 1, p.vals[p.vals > p.est][which.min(dif[p.vals >
+             p.est])])
+         ic <- c(inf, sup)
+     }
+     names(ic) <- c("lim.inf", "lim.sup")
+     ic
+ }
```

E agora vamos utilizar a função, primeiro com a aproximação assintótica e depois pela deviance. Note que os intervalos são diferentes!

```
> ic.binom(dados = y)

  lim.inf   lim.sup
0.5439957 0.8960043

> ic.binom(dados = y, tipo = "dev")

lim.inf lim.sup
  0.528   0.869
```

*(e)*

O cálculo do intervalo de cobertura consiste em:

1. simular dados com o valor especificado do parâmetro;
2. obter o intervalo de confiança;
3. verificar se o valor está dentro do intervalo
4. repetir (1) a (3) e verificar a proporção de simulações onde o valor está no intervalo.

Espera-se que a proporção obtida seja o mais próximo possível do nível de confiança definido para o intervalo.

Para isto vamos escrever uma função implementando estes passos e que utiliza internamente `ic.binom()` definida acima.

```
> cobertura.binom <- function(n, p, nsim, ...) {
+     conta <- 0
+     for (i in 1:nsim) {
+         ysim <- rbinom(1, size = n, prob = p)
+         ic <- ic.binom(n = n, x = ysim, ...)
+         if (p > ic[1] & p < ic[2])
+             conta <- conta + 1
+     }
+     return(conta/nsim)
+ }
```

E agora vamos utilizar esta função para cada um dos métodos de obtenção dos intervalos.

```
> set.seed(123)
> cobertura.binom(n = length(y), p = 0.8, nsim = 1000)
```

```
[1] 0.885
```

```
> set.seed(123)
> cobertura.binom(n = length(y), p = 0.8, nsim = 1000, tipo = "dev")
```

```
[1] 0.954
```

Note que a cobertura do método baseado na deviance é muito mais próxima do nível de 95% o que pode ser explicado pelo tamanho da amostra. O IC assintótico tende a se aproximar do nível nominal de confiança na medida que a amostra cresce.

## 18.2 Exercícios

1. Refaça o ítem (e) do exemplo acima com $n = 10$, $n = 50$ e $n = 200$. Discuta os resultados.

2. Seja $X_1, X_2, \cdots, X_n$ uma amostra aleatória da distribuição $U(0, \theta)$. Encontre uma quantidade pivotal e:

   (a) construa um intervalo de confiança de 90% para $\theta$

   (b) construa um intervalo de confiança de 90% para $\log \theta$

   (c) gere uma amostra de tamanho $n = 10$ da distribuição $U(0, \theta)$ com $\theta = 1$ e obtenha o intervalo de confiança de 90% para $\theta$. Verifique se o intervalo cobre o verdadeiro valor de $\theta$.

   (d) verifique se a probabilidade de cobertura do intervalo é consistente com o valor declarado de 90%. Para isto gere 1000 amostras de tamanho $n = 10$. Calcule intervalos de confiança de 90% para cada uma das amostras geradas e finalmente, obtenha a proporção dos intervalos que cobrem o verdadeiro valor de $\theta$. Espera-se que este valor seja próximo do nível de confiança fixado de 90%.

   (e) repita o item (d) para amostras de tamanho $n = 100$. Houve alguma mudança na probabilidade de cobertura?

   Note que se $-\sum_i^n \log F(x_i; \theta) \sim \Gamma(n, 1)$ então $-2\sum_i^n \log F(x_i; \theta) \sim \chi^2_{2n}$.

3. Acredita-se que o número de trens atrasados para uma certa estação de trem por dia segue uma distribuição Poisson($\theta$), além disso acredita-se que o número de trens atrasados em cada dia seja independente do valor de todos os outros dias. Em 10 dias sucessivos, o número de trens atrasados foi registrado em:

```
5 0 3 2 1 2 1 1 2 1
```

Obtenha:

(a) o gráfico da função de verossimilhança para $\theta$ com base nestes dados

(b) o estimador de máxima verossimilhança de $\theta$, a informação observada e a informação de Fisher

(c) um intervalo de confiança de 95% para o número médio de trens atrasados por dia baseando-se na normalidade assintótica de $\hat{\theta}$

(d) compare o intervalo obtido em (c) com um intervalo de confiança obtido com base na distribuição limite da função deviance

(e) o estimador de máxima verossimilhança de $\phi$, onde $\phi$ é a probabilidade de que não hajam trens atrasados num particular dia. Construa intervalos de confiança de 95% para $\phi$ como nos itens (c) e (d).

4. Encontre intervalos de confiança de 95% para a média de uma distribuição Normal com variância 1 dada a amostra

```
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
```

baseando-se:

(a) na distribuição assintótica de $\hat{\mu}$

(b) na distribuição limite da função deviance

5. Acredita-se que a produção de trigo, $X_i$, da área $i$ é normalmente distribuída com média $\theta z_i$, onde $z_i$ é quantidade (conhecida) de fertilizante utilizado na área. Assumindo que as produções em diferentes áreas são independentes, e que a variância é conhecida e igual a 1, ou seja, $X_i \sim N(\theta z_i, 1)$, para $i = 1, \cdots, n$:

(a) simule dados sob esta distribuição assumindo que $\theta = 1.5$, e $z = (1, 2, 3, 4, 5)$. Visualize os dados simulados através de um gráfico de $(z \times x)$

(b) encontre o EMV de $\theta$, $\hat{\theta}$

(c) mostre que $\hat{\theta}$ é um estimador não viciado para $\theta$ (lembre-se que os valores de $z_i$ são constantes)

(d) obtenha um intervalo de aproximadamente 95% de confiança para $\theta$ baseado na distribuição assintótica de $\hat{\theta}$

# 19 Intervalos de confiança baseados na deviance

Neste sessão discutiremos a obtenção de intervalos de confiança baseado na função deviance.

## 19.1 Média da distribuição normal com variância conhecida

Seja $X_1, \ldots, X_n$ a.a. de uma distribuição normal de média $\theta$ e variância 1. Vimos que:

1. A função de log-verossimilhança é dada por $l(\theta) = \text{cte} + \frac{1}{2}\sum_{i=1}^{n}(x_i - \theta)^2$;
2. o estimador de máxima verossimilhança é $\hat{\theta} = \frac{\sum_{i=1}^{n} X_i}{n} = \bar{X}$;
3. a função deviance é $D(\theta) = n(\bar{x} - \theta)^2$;
4. e neste caso a deviance tem distribuição exata $\chi^2_{(1)}$;
5. e os limites do intervalo são dados por $\bar{x} \stackrel{+}{-} \sqrt{c^*/n}$, onde $c^*$ é o quantil $(1-\alpha/2)$ da distribuição $\chi^2_{(1)}$.

Vamos considerar que temos uma amostra onde $n = 20$ e $\bar{x} = 32$. Neste caso a função deviance é como mostrada na Figura 38 que é obtida com os comandos abaixo onde primeiro definimos uma função para calcular a deviance que depois é mostrada em um gráfico para valores entre 30 e 34. Para obtermos um intervalo a 95% de confiança escolhemos o quantil correspondente na distribuição $\chi^2_{(1)}$ e mostrado pela linha tracejada no gráfico. Os pontos onde esta linha cortam a função são, neste exemplo, determinados analiticamente pela expressão dada acima e indicados pelos setas verticais no gráfico.

```
> dev.norm.v1 <- function(theta, n, xbar) {
+     n * (xbar - theta)^2
+ }
> thetaN.vals <- seq(31, 33, l = 101)
> dev.vals <- dev.norm.v1(thetaN.vals, n = 20, xbar = 32)
> plot(thetaN.vals, dev.vals, ty = "l", xlab = expression(theta),
+     ylab = expression(D(theta)))
> corte <- qchisq(0.95, df = 1)
> abline(h = corte, lty = 3)
> limites <- 32 + c(-1, 1) * sqrt(corte/20)
> limites
> segments(limites, rep(corte, 2), limites, rep(0, 2))
```

Vamos agora examinar o efeito do tamanho da amostra na função. A Figura 39 mostra as funções para três tamanhos de amostra, $n = 10, 20$ e 50 que são obtidas com os comandos abaixo. A linha horizontal mostra o efeito nas amplitudes dos IC's.

```
> dev10.vals <- dev.norm.v1(thetaN.vals, n = 10, xbar = 32)
> plot(thetaN.vals, dev10.vals, ty = "l", xlab = expression(theta),
+     ylab = expression(D(theta)))
> dev20.vals <- dev.norm.v1(thetaN.vals, n = 20, xbar = 32)
> lines(thetaN.vals, dev20.vals, lty = 2)
> dev50.vals <- dev.norm.v1(thetaN.vals, n = 50, xbar = 32)
> lines(thetaN.vals, dev50.vals, lwd = 2)
> abline(h = qchisq(0.95, df = 1), lty = 3)
> legend(31, 2, c("n=10", "n=20", "n=50"), lty = c(1, 2, 1), lwd = c(1,
+     1, 2), cex = 0.7)
```

```
[1] 31.56174 32.43826
```

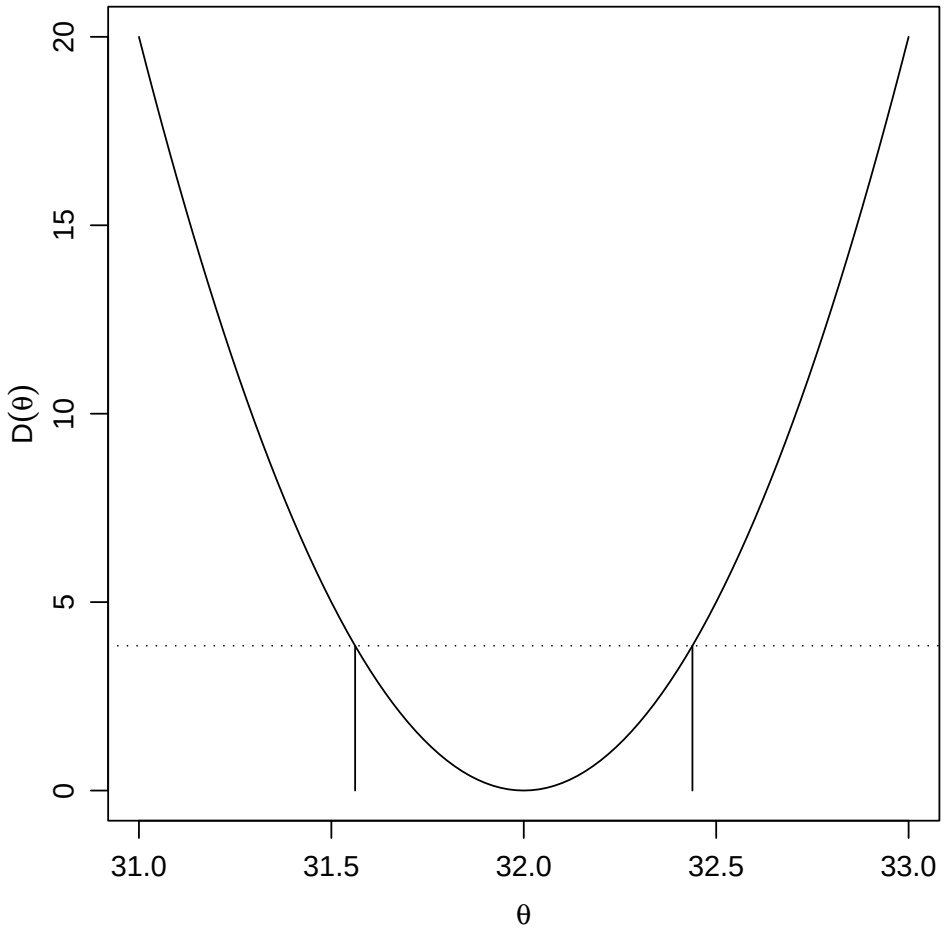



Figura 38: Função deviance para $\mathrm{N}(\theta, 1)$ para uma amostra de tamanho 20 e média 32.

## 19.2 IC para o parâmetro da distribuição exponencial

Seja $X_1, \ldots, X_n$ a.a. de uma distribuição exponencial de parâmetro $\theta$ com função de densidade $f(x) = \theta \exp\{-\theta x\}$. Vimos que:

1. A função de log-verossimilhança é dada por $l(\theta) = n \log(\theta) - \theta n \bar{x}$;
2. o estimador de máxima verossimilhança é $\hat{\theta} = \frac{n}{\sum_{i=1}^{n} X_i} = \frac{1}{\bar{X}}$;
3. a função deviance é $D(\theta) = 2n \left[\log(\hat{\theta}/\theta) + \bar{x}(\theta - \hat{\theta})\right]$;
4. e neste caso a deviance tem distribuição assintótica $\chi^2_{(1)}$;
5. e os limites do intervalo não podem ser obtidos analiticamente, devendo ser obtidos por:
   - métodos numéricos ou gráficos, ou,
   - pela aproximação quadrática da verossimilhança por série de Taylor que neste caso fornece uma expressão da deviance aproximada dada por $D(\theta) \approx n \left(\frac{\theta - \hat{\theta}}{\hat{\theta}}\right)^2$.

A seguir vamos ilustrar a obtenção destes intervalos no R. Vamos considerar que temos uma amostra onde $n = 20$ e $\bar{x} = 10$ para a qual a função deviance é mostrada na Figura 40 e obtida de forma análoga ao exemplo anterior.

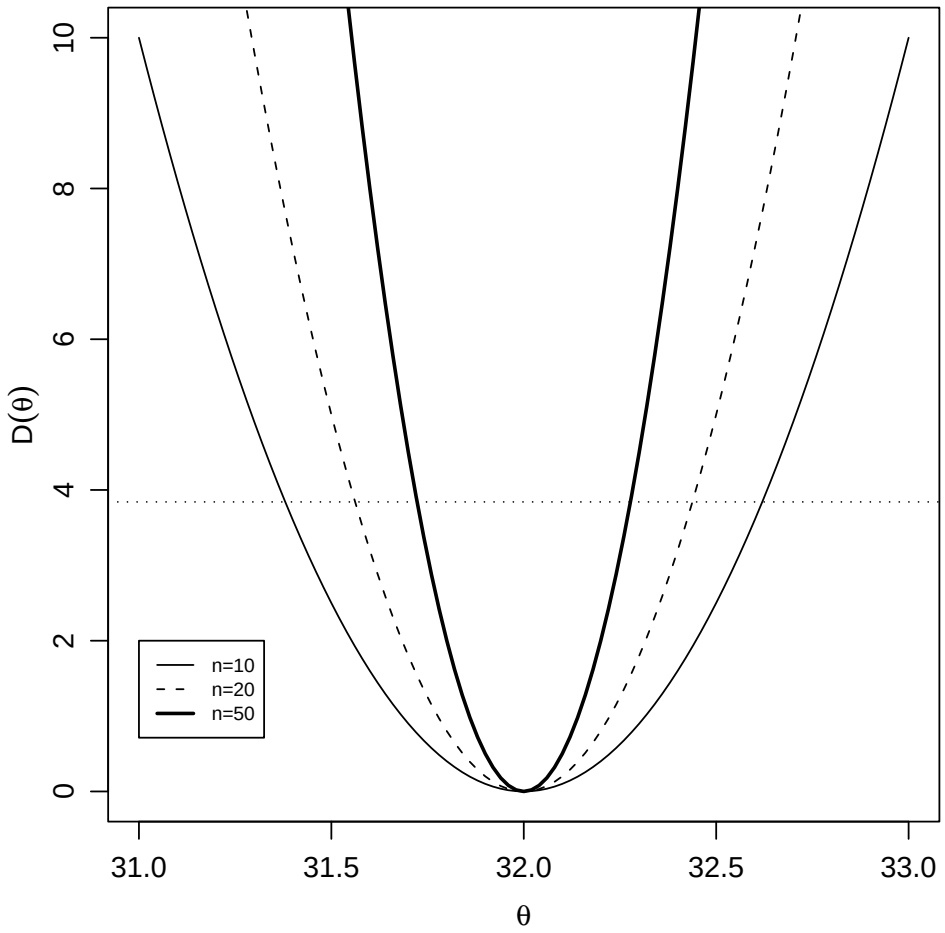



Figura 39: Funções deviance para o parâmetro $\theta$ da $\mathrm{N}(\theta, 1)$ para amostras de média 32 e tamanhos de amostra $n = 10, 20$ e 50.

```
> dev.exp <- function(theta, n, xbar) {
+     2 * n * (log((1/xbar)/theta) + xbar * (theta - (1/xbar)))
+ }
> thetaE.vals <- seq(0.04, 0.2, l = 101)
> dev.vals <- dev.exp(thetaE.vals, n = 20, xbar = 10)
> plot(thetaE.vals, dev.vals, ty = "l", xlab = expression(theta),
+     ylab = expression(D(theta)))
```

Neste exemplo, diferentemente do anterior, não determinamos a distribuição exata da deviance e usamos a distribuição assintótica $\chi^2_{(1)}$ na qual se baseia a linha de corte tracejada mostrada no gráfico para definir o IC do parâmetro ao nível de 95% de confiança.

Para encontrar os limites do IC precisamos dos valores no eixo dos parâmetros nos pontos onde a linha de corte toca a função deviance o que corresponde a resolver a equação $D(\theta) = 2n\left[\log(\hat{\theta}/\theta) + \bar{x}(\theta - \hat{\theta})\right] = c^*$ onde $c^*$ é quantil da distribuição da $\chi^2$ com 1 grau de liberdade correspondente ao nível de confiança desejado. Por exemplo, para 95% o valor de $\chi^2_{1,0.95}$ é 3.84. Como esta equação não tem solução analítica (diferentemente do exemplo anterior) vamos examinar a seguir duas possíveis soluções para encontrar os limites do intervalo.

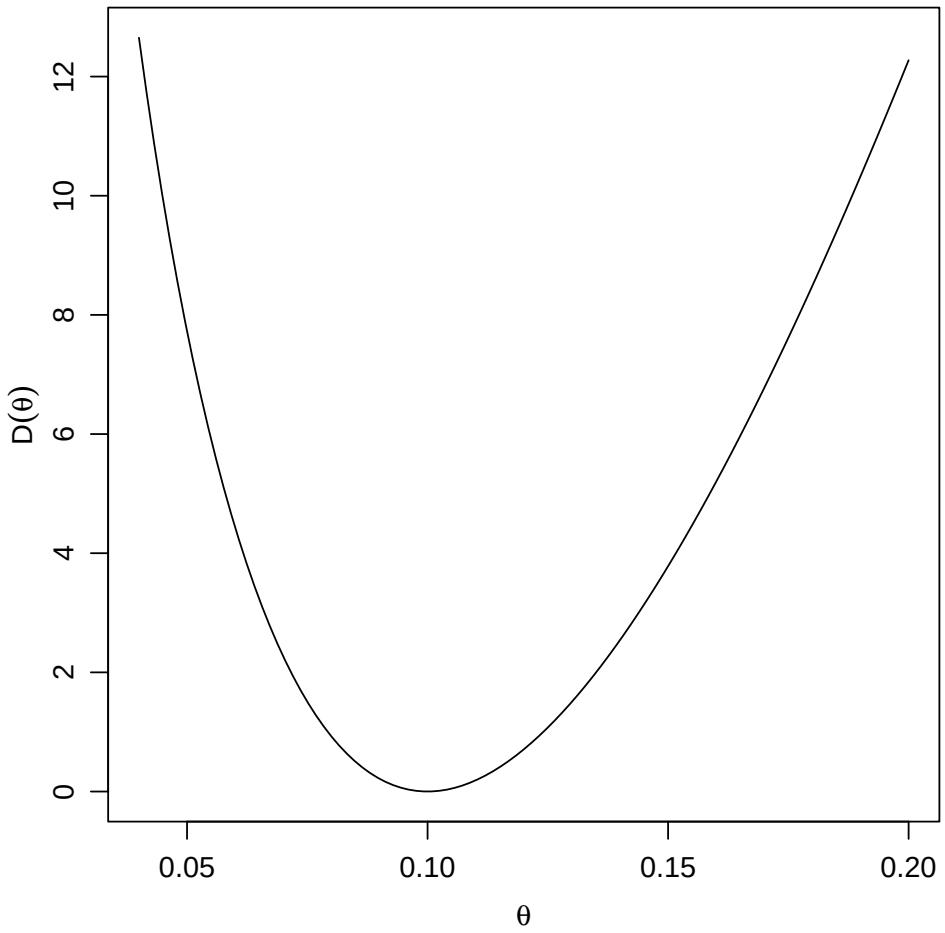



Figura 40: Função deviance da Exp($\theta$) para uma amostra de tamanho 20 e média 10.

### 19.2.1 Solução numérica/gráfica simplificada

Iremos aqui considerar uma solução simples baseada no gráfico da função deviance para encontrar os limites do IC que consiste no seguinte: Para fazermos o gráfico da deviance criamos uma sequência de valores do parâmetro $\theta$. A cada um destes valores corresponde um valor de $D(\theta)$. Vamos então localizar os valores de $\theta$ para os quais $D(\theta)$ é o mais próximo possível do ponto de corte. Isto é feito com o código abaixo e o resultado exibido na Figura 41.

```
> plot(thetaE.vals, dev.vals, ty = "l", xlab = expression(theta),
+     ylab = expression(D(theta)))
> corte <- qchisq(0.95, df = 1)
> abline(h = corte, lty = 3)
> dif <- abs(dev.vals - corte)
> linf <- thetaE.vals[thetaE.vals < (1/10)][which.min(dif[thetaE.vals <
+     (1/10)])]
> lsup <- thetaE.vals[thetaE.vals > (1/10)][which.min(dif[thetaE.vals >
+     (1/10)])]
> limites.dev <- c(linf, lsup)
> limites.dev

[1] 0.0624 0.1504

> segments(limites.dev, rep(corte, 2), limites.dev, rep(0, 2))
```

```
[1] 0.0624 0.1504
```

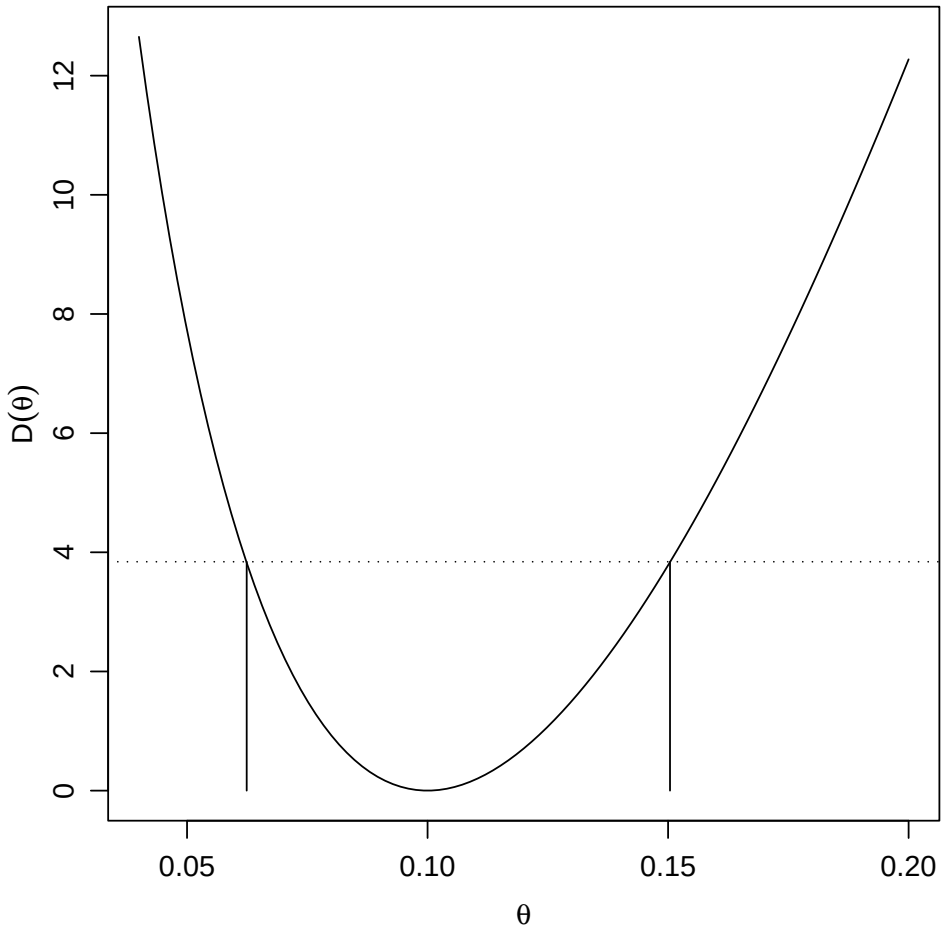



Figura 41: Obtenção gráfica do IC para o parâmetro $\theta$ da Exp($\theta$) para uma amostra de tamanho 20 e média 10.

Note que neste código procuramos primeiro e limite inferior entre os valores menores que a estimativa do parâmetro (1/10) e depois o limite superior entre os valores maiores que esta estimativa. Embora este procedimento bastante simples e sujeito a imprecisão podemos torná-lo quão preciso quanto quisermos bastando para isto definir um vetor com menor espaçamento para os valores para o parâmetro, por exemplo poderiamos usar `thetaE.vals <- seq(0.04,0.20,l=1001)`.

### 19.2.2 Aproximação quadrática da verossimilhança

Nesta abordagem aproximamos a função deviance por uma função quadrática obtida pela expansão por série de Taylor ao redor do estimador de máxima verossimilhança:

$$D(\theta) \approx n\left(\frac{\theta-\hat{\theta}}{\hat{\theta}}\right)^2.$$

A Figura 42 obtida com os comandos mostra o gráfico desta função deviance aproximada. A Figura também mostra os IC's obtido com esta função. Para a aproximação quadrática os limites dos intervalos são facilmente determinados analiticamente e neste caso dados por:

$$\left(\hat{\theta}(1-\sqrt{c^*/n}) \;\;,\;\; \hat{\theta}(1+\sqrt{c^*/n})\right).$$

```
> devap.exp <- function(theta, n, xbar) {
+     n * (xbar * (theta - (1/xbar)))^2
+ }
> devap.vals <- devap.exp(thetaE.vals, n = 20, xbar = 10)
> plot(thetaE.vals, devap.vals, ty = "l", xlab = expression(theta),
+     ylab = expression(D(theta)))
> corte <- qchisq(0.95, df = 1)
> abline(h = corte, lty = 3)
> limites.devap <- c((1/10) * (1 - sqrt(corte/20)), (1/10) * (1 +
+     sqrt(corte/20)))
> limites.devap

[1] 0.05617387 0.14382613

> segments(limites.devap, rep(corte, 2), limites.devap, rep(0, 2))


[1] 0.05617387 0.14382613
```

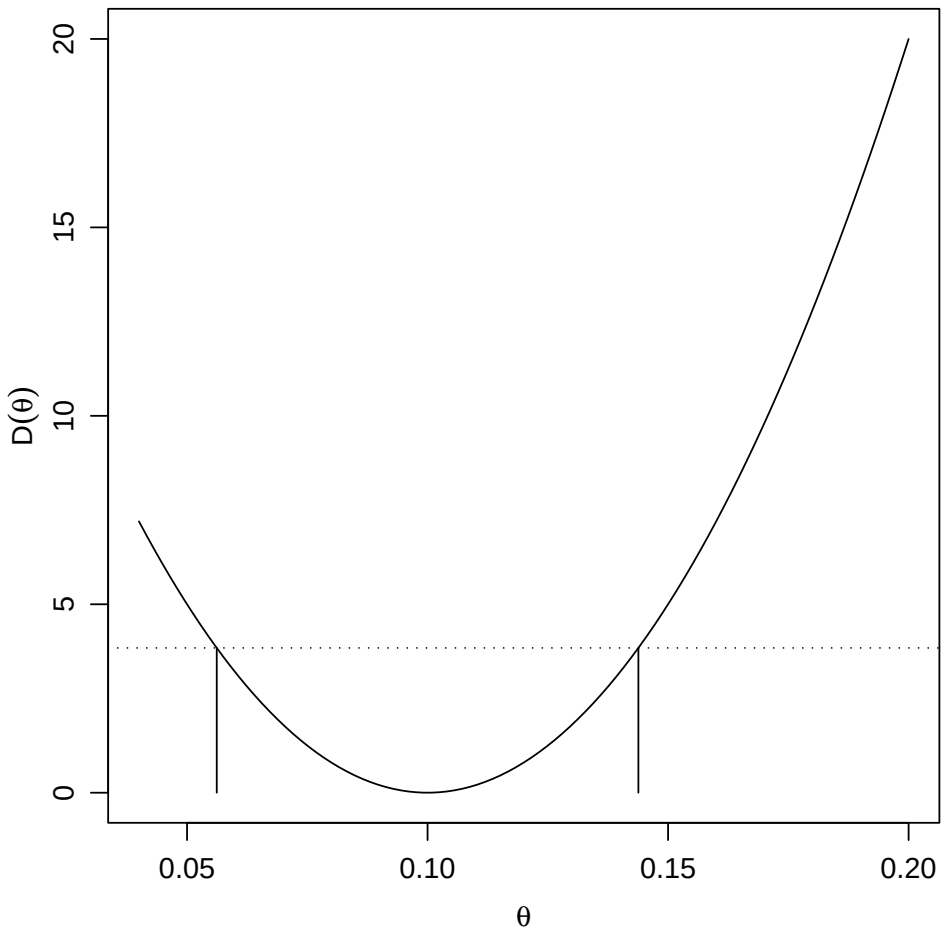



Figura 42: Função deviance obtida pela aproximação quadrática para Exp($\theta$) e uma amostra de tamanho 20 e média 10.

## 19.3 Comparando as duas estratégias

Examinando os limites dos intervalos encontrados anteriormente podemos ver que são diferentes. Vamos agora colocar os resultados pelos dois métodos em um mesmo gráfico (Figura 43) para comparar os resultados.

```
> plot(thetaE.vals, dev.vals, ty = "l", xlab = expression(theta),
+     ylab = expression(D(theta)))
> lines(thetaE.vals, devap.vals, lty = 2)
> abline(h = corte, lty = 3)
> segments(limites.dev, rep(corte, 2), limites.dev, rep(0, 2))
> segments(limites.devap, rep(corte, 2), limites.devap, rep(0, 2),
+     lty = 2)
> legend(0.07, 12, c("deviance", "aproximacão quadrática"), lty = c(1,
+     2), cex = 0.8)
```

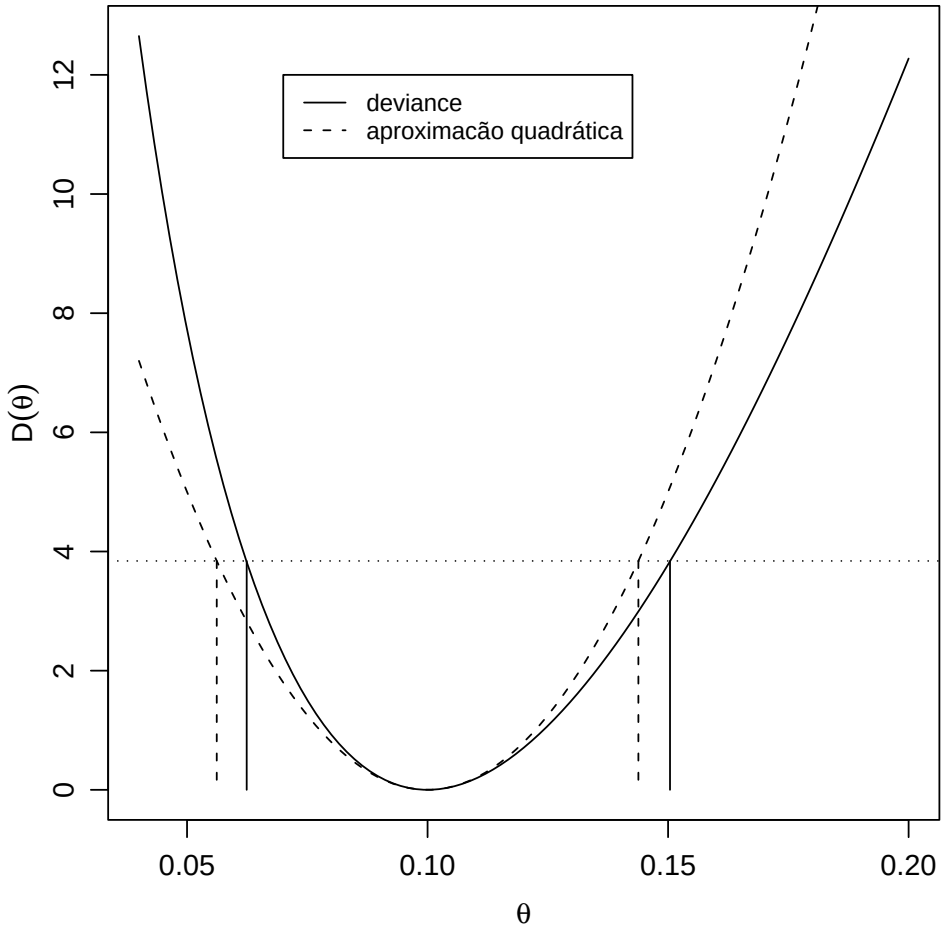



Figura 43: Comparação dos IC's de confiança obtidos pela solução gráfica/numérica (linha sólida) e pela aproximação quadrática (linha tracejada) para o parâmetro $\theta$ da Exp($\theta$) para uma amostra de tamanho 20 e média 10.

Vamos agora examinar o efeito do tamanho da amostra na função deviance e sua aproximação quadrática. A Figura 44 mostra as funções para três tamanhos de amostra, $n = 10, 30$ e 100 que são obtidas com os comandos abaixo onde vemos que a aproximação fica cada vez melhor com o aumento do tamanho da amostra.

```
> thetaE.vals <- seq(0.04, 0.2, l = 101)
> dev10.vals <- dev.exp(thetaE.vals, n = 10, xbar = 10)
> plot(thetaE.vals, dev10.vals, ty = "l", xlab = expression(theta),
+     ylab = expression(D(theta)))
> devap10.vals <- devap.exp(thetaE.vals, n = 10, xbar = 10)
> lines(thetaE.vals, devap10.vals, lty = 2)
> abline(h = qchisq(0.95, df = 1), lty = 3)
> dev30.vals <- dev.exp(thetaE.vals, n = 30, xbar = 10)
> plot(thetaE.vals, dev30.vals, ty = "l", xlab = expression(theta),
+     ylab = expression(D(theta)))
> devap30.vals <- devap.exp(thetaE.vals, n = 30, xbar = 10)
> lines(thetaE.vals, devap30.vals, lty = 2)
> abline(h = qchisq(0.95, df = 1), lty = 3)
> dev100.vals <- dev.exp(thetaE.vals, n = 100, xbar = 10)
> plot(thetaE.vals, dev100.vals, ty = "l", xlab = expression(theta),
+     ylab = expression(D(theta)))
> devap100.vals <- devap.exp(thetaE.vals, n = 100, xbar = 10)
> lines(thetaE.vals, devap100.vals, lty = 2)
> abline(h = qchisq(0.95, df = 1), lty = 3)
```

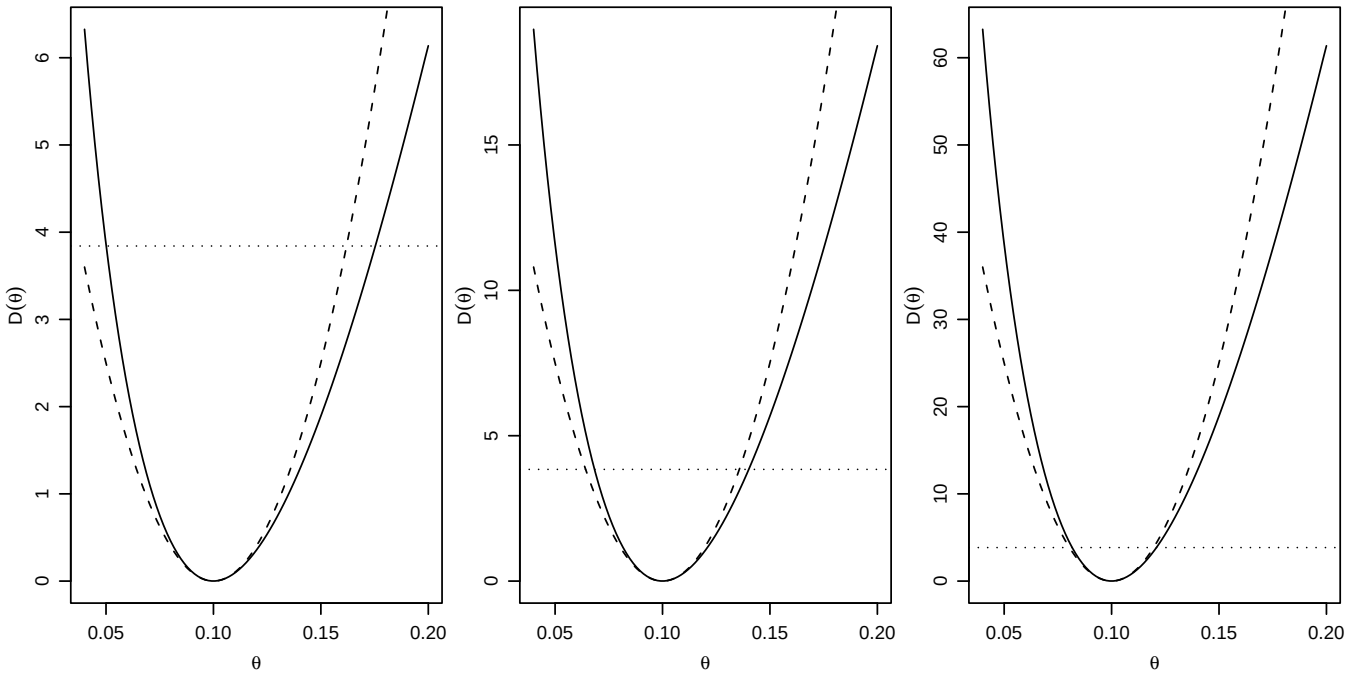



Figura 44: Funções deviance e deviance aproximada para o parâmetro $\theta$ da Exp($\theta$) em amostras de média 10 e tamanhos $n = 10$ (esquerda), 30 (centro) e 100 (direita).

## 19.4 Exercícios

1. Seja 14.1, 30.0, 19.6, 28.2, 12.5, 15.2, 17.1, 11.0, 25.9, 13.2, 22.8, 22.1 a.a. de uma distribuição normal de média 20 e variância $\sigma^2$.

   (a) Obtenha a função deviance para $\sigma^2$ e faça o seu gráfico.

(b) Obtenha a função deviance para $\sigma$ e faça o seu gráfico.

(c) Obtenha os IC's a 90% de confiança.

2. Repita as análises mostradas no exemplo acima da distribuição exponencial mas agora utilizando a seguinte parametrização para a função de densidade:

$$f(x) = \frac{1}{\lambda} \exp\left(-x/\lambda\right) \quad x \geq 0.$$

Discuta as diferenças entre os resultados obtidos nas duas parametrizações.

# 20 Ilustrando propriedades de estimadores

## 20.1 Consistência

Um estimador é consistente quando seu valor se aproxima do verdadeiro valor do parâmetro à medida que aumenta-se o tamanho da amostra. Vejamos como podemos ilustrar este resultado usando simulação. A idéia básica é a seguite:

1. escolher uma distribuição e seus parâmetros,
2. definir o estimador,
3. definir uma sequência crescente de valores de tamanho de amostras,
4. obter uma amostra de cada tamanho,
5. calcular a estatística para cada amostra,
6. fazer um gráfico dos valores das estimativas contra o tamanho de amostra, indicando neste gráfico o valor verdadeiro do parâmetro.

### 20.1.1 Média da distribuição normal

Seguindo os passos acima vamos:

1. tomar a distribuição Normal de média 10 e variância 4,
2. definir o estimador $\bar{X} = \sum_{i=1}^{n} \frac{x_i}{n}$,
3. escolhemos os tamanhos de amostra $n = 2, 5, 10, 15, 20, \ldots, 1000, 1010, 1020, \ldots, 5000$,
4. fazemos os cálculos e produzimos um gráfico como mostrado na 45 com os comandos a seguir.

```
> ns <- c(2, seq(5, 1000, by = 5), seq(1010, 5000, by = 10))
> estim <- numeric(length(ns))
> for (i in 1:length(ns)) {
+     amostra <- rnorm(ns[i], 10, 4)
+     estim[i] <- mean(amostra)
+ }
> plot(ns, estim)
> abline(h = 10)
```

## 20.2 Momentos das distribuições amostrais de estimadores

Para inferência estatística é necessário conhecer a distribuição amostral dos estimadores. Em alguns casos estas distribuições são derivadas analiticamente. Isto se aplica a diversos resultados vistos em um curso de Inferência Estatística. Por exemplo o resultado visto na sessão 26: se $Y_1, Y_2, \ldots Y_n \sim \mathrm{N}(\mu, \sigma^2)$ então $\bar{y} \sim \mathrm{N}(\mu, \sigma^2/n)$. Resultados como estes podem ser ilustrados computacionalmente como visto na Sessão 26.

Além disto este procedimento permite investigar distribuições amostrais que são complicadas ou não podem ser obtidas analiticamente.

Vamos ver um exemplo: considere $Y$ uma v.a. com distribuição normal $N(\mu, \sigma^2)$ e seja um parâmetro de interesse $\theta = \mu/\sigma^2$. Para obter por simulação a esperança e variância do estimador $T = \bar{Y}/S^2$ onde $\bar{Y}$ é a média e $S^2$ a variância de uma amostra seguimos os passos:

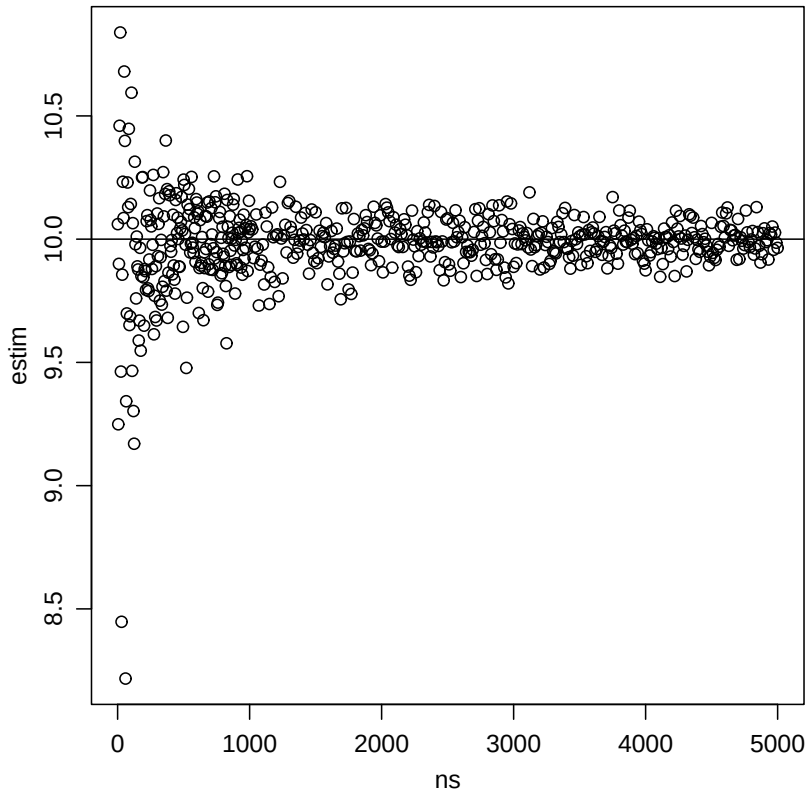



Figura 45: Médias de amostras de diferentes tamanhos.

1. escolher uma distribuição e seus parâmetros, no caso vamos escolher uma $N(180, 64)$,
2. definir um tamanho de amostra, no caso escolhemos $n = 20$,
3. obter por simulação um número $N$ de amostras, vamos usar $N = 1000$,
4. calcular a estatística de interesse para cada amostra,
5. usar as amostras para obter as estimativas $\hat{E}[T]$ e $\hat{\text{Var}}[T]$.

Vamos ver agora comandos do R.

```
> amostras <- matrix(rnorm(20 * 1000, mean = 180, sd = 8), nc = 1000)
> Tvals <- apply(amostras, 2, function(x) {
+     mean(x)/var(x)
+ })
> ET <- mean(Tvals)
> ET

[1] 3.133945

> VarT <- var(Tvals)
> VarT

[1] 1.329038
```

Nestes comandos primeiro obtemos 1000 amostras de tamanho 20 que armazenamos em uma matriz de dimensão $20 \times 1000$, onde cada coluna é uma amostra. A seguir usamos a função `apply`

para calcular a quantidade desejada que definimos com `function(x) {mean(x)/var(x)}`. No caso anterior foi obtido $\hat{E}[T] \approx 3.13$ e $\hat{\text{Var}}[T] \approx 1.33$.

Se voce rodar os comandos acima deverá obter resultados um pouco diferentes (mas não muito!) pois nossas amostras da distribuição normal não são as mesmas. Para obter as masmas amostras teríamos que usar a mesma semente para geração de números aleatórios.

## 20.3 Não-tendenciosidade

Fica como exercício.

## 20.4 Variância mínima

Fica como exercício.

## 20.5 Exercícios

1. Ilustre a consistência do estimador $\hat{\lambda} = 1/\bar{X}$ de uma distribuição exponencial $f(x) = \lambda \exp\{-\lambda x\}$.

2. No exemplo dos momentos das distribuições de estimadores visto em (20.2) ilustramos a obtenção dos momentos para um tamanho fixo de amostra $n = 20$. Repita o procedimento para vários tamanho de amostra e faça um gráfico mostrando o comportamento de $\hat{E}[T]$ e $\hat{\text{Var}}[T]$ em função de $n$.

3. Estime por simulação a esperança e variância do estimador $\hat{\lambda} = \bar{X}$ de uma distribuição de Poisson de parâmetro $\lambda$ para um tamanho de amostra $n = 30$. Compare com os valores obtidos analiticamente. Mostre em um gráfico como os valores de $\hat{E}[\hat{\lambda}]$ e $\hat{\text{Var}}[\hat{\lambda}]$ variam em função de $n$.

4. Crie um exemplo para ilustrar a não tendenciosidade de estimadores. Sugestão: compare os estimadores $S^2 = \sum_{i=1}^{n}(X_1 - \bar{X})^2/(n-1)$ e $\hat{\sigma}^2 = \sum_{i=1}^{n}(X_1 - \bar{X})^2/n$ do parâmetro de variância $\sigma^2$ de uma distribuição normal.

5. Crie um exemplo para comparar a variância de dois estimadores. Por exemplo compare por simulação as variâncias dos estimadores $T_1 = \bar{X}$ e $T_2 = (X_{[1]} + X_{[n]})/2$ do parâmetro $\mu$ de uma distribuição $\text{N}(\mu, \sigma^2)$, onde $X_{[1]}$ e $X_{[n]}$ são os valores mínimo e máximo da amostra, respectivamente.

# 21 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

## 21.1 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
```

### 21.1.1 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.

### 21.1.2 Escrevendo uma função

Esta fica por sua conta!
Escreva a sua própria função para testar hipóteses sobre variâncias de duas distribuições normais.

### 21.1.3 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.

## 21.2 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 $\mu$ e variância $400g^2$. O valor de $\mu$ 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?

# 22 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.

## 22.1 Teste $\chi^2$ 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 $\chi^2$, 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 $\chi^2$.

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 $\chi^2$ 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.

## 22.2 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.

## 22.3 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.

## 22.4 Exercícios

1. Revisite os dados `milsa` visto na aula de estatística descritiva e selecione pares de variáveis adequadas para efetuar:
   (a) um teste $\chi^2$
   (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.

# 23 Transformação de dados

Tranformação de dados é uma das possíveis formas de contarnar o problema de dados que não obedecem os pressupostos da análise de variância. Vamos ver como isto poder ser feito com o programa R.

Considere o seguinte exemplo da apostila do curso.

Tabela 4: Número de reclamações em diferentes sistemas de atendimento

| Trat | Repetições | | | | | |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 2370 | 1687 | 2592 | 2283 | 2910 | 3020 |
| 2 | 1282 | 1527 | 871 | 1025 | 825 | 920 |
| 3 | 562 | 321 | 636 | 317 | 485 | 842 |
| 4 | 173 | 127 | 132 | 150 | 129 | 227 |
| 5 | 193 | 71 | 82 | 62 | 96 | 44 |

Inicialmente vamos entrar com os dados usando `scan()` e montar um *data-frame*.

```
> y <- scan()
1: 2370
2: 1687
3: 2592
...
30: 44
31:
Read 30 items
```

```
> tr <- data.frame(trat = factor(rep(1:5, each = 6)), resp = y)
> tr
```

```
   trat resp
1     1 2370
2     1 1687
3     1 2592
4     1 2283
5     1 2910
6     1 3020
7     2 1282
8     2 1527
9     2  871
10    2 1025
11    2  825
12    2  920
13    3  562
14    3  321
15    3  636
16    3  317
17    3  485
18    3  842
```

```
19     4  173
20     4  127
21     4  132
22     4  150
23     4  129
24     4  227
25     5  193
26     5   71
27     5   82
28     5   62
29     5   96
30     5   44
```

A seguir vamos fazer ajustar o modelo e inspecionar os resíduos.

```
> tr.av <- aov(resp ~ trat, data = tr)
> plot(tr.av)
```

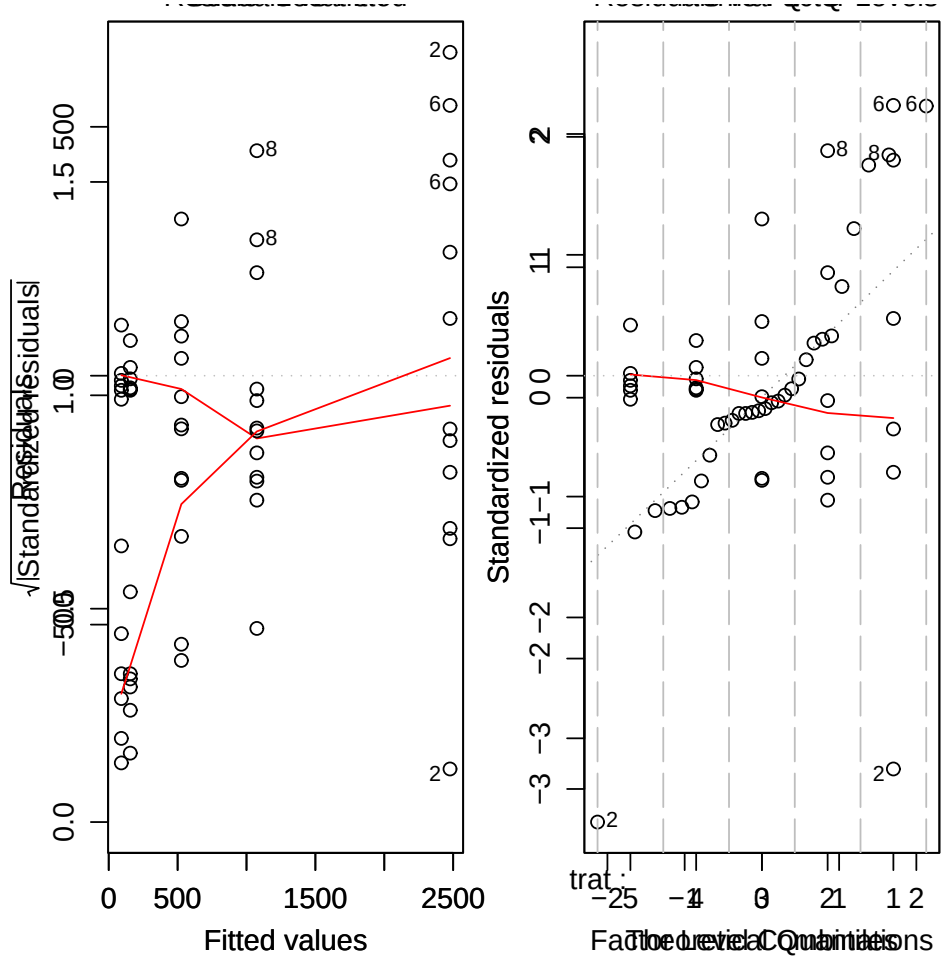



Figura 46: Gráficos de diagnóstico para dados originais

O gráfico de resíduos *vs* valores preditos mostra claramente uma heterogeneidade de variâncias e o $QQ - plot$ mostra um comportamento dos dados que se afasta muito da normal. A menssagem é clara mas podemos ainda fazer testes para verificar o desvio dos pressupostos.

```
> bartlett.test(tr$resp, tr$trat)

        Bartlett test of homogeneity of variances

data:  tr$resp and tr$trat
Bartlett's K-squared = 29.586, df = 4, p-value = 5.942e-06

> shapiro.test(tr.av$res)

        Shapiro-Wilk normality test

data:  tr.av$res
W = 0.8961, p-value = 0.006742
```

Nos resultados acima vemos que a homogeneidade de variâncias foi rejeitada.

Para tentar contornar o problema vamos usar a transformação Box-Cox, que consiste em transformar os dados de acordo com a expressão

$$y' = \frac{y^\lambda - 1}{\lambda},$$

onde $\lambda$ é um parâmeto a ser estimado dos dados. Se $\lambda = 0$ a equação acima se reduz a

$$y' = \log(y),$$

onde log é o logarítmo neperiano. Uma vez obtido o valor de $\lambda$ encontramos os valores dos dados transformados conforme a equação acima e utilizamos estes dados transformados para efetuar as análises.

A função `boxcox()` do pacote **MASS** calcula a verossimilhança perfilhada do parâmetro $\lambda$. Devemos escolher o valor que maximiza esta função. Nos comandos a seguir começamos carregando o pacote **MASS** e depois obtemos o gráfico da verossimilhança perfilhada. Como estamos interessados no máximo fazermos um novo gráfico com um *zoom* na região de interesse.

```
> require(MASS)
> boxcox(resp ~ trat, data = tr, plotit = T)
> boxcox(resp ~ trat, data = tr, lam = seq(-1, 1, 1/10))
```

O gráfico mostra que o valor que maximiza a função é aproximadamente $\hat{\lambda} = 0.1$. Desta forma o próximo passo é obter os dados transformados e depois fazer as análise utilizando estes novos dados.

```
> tr$respt <- (tr$resp^(0.1) - 1)/0.1
> tr.avt <- aov(respt ~ trat, data = tr)
> plot(tr.avt)
```

Note que os resíduos tem um comportamento bem melhor do que o observado para os dados originais. A análise deve prosseguir usando então os dados transformados.

**NOTA:** No gráfico da verossimilhança perfilhada notamos que é mostrado um intervalo de confiança para $\lambda$ e que o valor 0 está contido neste intervalo. Isto indica que podemos utilizar a transformação logarítimica dos dados e os resultados serão bom próximos dos obtidos com a transformação préviamente adotada.

```
> tr.avl <- aov(log(resp) ~ trat, data = tr)
> plot(tr.avl)
```

```
[1] TRUE
```

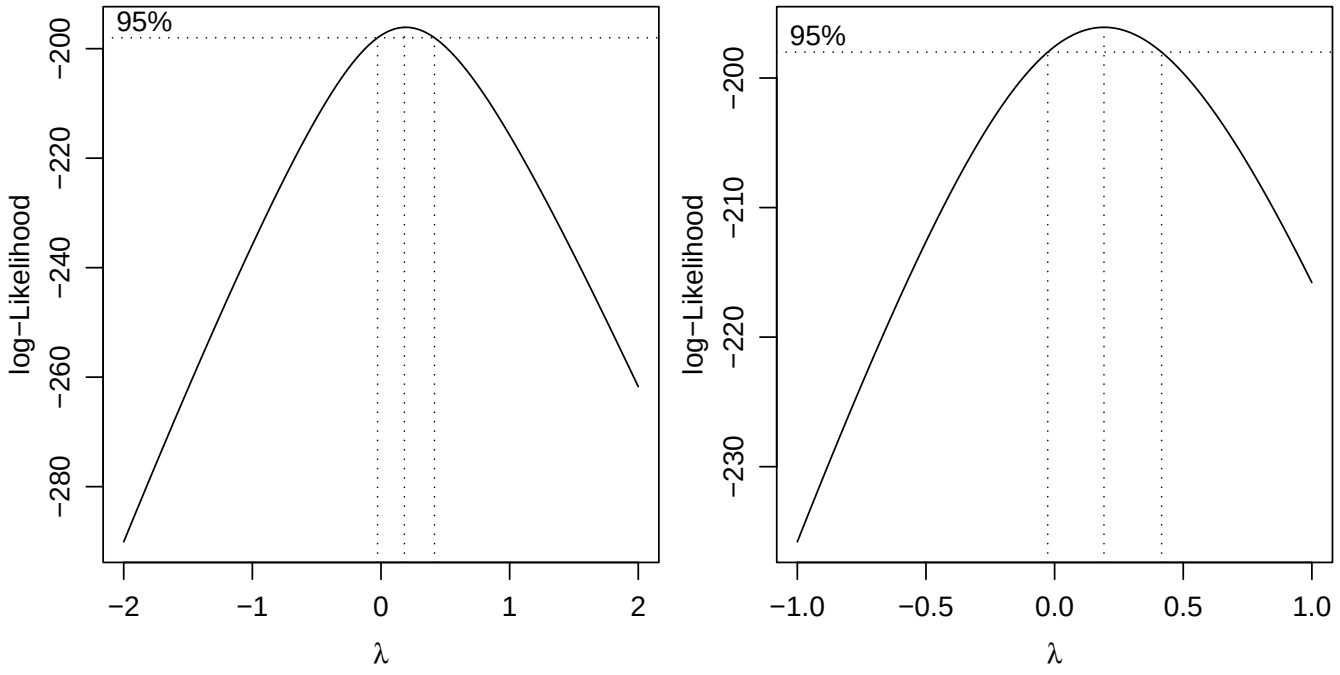



Figura 47: Perfis de verossimilhança para o parâmetro $\lambda$ da transformação Box-Cox

# 24 Fórmulas e especificação de modelos

Objetos do R podem ser separados entre *objetos de dados* (vetores, matrizes, arrays, data-frames e listas) e *outros tipos de objetos*. As fórmulas constituem um tipo especial de objeto no Rque representam simbolicamente relação entre variáveis e/ou objetos de dados. Fórmulas podem são em geral usadas em funções gráficas e funções que analisam dados a partir de algum *modelo* definido pela fórmula.

Nesta seção vamos fazer uma breve introdução ao uso de fórmulas através de alguns exemplos de análises de dados. Para isto iremos utilizar o conjunto de dados `mtcars` disponível com o R. Este conjunto contém características técnicas de diversos modelos da automóvel. Para carregar os dados e e listar os nomes das variáveis utilize os comandos a seguir. Lembre-se ainda que `help(mtcars)` irá fornecer mais detalhes sobre estes dados.

```
> data(mtcars)
> names(mtcars)
```

```
 [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
[11] "carb"
```

## 24.1 Fórmulas em gráficos

Algumas (mas não todas!) funções gráficas do R aceitam uma fórmula como argumento. Em geral tais funções exibem gráficos para explorar a relação entre variáveis. O R possui dois tipos de sistemas gráficos: (i) gráficos base (*base graphics*) e (ii) gráficos *lattice*. Os exemplos mostrados aqui se referem apenas ao primeiro sistema. Gráficos láttice são disponibilizados pelo pacote **lattice**. no qual as fórmulas são ainda mais importante e largamente utilizadas.

A Figura 24.1 mostra dois tipos de gráficos que são definidos a partir de fórmulas. No primeiro a variável de rendimento (*mpg*: milhas por galão) é relacionada com uma variável categórica (*cyl*:

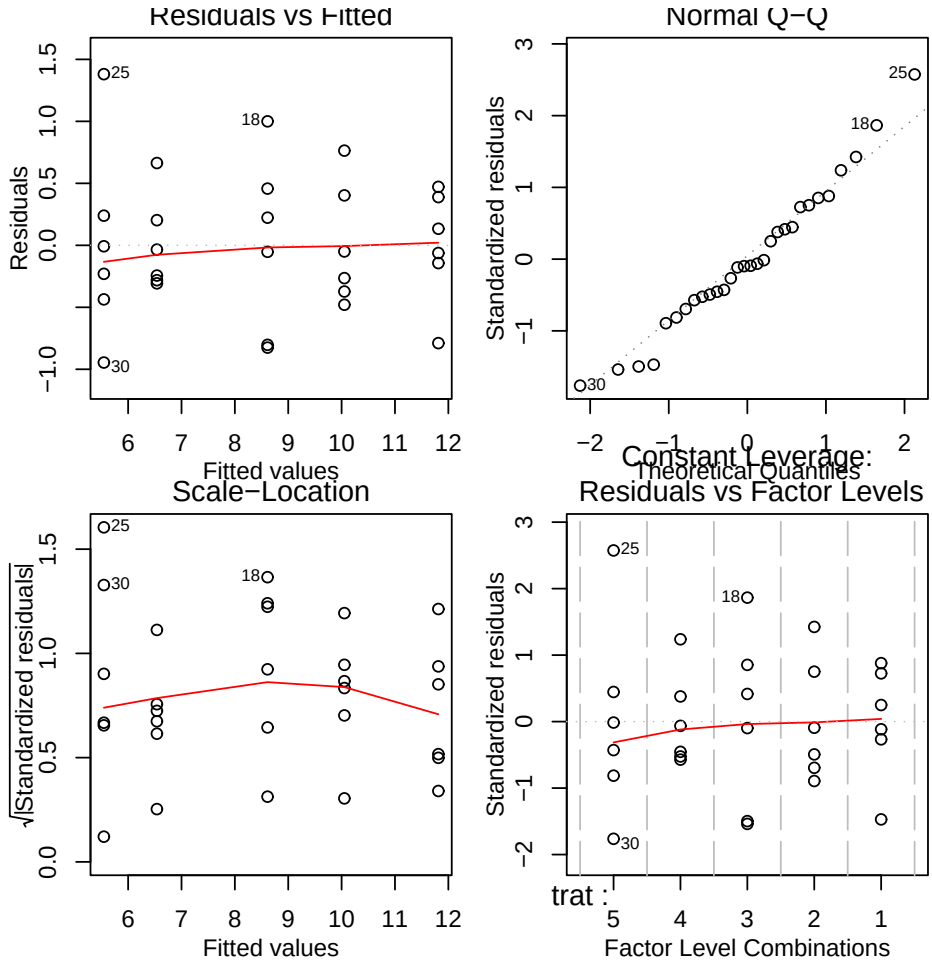



Figura 48: Gráficos de diagnóstico para dados transformados

número de cilindros). No segundo caso o rendimento é relacionado com o peso do veículo. A fórmula do tipo $y \sim x$ pode ser lida como: *a variável y é explicada por x*.

```
> with(mtcars, boxplot(mpg ~ cyl))
> with(mtcars, plot(mpg ~ cyl))
> with(mtcars, plot(mpg ~ wt))
```

A Figura 24.1 mostra agora um exemplo onde o gráfico de rendimento explicado pelo peso é feito para cada número de cilindros separadamente. Neste caso a formula usa o símbolo | para indicar condicionamento e é do tipo $y \sim x|A$ podendo ser lida como *a relação entre y e x para cada nível da variável A*.

```
> coplot(mpg ~ wt | as.factor(cyl), data = mtcars, panel = panel.smooth,
+     rows = 1)
```

## 24.2 Fórmulas em funções

Assim como no caso de gráficos, algums funções de análise de dados também aceitam fórmulas em seus argumentos. Considere o exemplo do texte-*t* para comparação de duas amostras na comparação do rendimento de veículos com cambio automático e manual. No exemplo a seguir mostramos o uso da função de duas formas que produzem resultados idênticos, uma sem usar fórmula e outra usando fórmula.

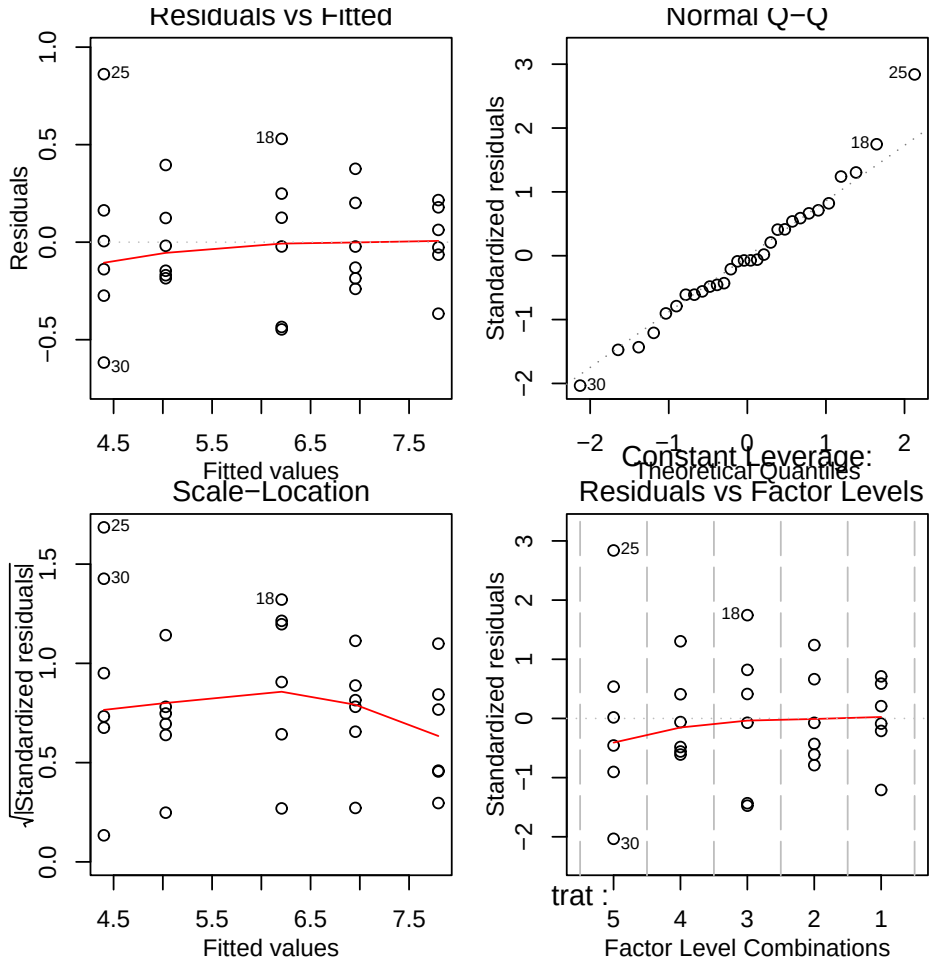



Figura 49: Gráficos de diagnóstico para dados com transformação logarítmica

```
> with(mtcars, t.test(mpg[am == 0], mpg[am == 1], var.eq = T))

        Two Sample t-test

data:  mpg[am == 0] and mpg[am == 1]
t = -4.1061, df = 30, p-value = 0.000285
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -10.848369  -3.641510
sample estimates:
mean of x mean of y
 17.14737  24.39231

> with(mtcars, t.test(mpg ~ am, var.eq = T))

        Two Sample t-test

data:  mpg by am
t = -4.1061, df = 30, p-value = 0.000285
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -10.848369  -3.641510
```

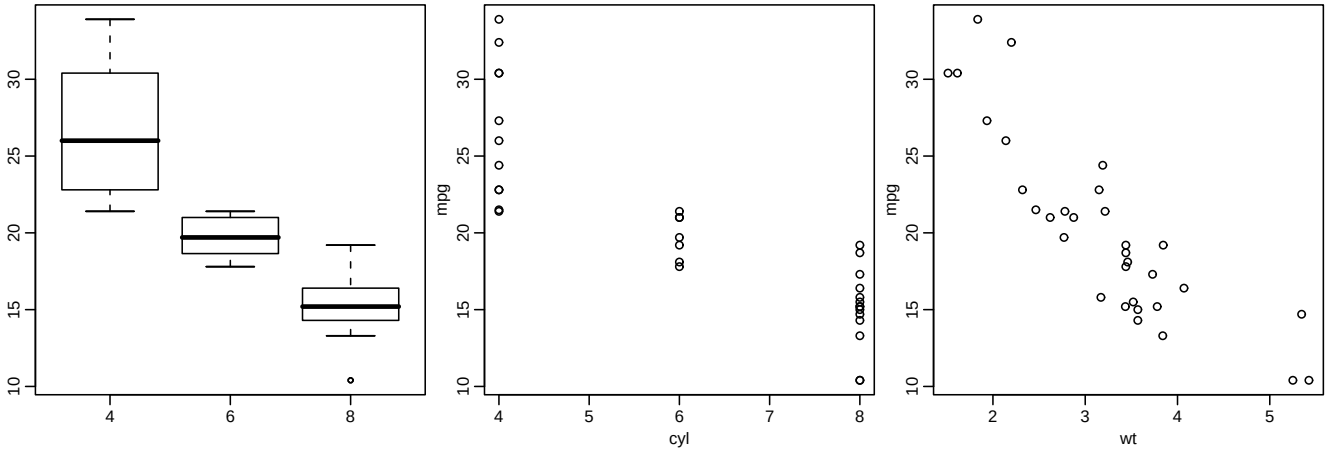



Figura 50: Exemplos de gráficos definidos através de fórmulas.

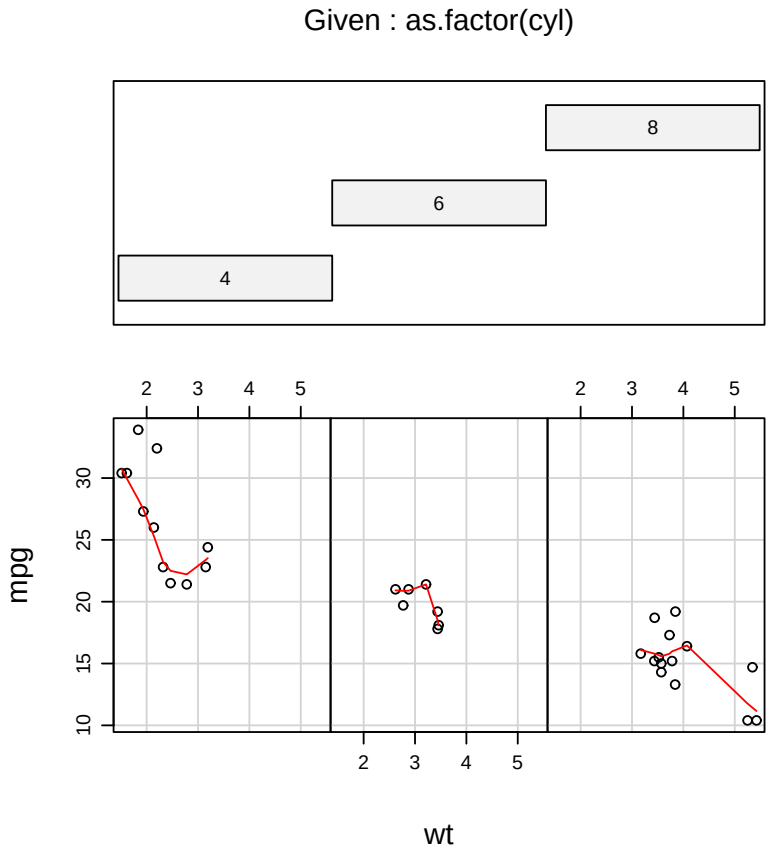



Figura 51: Gráfico obtido através de fórmula com termo condicional.

```
sample estimates:
mean in group 0 mean in group 1
       17.14737        24.39231
```

Portanto em $mpg \sim am$ pode-se ler: rendimento (mpg) explicado por tipo de câmbio (am). De forma similar a função para comparação de variâncias também pode utilizar fórmulas.

```
> with(mtcars, bartlett.test(mpg ~ am))

        Bartlett test of homogeneity of variances

data:  mpg by am
Bartlett's K-squared = 3.2259, df = 1, p-value = 0.07248
```

## 24.3 O objeto da classe `formula`

A fórmula é um objeto do R e possui a classe `formula`. Desta forma, funções que tenham métodos para esta classe tratam o objeto adequadamente. Por exemplo, no caso de `t.test` recebendo uma formula como argumento o método `formula` para `t.test` é disponível, como indica a documentação da função.

```
  ## S3 method for class 'formula':
  t.test(formula, data, subset, na.action, ...)
```

A seguir reforçamos estas idéias e vemos alguns comandos aplicados à manipulação de fórmulas. As funções `all.vars()` e `terms()` são particularmente úteis para manipulação de fórmulas o objetos dentro de funções.

```
> class(mpg ~ wt)

[1] "formula"

> form1 <- mpg ~ wt
> class(form1)

[1] "formula"

> all.vars(form1)

[1] "mpg" "wt"

> terms(form1)

mpg ~ wt
attr(,"variables")
list(mpg, wt)
attr(,"factors")
    wt
mpg  0
wt   1
attr(,"term.labels")
[1] "wt"
attr(,"order")
```

```
[1] 1
attr(,"intercept")
[1] 1
attr(,"response")
[1] 1
attr(,".Environment")
<environment: R_GlobalEnv>
```

## 24.4 Especificação de modelos com uma covariável

Entre os diversos usos de fórmulas, o mais importante deles é sem dúvida o fato que fórmulas são utimizadas na declaração de modelos estatísticos. Um aspecto particularmente importante da linguagem S, o portanto no programa R, é que adota-se uma abordagem unificada para modelagem, o que inclui a sintaxe para especificação de modelos. Variáveis *respostas* e *covariáveis* (variáveis explanatórias) são sempre especificadas de mesma forma básica, ou seja, na forma *resposta* $\sim$ *covariavel*, onde:

- à esquerda indica-se a(s) variável(eis) resposta
- o símbolo $\sim$ significa *é modelada por*
- à direita indica-se a(s) covariável(eis)

No restante deste texto vamos, por simplicidade, considerar que há apenas uma variável resposta que poderá ser explicada por uma ou mais covariáveis.

Considere, para o conjunto de dados `mtcars`, ajustar um modelo que explique o rendimento (Y:mpg) pelo peso do veículo (X:wt). O modelo linear é dado por:

$$Y = \beta_0 + \beta_1 X + \epsilon \, ,$$

e pode ser ajustado no R usando `lm()` (*lm : linear model*). Na sintaxe da chamada função *mpg* $\sim$ *wt* lê-se: *mpg é modelado por wt*, através de um modelo linear `lm()`, o que implica no modelo acima. A Figura 52 mostra os dados e a linha sólida mostra a equação do modelo ajustado.

```
> reg1 <- lm(mpg ~ wt, data = mtcars)
> reg1

Call:
lm(formula = mpg ~ wt, data = mtcars)

Coefficients:
(Intercept)           wt
     37.285       -5.344
```

Note que a fórmula apenas especifica a relação entre as variáveis resposta e explanatórias e **não** implica que o modelo seja necessariamente linear. A linearidade é dada pela função `lm()`. Portanto a mesma fórmula pode ser usada para outros tipos de ajuste como o mostrado na linha tracejada do gráfico resultantes de regressão local polinomial obtida por `loess()`).

Nem todas as funções que relacionam variáveis aceitam formulas, como por exemplo o caso da regressão por núcleo (*kernel*) dada por `ksmooth()` cujo o ajuste é mostrado na linha pontilhada. Outras funções extendem a notação de funções como é o caso do ajuste por *modelos aditivos generalizados* `gam()` mostrado na linha sólida grossa, onde o termo `s()` indica que a variável resposta deve ser descrita por uma função *suave* da covariável incluída neste termo.

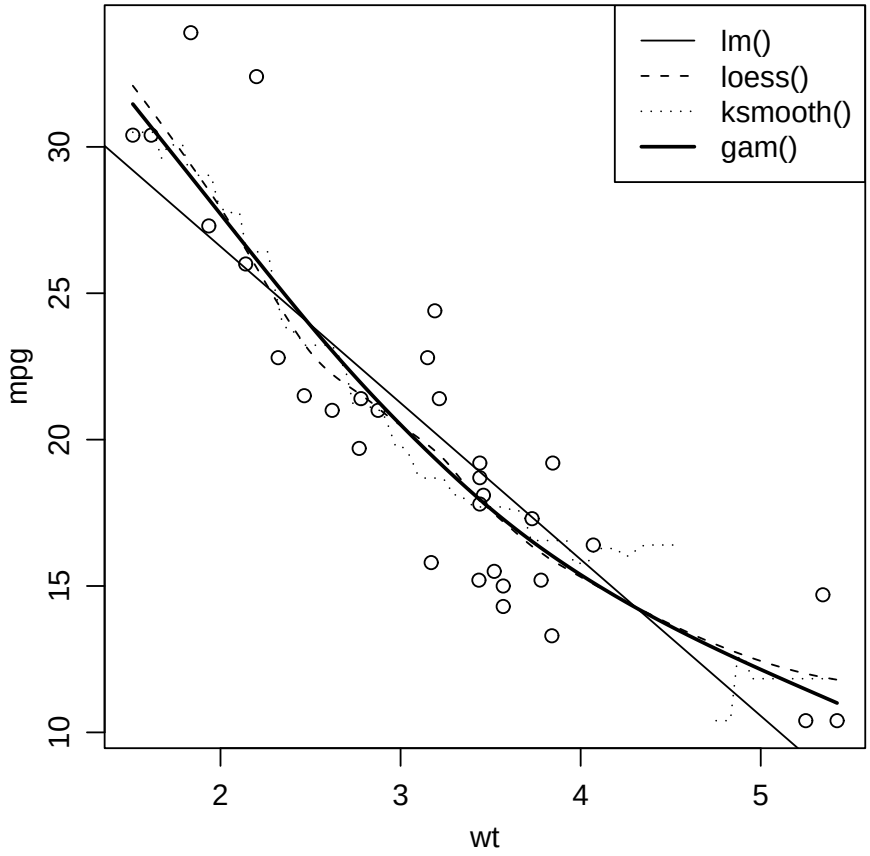



Figura 52: Diferentes modelos ajustados para descrever a relação entre duas variáveis quantitativas.

```
> with(mtcars, plot(mpg ~ wt))
> abline(reg1)
> reg2 <- loess(mpg ~ wt, data = mtcars)
> wts <- with(mtcars, seq(min(wt), max(wt), len = 201))
> lines(wts, predict(reg2, data.frame(wt = wts)), lty = 2)
> lines(with(mtcars, ksmooth(wt, mpg, band = 1)), lty = 3)
> require(mgcv)
> reg3 <- gam(mpg ~ s(wt), data = mtcars)
> lines(wts, predict(reg3, data.frame(wt = wts)), lwd = 2)
> legend("topright", c("lm()", "loess()", "ksmooth()", "gam()"),
+     lty = c(1:3, 1), lwd = c(1, 1, 1, 2))
```

Nos exemplos acima é interessante notar o uso de `predict()` que é utilizada para predizer o valor da resposta para um conjunto arbitrário de valores da covariável, baseando-se no modelo ajustado. No exemplo utilizamos este recurso para produzir o gráfico com a "curva"do modelo ajustado para uma sequência de valores da covariável. Para a função `lm()` utilizamos apenas `abline()` devido ao fato que esta função retorna a equação de uma reta que é interpretada a traçada por um método `abline`. Entretanto `predict()` também poderia ser usada e a reta traçada com o comando a seguir. Esta forma é mais flexível para traçar funções (modelos) ajustados que sejam mais complexos que uma equação de uma reta.

```
> lines(wts, predict(reg1, data.frame(wt = wts)))
```

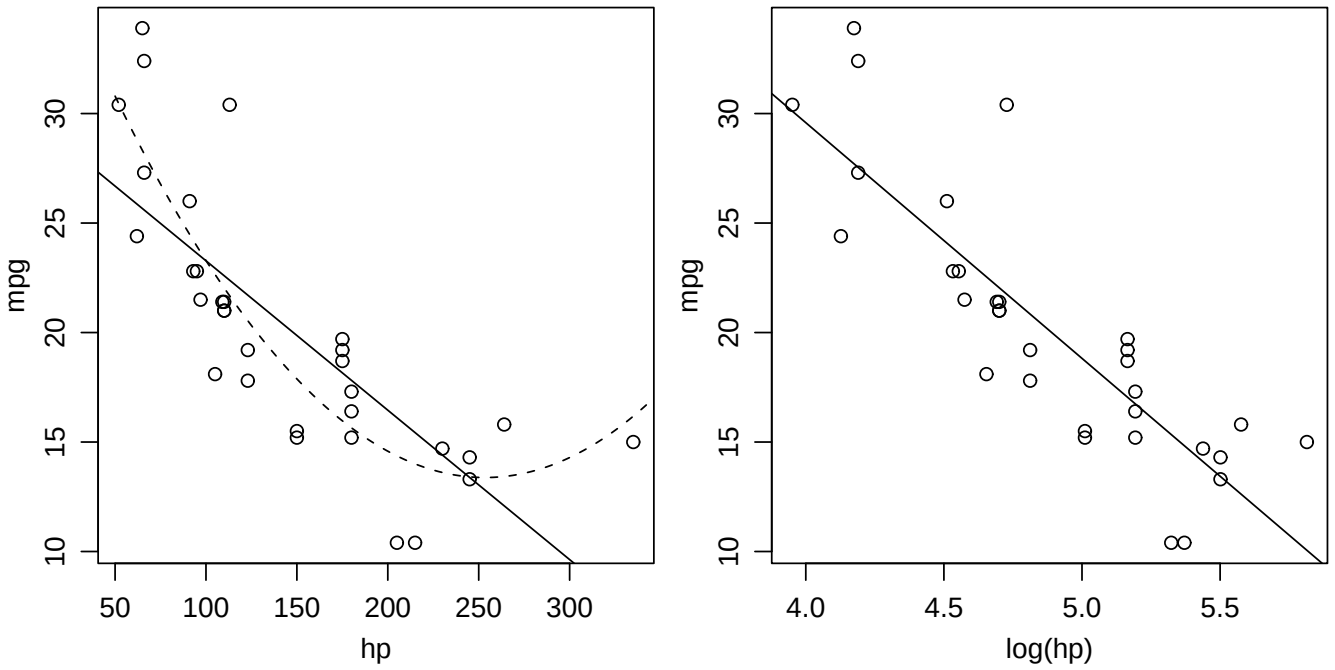



Figura 53: Ilustração do uso de operadores aritméticos e funções polinomiais na especificação de fórmulas.

## 24.5 Extensões de modelos com uma covariável

As formulas admitem operadores aritméticos em seus termos. Por exemplo considere a relação entre o rendimento (*mgp*) e a potência (*hp*). A linha sólida no gráfico da esquerda da Figura 24.5 sugere que o modelo linear não descreve bem a relação entre estas variáveis, enquanto no gráfico da direita sugere a relação é melhor descrita por um modelo linear entre o rendimento e o logarítmo de potência. Na chamada das funções utilizamos a operação aritmética `log()` diretamente na fórmula, sem a necessidade de transformar os dados originais.

```
> with(mtcars, plot(mpg ~ hp))
> abline(lm(mpg ~ hp, data = mtcars))
> with(mtcars, plot(mpg ~ log(hp)))
> abline(lm(mpg ~ log(hp), data = mtcars))
```

Uma outra possibilidade para os dados originais é o ajuste de um modelo dado por uma função polinomial, conforme mostrado na linha tracejada no gráfico da esquerda da Figura 24.5 e que é ajustado com os comandos a seguir. Neste ajuste é importante notar que a variável quadrática **deve** ser especificada com `I(hp^2)` e o uso de `I()` é obrigatório para garantir que os valores de `hp` sejam de fato elevados ao quadrado. O uso de `hp^2` possui um significado diferente que veremos na próxima sessão.

```
> polA <- lm(mpg ~ hp + I(hp^2), data = mtcars)
> hps <- seq(50, 350, len = 200)
> lines(hps, predict(polA, data.frame(hp = hps)), lty = 2)
```

Uma outra forma de especificar regressões polinomiais é com o uso de `poly()`, onde o grau do desejado do polinômio é um argumento desta função conforme ilustrado nos comandos a seguir. É importante notar que a interpretação dos parâmetros é diferente devido ao fato que polinômios ortogonais são utilizados, entretanto os valores preditos e as estatísticas de ajuste são iguais. O

ajuste por polinômios ortogonais é numericamente mais estável e portanto deve ser preferido quando possível.

```
> polA

Call:
lm(formula = mpg ~ hp + I(hp^2), data = mtcars)

Coefficients:
(Intercept)           hp      I(hp^2)
 40.4091172   -0.2133083    0.0004208

> polB <- lm(mpg ~ poly(hp, 2), data = mtcars)
> polB

Call:
lm(formula = mpg ~ poly(hp, 2), data = mtcars)

Coefficients:
 (Intercept)  poly(hp, 2)1  poly(hp, 2)2
       20.09        -26.05         13.15

> hps <- seq(50, 350, by = 50)
> predict(polA, data.frame(hp = hps))

       1        2        3        4        5        6        7
30.79574 23.28645 17.88123 14.58009 13.38303 14.29005 17.30114

> predict(polB, data.frame(hp = hps))

       1        2        3        4        5        6        7
30.79574 23.28645 17.88123 14.58009 13.38303 14.29005 17.30114
```

Vamos considerar agora um outro exemplo de ajuste de modelo linear, agora para o conjunto de dados `women` que fornece peso (*weight*) em libras (lbs) e altura (*height*) em polegadas (in) de 15 mulheres americanas de 30-39 anos. Os comandos a seguir mostram os quatro ajustes indicados na Figura 54. O primeiro (linha fina sólida) é uma regressão linear, o segundo (linha fina tracejada) é uma regressão linear com intercepto igual a zero, isto é, a reta passa pela origem. O terceiro (linha sólida grossa) é uma regressão quadrática e o quarto (linha sólida grossa) é uma regressão quadrática passando pela origem. Neste exemplo fica então ilustrado que a adição do termo `+ 0` na fórmula faz com que o intercepto do modelo seja nulo e apenas o parâmetro referente ao coeficiente angular da reta seja estimado.

```
> data(women)
> wm1 <- lm(weight ~ height, data = women)
> wm2 <- lm(weight ~ height + 0, data = women)
> wm3 <- lm(weight ~ height + I(height^2), data = women)
> wm4 <- lm(weight ~ height + I(height^2) + 0, data = women)
> with(women, plot(weight ~ height))
> hgs <- seq(58, 72, l = 200)
> lines(hgs, predict(wm1, data.frame(height = hgs)))
> lines(hgs, predict(wm2, data.frame(height = hgs)), lty = 2)
```

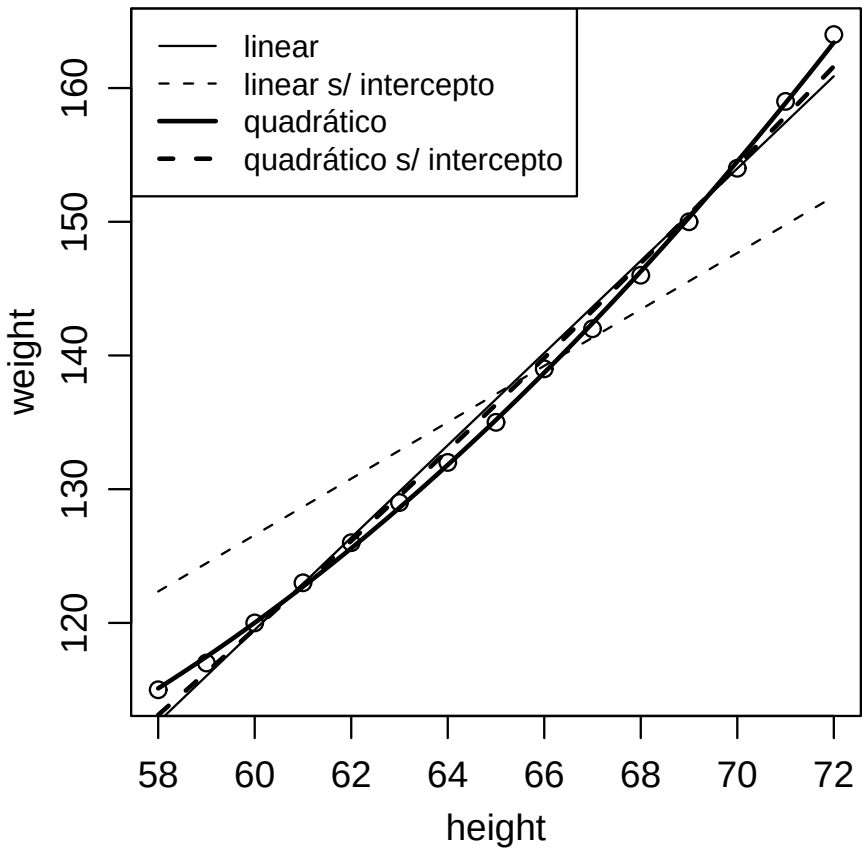



Figura 54: Ajustes de modelos de primeiro e segundo grau, com e sem estimação do intercepto.

```
> lines(hgs, predict(wm3, data.frame(height = hgs)), lwd = 2)
> lines(hgs, predict(wm4, data.frame(height = hgs)), lty = 2, lwd = 2)
> legend("topleft", c("linear", "linear s/ intercepto", "quadrático",
+     "quadrático s/ intercepto"), lty = c(1, 2, 1, 2), lwd = c(1,
+     1, 2, 2), cex = 0.85)
```

## 24.6 Especificações mais gerais de modelos

Nos exemplos anteriores a variável resposta era explicada por apenas uma variável explanatória. Isto pode ser expandido considerando-se a presença de duas ou mais variáveis explicativas. A Tabela 24.6 resume as principais operações possíveis para definir modelos com uma ou duas variáveis e que podem ser extendidas para o caso de mais variáveis.

Esta notação é uma implementação das idéias propostas por Wilkinson e Rogers para especificação de modelos estatísticos. (G. N. Wilkinson. C. E. Rogers. Symbolic Description of Factorial Models for Analysis of Variance. *Applied Statistics*, Vol. 22, No. 3, 392-399. 1973).

Para ilustrar algumas destas opções vamos considerar novamente o conjunto de dados `mtcars` ajustando modelos para o rendimento (*mpg*) explicado pelo peso (*wt*) e potência (*hp*) do veículo. Nos comandos a seguir mostramos os coeficientes estimados a partir de cinco formas de especificação de modelos.

```
> coef(lm(mpg ~ I(wt + hp), data = mtcars))
```

```
(Intercept)  I(wt + hp)
 30.2877307  -0.0680239
```

Tabela 5: Sintaxe para especificação de termos dos modelos

| Termos | Especificação |
|---|---|
| `A + B` | Efeitos principais $A$ e $B$ |
| `A : B` | Termo de interação entre $A$ e $B$ |
| `A * B` | Efeitos principais e interação, corresponde a $A + B + A : B$ |
| `B \%in\% A` | $B$ dentro (aninhado) de $A$ |
| `A/B` | Efeito principal e aninhado, corresponde a $A + B\%in\%A$ |
| `A-B` | tudo de $A$ exceto o que está em $B$ |
| `A^k` | Todos os termos de $A$ e interação de ordem $k$ |
| `A + 0` | exclui o intercepto de modelo |
| `I()` | operador de identidade aritmética, ver explicação no texto |

```
> coef(lm(mpg ~ wt + hp, data = mtcars))

(Intercept)          wt          hp
37.22727012 -3.87783074 -0.03177295

> coef(lm(mpg ~ I(wt * hp), data = mtcars))

(Intercept)  I(wt * hp)
27.74564216 -0.01487156

> coef(lm(mpg ~ wt * hp, data = mtcars))

(Intercept)          wt          hp       wt:hp
49.80842343 -8.21662430 -0.12010209  0.02784815

> coef(lm(mpg ~ (wt + hp)^2, data = mtcars))

(Intercept)          wt          hp       wt:hp
49.80842343 -8.21662430 -0.12010209  0.02784815

> coef(lm(mpg ~ I((wt + hp)^2), data = mtcars))

   (Intercept) I((wt + hp)^2)
 24.4985252043  -0.0001625815
```

Os resultados sugerem que as fórmulas definem modelos diferentes, exceto pelos termos `wt * hp` e `(wt * hp)^2` onde o mesmo modelo é especificado de duas formas alternativas. Os modelos ajustados para explicar o rendimento *mpg* denotado por $Y$ são:

- 1. $Y = \beta_0 + \beta_1 X_1 + epsilon$, um modelo com apenas uma covariável onde $X_1$ é a covariável única com valores dados pela soma dos valores de *wt* e *hp* de cada veículo;
- 2. $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + epsilon$, um modelo com duas covariáveis onde $X_1$ é a covariável *wt* e $X_2$ é a covariável *hp*.
- 3. $Y = \beta_0 + \beta_1 X_1 + epsilon$, um modelo com apenas uma covariável onde $X_1$ é a covariável única com valores dados pelo produto dos valores de *wt* e *hp* de cada veículo;

Tabela 6: Outros exemplos de sintaxe para especificação de modelos.

| Declaração | Modelo equivalente | Descrição |
|---|---|---|
| `A+B*C` | A+B+C+B:C | todos efeitos principais e interação dupla apenas entre $B$ e |
| `A+B*(C+D)` | `A+B+C+D+B:C+B:D` | todos efeitos principais e interações duplas de $B$ com $C$ e |
| `A*B*C` | `A+B+C+A:B+A:C+B:C+A:B:C` | todos efeitos pricipais e interações possíveis |
| `(A+B+C)^3` | `A+B+C+A:B+A:C+B:C+A:B:C` | três covariáveis e interações de ordem 2 e 3 (igual ao anter |
| `(A+B+C)^2` | `A+B+C+A:B+A:C+B:C` | três covariáveis e interações de ordem 2 |
| `(A+B+C)^3 - A:B:C` | `(A+B+C)^2` | três covariáveis e interações de ordem 2 |
| `(A+B+C)^2 - A:C` | `A+B+C+A:B+B:C` | três covariáveis e interações de ordem 2, exceto por $A:C$ |
| `A+I(A^2)+B` | `poly(A,2)+B` | termos lineares em $A$ e $B$ e quadrático em $A$ (*) |

- 4. e 5. $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + epsilon$, um modelo com duas covariáveis mais o termo de interação entre elas, onde $X_1$ é a covariável *wt*, $X_2$ é a covariável *hp* e $X_3$ é a interação dada pelo produto $X_3 = X_1 \times X_2$.

- 6. $Y = \beta_0 + \beta_1 X_1 + epsilon$, um modelo com apenas uma covariável onde $X_1$ é a covariável única com valores dados pelo quadrado da soma dos valores de *wt* e *hp* de cada veículo;

Chama-se atenção ao fato que a notação de "potência"em `(wt+hp)^2` **não** indica uma operação aritmética mas sim a *inclusão de todos os efeitos principais e interações até as de ordem indicada pela potência*. Para incluir a operação aritmética de potência é necessário utilizar `I()` no termo a ser exponenciado.

De forma geral, a mensagem é de que os operadores *soma* (`+`), *produto* (`*`), *divisão* (`/`) e *potência* (`^`) têm nas fórmulas o papel de definir *numa notação símbólica* quais e como os termos devem ser incluídos no modelo. Em fórmulas, tais operadores só indicam operações aritméticas com os termos envolvidos quando utilizados dentro de `I()`. A função `I()` garantre que a expressão nela contida seja avaliada como uma função aritmética, tal e qual está escrita (*"as is"*).

Na tabela 24.6 são ilustradas mais algumas especificações de modelos. No caso marcado com (*) os modelos são equivalentes porém os coeficientes resultantes são diferentes como comentado sobre polinômios ortogonais na Sessão 24.5.

## 24.7 Atualizando e modificando fórmulas

Uma vez que um objeto contenha uma fórmula, é possível obter uma nova fórmula que seja uma modificação da original utilizando `update.formula()`.

```
> form1 <- y ~ x1 + x2 + x3
> form1

y ~ x1 + x2 + x3

> form2 <- update.formula(form1, . ~ . - x2)
> form2

y ~ x1 + x3
```

A lógica da sintaxe é que o primeiro argumento recebe uma fórmula inicial e o segundo indica a modificação. O caracter ponto ($\cdot$) indica *tudo*. Ou seja, em `. ~ . - x2` entende-se: a nova fórmula deverá possuir tudo que estava do lado esquerdo, e tudo do lado direito, excluindo a variável `x2`.

Este mecanismos é útil para evitar que fórmulas precisem ser totalmente redigitadas a cada redefinição do modelo, o que é útil ao se investigar vários modelos que são obtidos uns a partir de outros. O mecanismo também reduz a chance de erros nas especificações uma vez que garante a igualdade daquilo que é indicado pelo ponto $(\cdot)$.

# 25 Experimentos em esquema fatorial

O experimento fatorial descrito na apostila do curso de Planejamento de Experimentos II comparou o crescimento de mudas de eucalipto considerando diferentes recipientes e espécies.

1. **Lendo os dados**

   Vamos considerar agora que os dados já estejam digitados em um arquivo texto. Clique aqui para ver e copiar o arquivo com conjunto de dados para o seu diretório de trabalho.

   A seguir vamos ler (importar) os dados para R com o comando `read.table`:

```
> ex04 <- read.table("exemplo04.txt", header=T)
> ex04
```

   Antes de começar as análise vamos inspecionar o objeto que contém os dados para saber quantas observações e variáveis há no arquivo, bem como o nome das variáveis. Vamos também pedir o R que exiba um rápido resumo dos dados.

```
> dim(ex04)
[1] 24  3

> names(ex04)
[1] "rec"  "esp"  "resp"

> attach(ex04)

> is.factor(rec)
[1] TRUE
> is.factor(esp)
[1] TRUE
> is.factor(resp)
[1] FALSE
> is.numeric(resp)
[1] TRUE
```

   Nos resultados acima vemos que o objeto `ex04` que contém os dados tem 24 linhas (observações) e 3 colunas (variáveis). As variáveis tem nomes `rec`, `esp` e resp, sendo que as duas primeiras são *fatores* enquanto `resp` é uma variável numérica, que neste caso é a variável resposta. O objeto `ex04` foi incluído no caminho de procura usando o comando `attach` para facilitar a digitação.

2. **Análise exploratória**

   Inicialmente vamos obter um resumo de nosso conjunto de dados usando a função `summary`.

```
> summary(ex04)
 rec    esp          resp
 r1:8   e1:12   Min.   :18.60
 r2:8   e2:12   1st Qu.:19.75
 r3:8           Median :23.70
                Mean   :22.97
                3rd Qu.:25.48
                Max.   :26.70
```

Note que para os fatores são exibidos o número de dados em cada nível do fator. Já para a variável numérica são mostrados algumas medidas estatísticas. Vamos explorar um pouco mais os dados

```
> ex04.m <- tapply(resp, list(rec,esp), mean)
> ex04.m
       e1     e2
r1 25.650 25.325
r2 25.875 19.575
r3 20.050 21.325

> ex04.mr <- tapply(resp, rec, mean)
> ex04.mr
     r1      r2      r3
25.4875 22.7250 20.6875

> ex04.me <- tapply(resp, esp, mean)
> ex04.me
      e1       e2
23.85833 22.07500
```

Nos comandos acima calculamos as médias para cada fator, assim como para os cruzamentos entre os fatores. Note que podemos calcular outros resumos além da média. Experimente nos comandos acima substituir `mean` por `var` para calcular a variância de cada grupo, e por `summary` para obter um outro resumo dos dados.

Em experimentos fatoriais é importante verificar se existe interação entre os fatores. Inicialmente vamos fazer isto graficamente e mais a frente faremos um teste formal para presença de interação. Os comandos a seguir são usados para produzir os gráficos exibidos na Figura 55.

```
> par(mfrow=c(1,2))
> interaction.plot(rec, esp, resp)
> interaction.plot(esp, rec, resp)
```

3. **Análise de variância**

Seguindo o modelo adequado, o análise de variância para este experimento inteiramente casualizado em esquema fatorial pode ser obtida com o comando:

```
> ex04.av <- aov(resp ~ rec + esp + rec * esp)
```

Entretanto o comando acima pode ser simplificado produzindo os mesmos resultados com o comando

```
> ex04.av <- aov(resp ~ rec * esp)
> summary(ex04.av)
            Df Sum Sq Mean Sq F value    Pr(>F)
rec          2 92.861  46.430  36.195 4.924e-07 ***
esp          1 19.082  19.082  14.875  0.001155 **
rec:esp      2 63.761  31.880  24.853 6.635e-06 ***
Residuals   18 23.090   1.283
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1
```

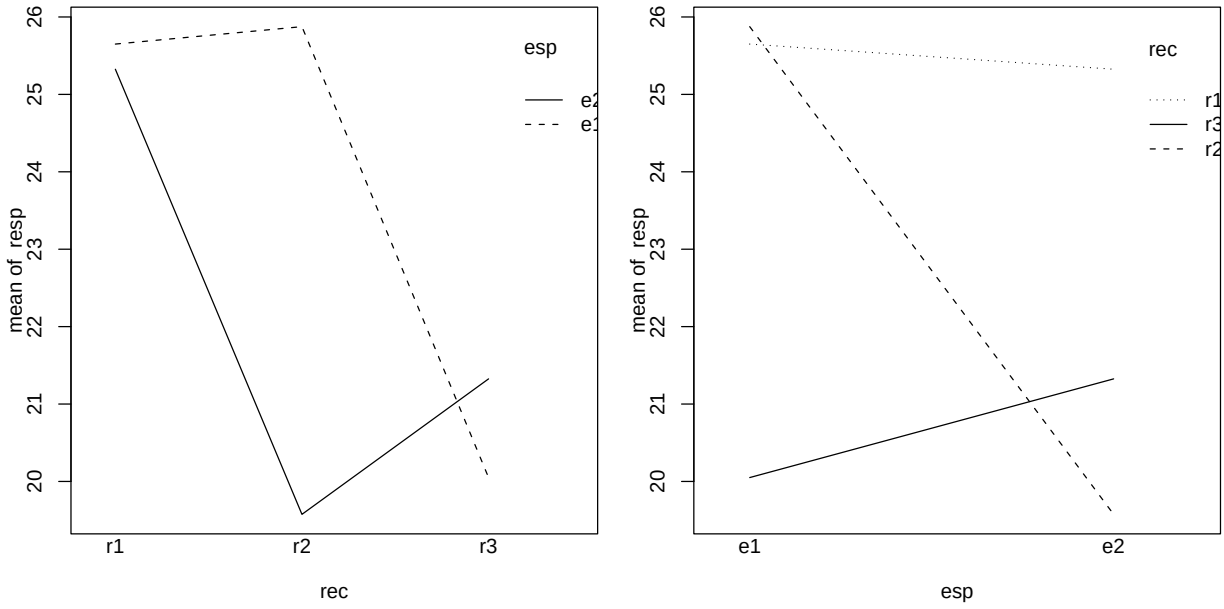



Figura 55: Gráficos de interação entre os fatores.

Isto significa que no R, ao colocar uma interação no modelo, os efeitos principais são incluídos automaticamente. Note no quadro de análise de variância que a interação é denotada por `rec:esp`. A análise acima ostra que este efeito é significativo, confirmando o que verificamos nos gráficos de interação vistos anteriormente.

O objeto `ex04.av` guarda todos os resultados da análise e pode ser explorado por diversos comandos. Por exemplo a função `model.tables` aplicada a este objeto produz tabelas das médias definidas pelo modelo. O resultado mostra a média geral, médias de cada nível fatores e das combinações dos níveis dos fatores. Note que no resultado está incluído também o número de dados que gerou cada média.

```
> ex04.mt <- model.tables(ex04.av, ty="means")
> ex04.mt
Tables of means
Grand mean

22.96667

 rec
       r1    r2    r3
    25.49 22.73 20.69
rep  8.00  8.00  8.00

 esp
       e1    e2
    23.86 22.07
rep 12.00 12.00

 rec:esp
     esp
rec    e1     e2
  r1  25.650 25.325
```

```
  rep  4.000  4.000
  r2  25.875 19.575
  rep  4.000  4.000
  r3  20.050 21.325
  rep  4.000  4.000
```

Mas isto não é tudo! O objeto `ex04.av` possui vários elementos que guardam informações sobre o ajuste.

```
> names(ex04.av)
 [1] "coefficients"  "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residual"
 [9] "contrasts"     "xlevels"       "call"          "terms"
[13] "model"

> class(ex04.av)
[1] "aov" "lm"
```

O comando `class` mostra que o objeto `ex04.av` pertence às classes `aov` e `lm`. Isto significa que devem haver *métodos* associados a este objeto que tornam a exploração do resultado mais fácil. Na verdade já usamos este fato acima quando digitamos o comando `summary(ex04.av)`. Existe uma função chamada `summary.aov` que foi utilizada já que o objeto é da classe `aov`. Iremos usar mais este mecanismo no próximo passo da análise.

4. **Análise de resíduos**

Após ajustar o modelo devemos proceder a análise dos resíduos para verificar os pressupostos. O R produz automaticamente 4 gráficos básicos de resíduos conforme a Figura 56 com o comando `plot`.

```
> par(mfrow=c(2,2))
> plot(ex04.av)
```

Os gráficos permitem uma análise dos resíduos que auxiliam no julgamento da adequacidade do modelo. Evidentemente voce não precisa se limitar os gráficos produzidos automaticamente pelo R – voce pode criar os seus próprios gráficos muito facilmente. Neste gráficos voce pode usar outras variáveis, mudar texto de eixos e títulos, etc, etc, etc. Examine os comandos abaixo e os gráficos por eles produzidos.

```
> par(mfrow=c(2,1))
> residuos <- resid(ex04.av)

> plot(ex04$rec, residuos)
> title("Resíduos vs Recipientes")

> plot(ex04$esp, residuos)
> title("Resíduos vs Espécies")

> par(mfrow=c(2,2))
> preditos <- (ex04.av$fitted.values)
> plot(residuos, preditos)
```

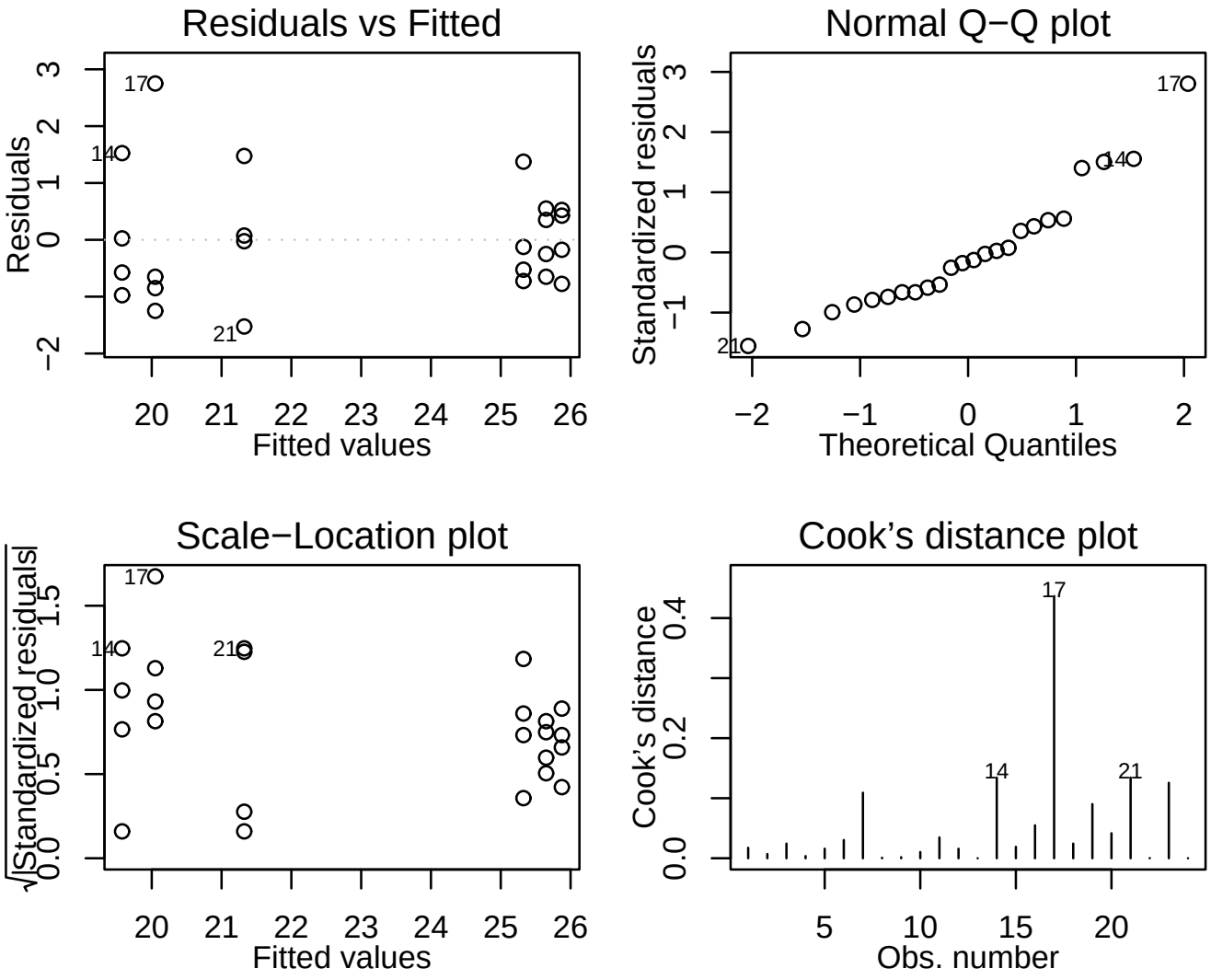



Figura 56: Gráficos de resíduos produzidos automaticamente pelo R.

```
> title("Resíduos vs Preditos")
> s2 <- sum(resid(ex04.av)^2)/ex04.av$df.res
> respad <- residuos/sqrt(s2)
> boxplot(respad)
> title("Resíduos Padronizados")
> qqnorm(residuos,ylab="Residuos", main=NULL)
> qqline(residuos)
> title("Grafico Normal de \n Probabilidade dos Resíduos")
```

Além disto há alguns testes já programados. Como exemplo vejamos e teste de Shapiro-Wilk para testar a normalidade dos resíduos.

```
> shapiro.test(residuos)

        Shapiro-Wilk normality test

data:  residuos
W = 0.9293, p-value = 0.09402
```

5. **Desdobrando interações**

   Conforma visto na apostila do curso, quando a interação entre os fatores é significativa podemos desdobrar os graus de liberdade de um fator dentro de cada nível do outro. A forma de fazer isto no R é reajustar o modelo utilizando a notação `/` que indica efeitos aninhados. Desta forma podemos desdobrar os efeitos de espécie dentro de cada recipiente e vice versa conforme mostrado a seguir.

```
> ex04.avr <- aov(resp ~ rec/esp)
> summary(ex04.avr, split=list("rec:esp"=list(r1=1, r2=2)))
              Df Sum Sq Mean Sq F value    Pr(>F)
rec            2 92.861  46.430 36.1952 4.924e-07 ***
rec:esp        3 82.842  27.614 21.5269 3.509e-06 ***
  rec:esp: r1  1  0.211   0.211  0.1647    0.6897
  rec:esp: r2  1 79.380  79.380 61.8813 3.112e-07 ***
  rec:esp: r3  1  3.251   3.251  2.5345    0.1288
Residuals     18 23.090   1.283
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1


> ex04.ave <- aov(resp ~ esp/rec)
> summary(ex04.ave, split=list("esp:rec"=list(e1=c(1,3), e2=c(2,4))))
              Df  Sum Sq Mean Sq F value    Pr(>F)
esp            1  19.082  19.082  14.875  0.001155 **
esp:rec        4 156.622  39.155  30.524 8.438e-08 ***
  esp:rec: e1  2  87.122  43.561  33.958 7.776e-07 ***
  esp:rec: e2  2  69.500  34.750  27.090 3.730e-06 ***
Residuals     18  23.090   1.283
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1
```

6. **Teste de Tukey para comparações múltiplas**

   Há vários testes de comparações múltiplas disponíveis na literatura, e muitos deles implementados no R. Os que não estão implementados podem ser facilmente calculados utilizando os recursos do R.

   Vejamos por exemplo duas formas de usar o *Teste de Tukey*, a primeira usando uma implementação com a função `TukeyHSD` e uma segunda fazendo ops cálculos necessários com o R.

   Poderíamos simplesmente digitar:

```
> ex04.tk <- TukeyHSD(ex04.av)
> plot(ex04.tk)
> ex04.tk
```

   e obter diversos resultados. Entretanto nem todos nos interessam. Como a interação foi significativa na análise deste experimento a comparação dos níveis fatores principais não nos interessa.

   Podemos então pedir a função que somente mostre a comparação de médias entre as combinações dos nívies dos fatores.

```
> ex04.tk <- TukeyHSD(ex04.ave, "esp:rec")
> plot(ex04.tk)
> ex04.tk
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = resp ~ esp/rec)

$"esp:rec"
        diff        lwr       upr
 [1,] -0.325 -2.8701851  2.220185
 [2,]  0.225 -2.3201851  2.770185
 [3,] -6.075 -8.6201851 -3.529815
 [4,] -5.600 -8.1451851 -3.054815
 [5,] -4.325 -6.8701851 -1.779815
 [6,]  0.550 -1.9951851  3.095185
 [7,] -5.750 -8.2951851 -3.204815
 [8,] -5.275 -7.8201851 -2.729815
 [9,] -4.000 -6.5451851 -1.454815
[10,] -6.300 -8.8451851 -3.754815
[11,] -5.825 -8.3701851 -3.279815
[12,] -4.550 -7.0951851 -2.004815
[13,]  0.475 -2.0701851  3.020185
[14,]  1.750 -0.7951851  4.295185
[15,]  1.275 -1.2701851  3.820185
```

Mas ainda assim temos resultados que não interessam. Mais especificamente estamos intessados nas comparações dos níveis de um fator dentro dos nívies de outro. Por exemplo, vamos fazer as comparações dos recipientes para cada uma das espécies.

Primeiro vamos obter

```
> s2 <- sum(resid(ex04.av)^2)/ex04.av$df.res
> dt <- qtukey(0.95, 3, 18) * sqrt(s2/4)
> dt
[1] 2.043945
>
> ex04.m
       e1     e2
r1 25.650 25.325
r2 25.875 19.575
r3 20.050 21.325
>
> m1 <- ex04.m[,1]
> m1
    r1     r2     r3
25.650 25.875 20.050
> m1d <- outer(m1,m1,"-")
> m1d
       r1     r2    r3
r1  0.000 -0.225 5.600
```

```
r2  0.225  0.000 5.825
r3 -5.600 -5.825 0.000
> m1d <- m1d[lower.tri(m1d)]
> m1d
    r2     r3   <NA>
 0.225 -5.600 -5.825
>
> m1n <- outer(names(m1),names(m1),paste, sep="-")
> names(m1d) <- m1n[lower.tri(m1n)]
> m1d
 r2-r1  r3-r1  r3-r2
 0.225 -5.600 -5.825
>
> data.frame(dif = m1d, sig = ifelse(abs(m1d) > dt, "*", "ns"))
         dif sig
r2-r1  0.225  ns
r3-r1 -5.600   *
r3-r2 -5.825   *
>
> m2 <- ex04.m[,2]
> m2d <- outer(m2,m2,"-")
> m2d <- m2d[lower.tri(m2d)]
> m2n <- outer(names(m2),names(m2),paste, sep="-")
> names(m2d) <- m2n[lower.tri(m2n)]
> data.frame(dif = m2d, sig = ifelse(abs(m2d) > dt, "*", "ns"))
        dif sig
r2-r1 -5.75   *
r3-r1 -4.00   *
r3-r2  1.75  ns
```

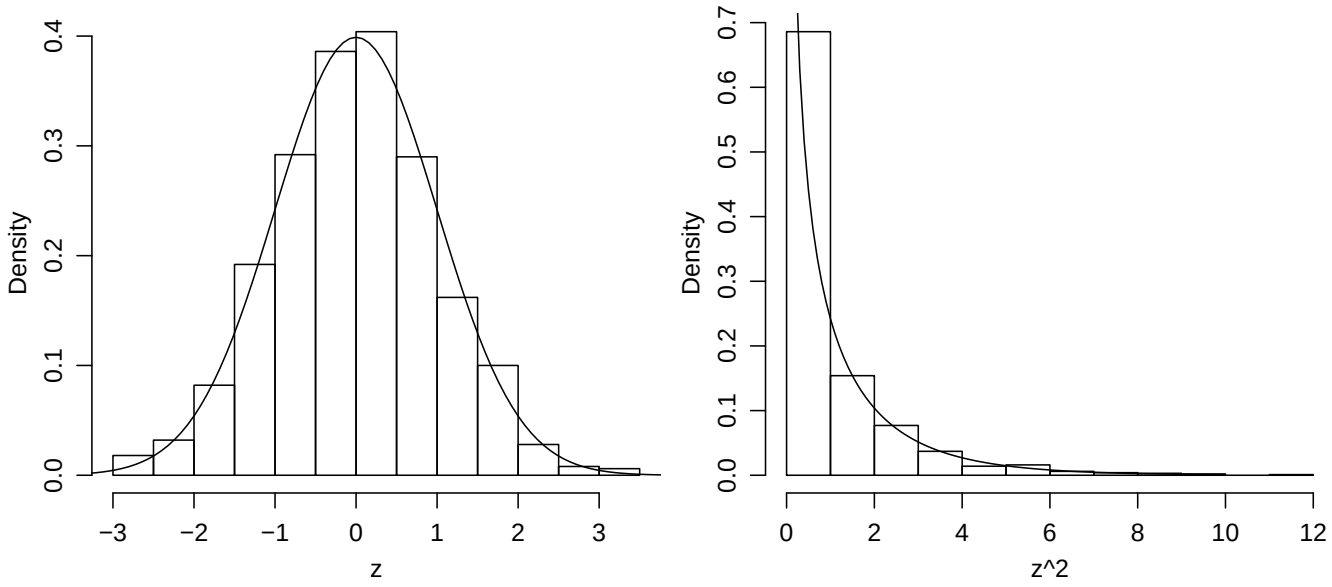



Figura 57: Histograma das amostra da e a curva teórica da distribuição normal padrão (esquerda) e histograma dos valores ao quadrado com a curva teórica da distribuição $\chi^2_{(1)}$ (direita).

# 26 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.

## 26.1 Relações entre a distribuição normal e a $\chi^2$

**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 57.

```
> 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 57 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 $\chi^2$ 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 58, 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 $\chi^2$ 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)`.

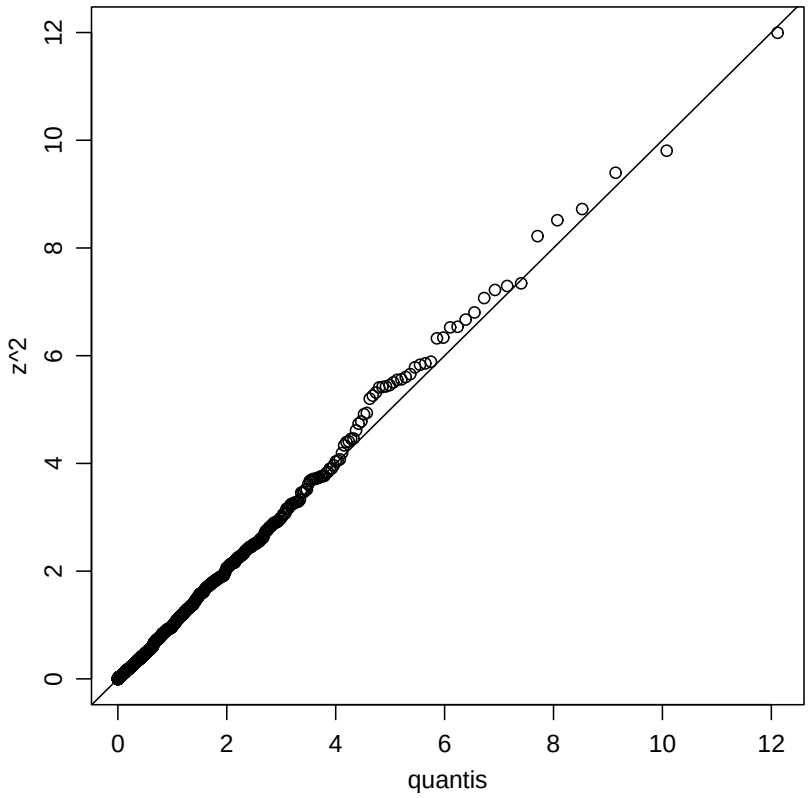



Figura 58: Comparando dados e quantis da $\chi^2$ utilizando o *qq-plot*

**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 distribuiçâo normal padrão, elevar os valores ao quadrado e, para cada amostra, somar os quadrados dos três

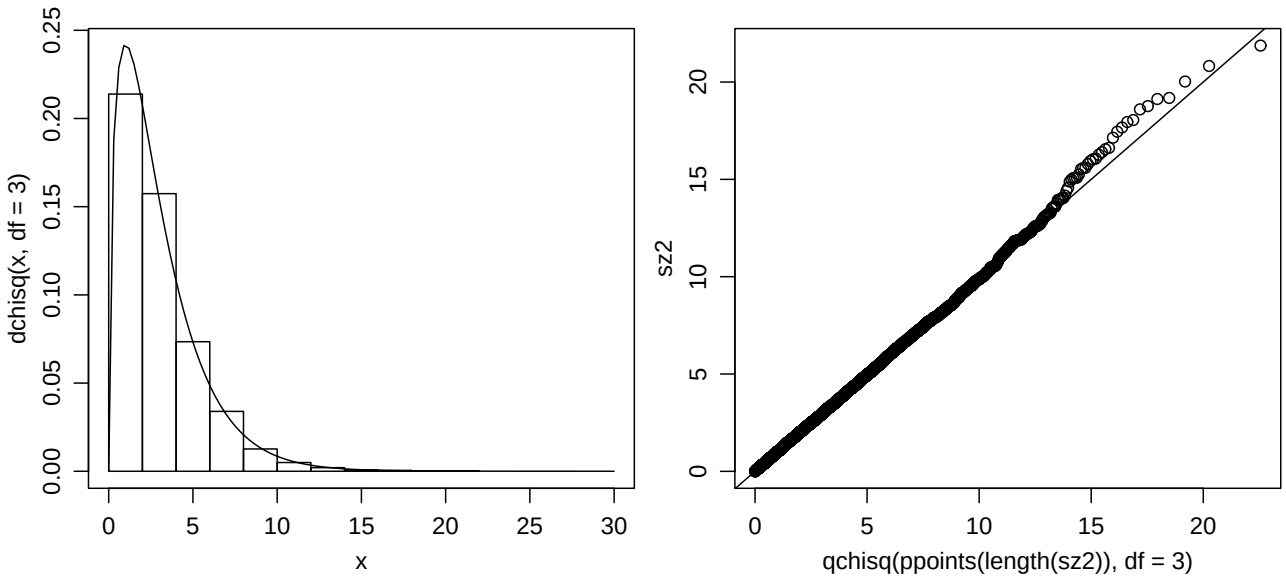



Figura 59: Histograma da uma amostra da soma dos quadrados de três valores da normal padrão e a curva teórica da distribuição de $\chi^2_{(3)}$ (esquerda) e o respectivo *qq-plot*.

números. Na Figura 59 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)
```

## 26.2 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 60 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.

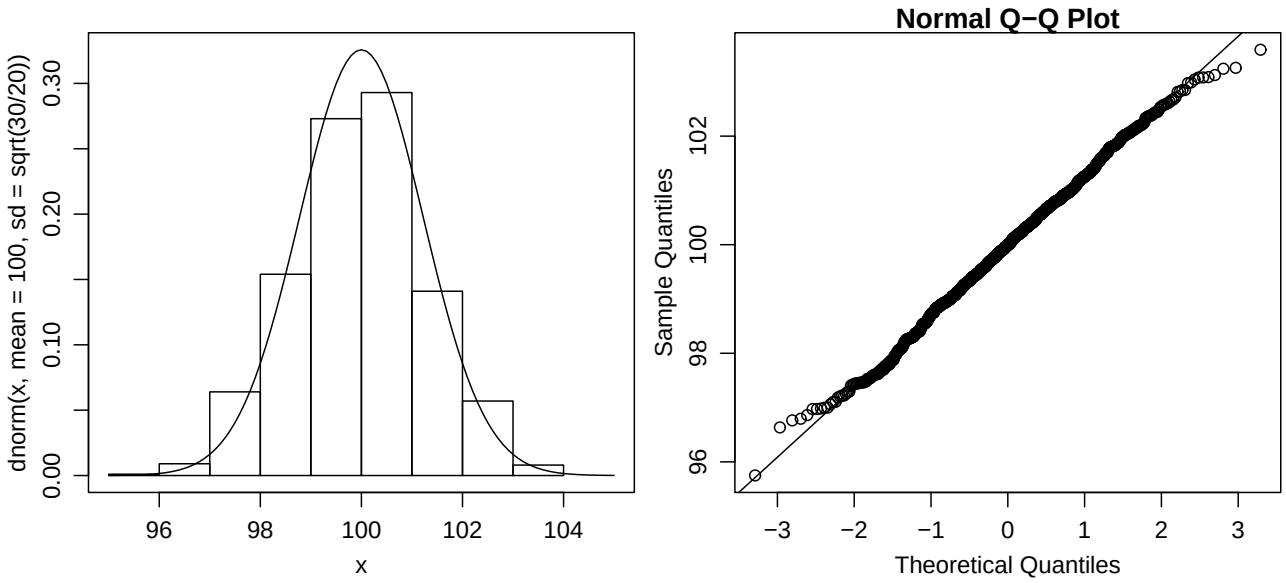



Figura 60: Histograma de uma amostra da distribuição amostral da média e a curva teórica da distribuição e o respectivo *qq-plot*.

## 26.3 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 $\chi^2_{n-1}$.
   **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 $\chi^2_{n-1}$.
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 $\chi^2$ tem distribuição $F$.

# 27 Agrupando comandos, execução condicional, controle de fluxo, ”loops” e a ”família” `*apply`

## 27.1 Agrupando comandos

O R é uma linguagem que interpreta expressões, o que implica que o único tipo de comando usado é uma expressão ou função que executa o processamento da requisição e retorna algum resultado. Nesta sessão vamos alguns formatos para facilitar/agilizar o uso de comandos.

É possível atribuir os mesmos valores a vários objetos de uma só vez utilizando atribuições múltiplas de valores.

```
> a <- b <- 10
> a

[1] 10

> b

[1] 10

> x <- y <- z <- numeric(5)
> x

[1] 0 0 0 0 0

> y

[1] 0 0 0 0 0

> z

[1] 0 0 0 0 0
```

Um grupo de comandos podem ser agrupado com `"{ }"` e separados por `";"` para digitação em uma mesma linha. Em certas situações, como no ”prompt”do R as chaves são opcionais.

```
> {
+     x <- 1:3
+     y <- x + 4
+     z <- y/x
+ }
> x <- 1:3
> y <- x + 4
> z <- y/x
> x

[1] 1 2 3

> y

[1] 5 6 7

> z

[1] 5.000000 3.000000 2.333333
```

## 27.2 Execução condicional

Execuções condicionais são controladas por funções especiais que verificam se uma condição é satisfeita para permitir a execução de um comando. As seguintes funções e operadores podem ser usadas para controlar execução condicional.

- `if()` (opcionalmente) acompanhado de `else`
- &, ||, && e |||
- `ifelse()`
- `switch()`

A estrutura `if() else` é comumente usada, em especial dentro de funções. Quando aplicada diretamente na linha de comando, é uma prática recomendada colocar chaves marcando o início e fim dos comandos de execução condicional. Quando a expressão que segue o `if()` e/ou `else` tem uma única linha ela pode ser escrita diretamente, entretando, caso sigam-se mais de duas linhas deve-se novamente usar chaves, agora também depois destes de forma que todos os comandos da execução condicional fiquem contidos na chave, caso contrário apenas a primeira linha será considerada para execução condicional e todas as demais são processadas normalmente. Inspecione os comandos a seguir que ilustram os diferentes usos.

```
> x <- 10
> y <- 15
> {
+     if (x > 8)
+         z <- 2 * x
+ }
> z

[1] 20

> rm(x, y, z)
> x <- 10
> y <- 15
> {
+     if (x > 12)
+         z <- 2 * x
+     else z <- 5 * x
+ }
> z

[1] 50

> rm(x, y, z)
> x <- 10
> y <- 15
> {
+     if (x > 8) {
+         z <- 2 * x
+         w <- z + y
+     }
```

```
+     else {
+         z <- 5 * x
+         w <- z - y
+     }
+ }
> z

[1] 20

> w

[1] 35

> rm(x, y, z, w)
> x <- 10
> y <- 15
> {
+     if (x > 8)
+         z <- 2 * x
+     w <- z + y
+     if (x <= 8)
+         z <- 5 * x
+     w <- z - y
+ }
> z

[1] 20

> w

[1] 5

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

Um comando útil para manipulação de dados é o `split()` que permite separa dados por grupos. Por exemplo considere o conjunto de dados codemtcars, que possui várias variáveis relacionadas a características de veículos. Entre as variáveis estão as que indicam o consumo (`mpg` - *miles per gallon*) e o tipo de câmbio, manual ou automático (`am`). Para separar os dados da variável `mpg` para cada tipo de câmbio, podemos usar:

```
> data(mtcars)
> with(mtcars, split(mpg, am))

$`0`
 [1] 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4 10.4 14.7 21.5 15.5 15.2 13.3

$`1`
 [1] 21.0 21.0 22.8 32.4 30.4 33.9 27.3 26.0 30.4 15.8 19.7 15.0 21.4
```

Outro comando com funcionalidade similar é `agregate()`.

## 27.3 Controle de fluxo

O controle de fluxo no R é implementado pelas funções `for()`, `while()` e `repeat()`. A escolha de qual usar vai depender do contexto e objetivo do código e em geral não existe solução única, sendo que uma mesma tarefa pode ser feita por uma ou outra.

Apenas para ilustração considere o seguinte exemplo resolvido de três formas diferentes com cada uma destas funções:
Dado um valor de $n$ gerar amostrar de tamanho $1, 2, \ldots, n$ e para calcule a média de cada amostra, com 3 casas decimais.

Primeiro vamos implementar uma solução usando `for()`.

```
> f1 <- function(n) {
+     medias <- numeric(n)
+     for (i in 1:n) {
+         am <- rnorm(i)
+         medias[i] <- round(mean(am), dig = 3)
+     }
+     return(medias)
+ }
> set.seed(283)
> f1(10)

 [1]  1.007 -0.063 -0.392  1.546  0.341 -0.514 -0.086 -0.224  0.137  0.138
```

Agora vamos executar a mesma tarefa com `while()`

```
> f2 <- function(n) {
+     medias <- numeric(n)
+     i <- 1
+     while (i <= n) {
+         am <- rnorm(i)
+         medias[i] <- round(mean(am), dig = 3)
+         i <- i + 1
+     }
+     return(medias)
+ }
> set.seed(283)
> f2(10)

 [1]  1.007 -0.063 -0.392  1.546  0.341 -0.514 -0.086 -0.224  0.137  0.138
```

E finalmente a mesma tarefa com `repeat()`

```
> f3 <- function(n) {
+     medias <- numeric(n)
+     i <- 1
+     repeat {
+         am <- rnorm(i)
+         medias[i] <- round(mean(am), dig = 3)
+         if (i == n)
+             break
+         i <- i + 1
```

```
+     }
+     return(medias)
+ }
> set.seed(283)
> f3(10)

 [1]  1.007 -0.063 -0.392  1.546  0.341 -0.514 -0.086 -0.224  0.137  0.138
```

**NOTA:** as soluções acima são apenas ilustrativas e não representam a forma mais eficiente de efetuar tal operação o R. Na verdade, para este tipo de cálculo recomenda-se o uso de funções do tipo `*apply` que veremos no restante desta sessão.

## 27.4 Alguns comentários adicionais

Nas soluções acima as amostras foram usadas para calcular as médias e depois descartadas. Suponha agora que queremos preservar e retornar também os dados simulados. Para ilustrar vamos mostrar como fazer isto modificando um pouco a primeira função.

```
> f1a <- function(n) {
+     res <- list()
+     res$amostras <- list()
+     res$medias <- numeric(n)
+     for (i in 1:n) {
+         res$amostras[[i]] <- rnorm(i)
+         res$medias[i] <- round(mean(res$amostras[[i]]), dig = 3)
+     }
+     return(res)
+ }
> set.seed(283)
> ap <- f1a(4)
> names(ap)

[1] "amostras" "medias"

> ap

$amostras
$amostras[[1]]
[1] 1.006870

$amostras[[2]]
[1]  0.2003886 -0.3257288

$amostras[[3]]
[1]  0.4913491 -1.0009700 -0.6665789

$amostras[[4]]
[1] 2.035963 1.174572 1.214059 1.761383



$medias
[1]  1.007 -0.063 -0.392  1.546
```

Vamos agora ver uma outra modificação. Nas funções acima geravamos amostras com tamanhos sequênciais com incremento de 1 elemento no tamanho da amostra. A função a seguir mostra como gerar amostras de tamanhos especificados pelo usuário e para isto toma como argumento um vetor de tamanhos de amostra.

```
> f5 <- function(ns) {
+     medias <- numeric(length(ns))
+     j <- 1
+     for (i in ns) {
+         am <- rnorm(i)
+         medias[j] <- round(mean(am), dig = 3)
+         j <- j + 1
+     }
+     return(medias)
+ }
> set.seed(231)
> f5(c(2, 5, 8, 10))

[1] -1.422 -0.177  0.056  0.158
```

## 27.5 Evitando "loops" — a "família" *apply

O R é uma linguagem vetorial e "loops"podem **e devem** ser substituídos por outras formas de cálculo sempre que possível. Usualmente usamos as funções `apply()`, `sapply()`, `tapply()` e `lapply()` para implementar cálculos de forma mais eficiente. Vejamos algums exemplos.

- `apply()` para uso em matrizes, arrays ou data-frames
- `tapply()` para uso em vetores, sempre retornando uma lista
- `sapply()` para uso em vetores, simplificando a estrutura de dados do resultado se possível (para vetor ou matriz)
- `mapply()` para uso em vetores, versão multivariada de `sapply()`
- `lapply()` para ser aplicado em listas

1. Seja o problema mencionado no início desta sessão de gerar amostras de tamanhos sequenciais e calcular a média para cada uma delas. Uma alternativa aos códigos apresentados seria:

   ```
   > set.seed(283)
   > sapply(1:10, function(x) round(mean(rnorm(x)), dig = 3))

    [1]  1.007 -0.063 -0.392  1.546  0.341 -0.514 -0.086 -0.224  0.137  0.138
   ```

2. Considere agora a modificação mencionado anteriormente de calcular médias de amostras com tamanho fornecidos pelo usuário

   ```
   > vec <- c(2, 5, 8, 10)
   > f6 <- function(n) round(mean(rnorm(n)), dig = 3)
   > set.seed(231)
   > sapply(vec, f6)
   ```

```
[1] -1.422 -0.177  0.056  0.158
```

3. No próximo exemplo consideramos uma função que simula dados e calcula medidas de posição e dispersão associadas utilizando para cada uma delas duas medidas alternativas. Inicialmente definimos a função:

```
> proc <- function(...) {
+     x <- rnorm(500)
+     modA <- list(pos = mean(x), disp = sd(x))
+     modB <- list(pos = mean(x, trim = 0.1), disp = mad(x))
+     return(list(A = modA, B = modB))
+ }
```

Agora vamos rodar a função 10 vezes.

```
> set.seed(126)
> res <- lapply(1:10, proc)
```

O resultado está armazanado no objeto `res`, que neste caso é uma lista. Agora vamos extrair desta lista as médias aritméticas e depois ambas, média aritmética e aparada:

```
> mediaA <- function(x) x$A$pos
> mA <- sapply(res, mediaA)
> mediaAB <- function(x) c(x$A$pos, x$B$pos)
> mAB <- sapply(res, mediaAB)
> rownames(mAB) <- paste("modelo", LETTERS[1:2], sep = "")
> mAB

              [,1]        [,2]      [,3]       [,4]       [,5]       [,6]       [,7]
modeloA 0.02725767 -0.01017973 0.0958355 0.02058979 0.04582751 0.07898205 0.06122656 -0.0
modeloB 0.01706928 -0.02781770 0.1023454 0.02210935 0.06210404 0.05914628 0.04085053 -0.0
                [,9]       [,10]
modeloA  0.006781871 -0.02798788
modeloB -0.020411456 -0.02029610
```

Os comandos acima podem ser reescritos em versões simplificadas:

```
> mA <- sapply(res, function(x) x$A$pos)
> mA

 [1]  0.027257675 -0.010179733  0.095835502  0.020589788  0.045827513  0.078982050  0.061
 [8] -0.059818054  0.006781871 -0.027987878

> mAB <- sapply(res, function(x) sapply(x, function(y) y$pos))
> mAB

        [,1]        [,2]      [,3]       [,4]       [,5]       [,6]       [,7]        [,8
A 0.02725767 -0.01017973 0.0958355 0.02058979 0.04582751 0.07898205 0.06122656 -0.059818
B 0.01706928 -0.02781770 0.1023454 0.02210935 0.06210404 0.05914628 0.04085053 -0.056808
          [,9]       [,10]
A  0.006781871 -0.02798788
B -0.020411456 -0.02029610
```

E para obter as médias das médias de cada medida:

```
> apply(mAB, 1, mean)

         A          B
0.02385153 0.01782913
```

4. A função `tapply()` pode ser usada para calcular o resultado de uma operação sobre dados, para cada um dos níveis de uma segunda variável No primeiro exemplo consideramos novamente o conjunto de dados `mtcars` mencionado anteriormente. Os comandos abaixo calculam média, variância e coeficinte de variação do consumo para cada tipo de cambio.

```
> with(mtcars, tapply(mpg, am, mean))

       0        1
17.14737 24.39231

> with(mtcars, tapply(mpg, am, var))

       0        1
14.69930 38.02577

> with(mtcars, tapply(mpg, am, function(x) 100 * sd(x)/mean(x)))

       0        1
22.35892 25.28053
```

Vejamos ainda um outro exemplo onde definimos 50 dados divididos em 5 grupos.

```
> x <- rnorm(50, mean = 50, sd = 10)
> y <- rep(LETTERS[1:5], each = 10)
> x

 [1] 55.66788 43.71391 42.78483 50.28745 40.77170 62.06800 60.53166 51.90432 56.41214 65.
[11] 35.99390 42.67566 40.26776 47.61359 57.92209 60.69673 48.80234 44.29422 44.48886 39.
[21] 60.93054 32.73959 39.37867 56.89312 37.06637 61.45986 44.66166 50.60778 37.78913 39.
[31] 48.53931 43.29661 66.03602 65.55652 58.05864 55.21829 45.90542 45.01864 37.73984 38.
[41] 34.85114 34.24760 65.07629 49.01286 62.37572 38.36997 57.93003 39.72861 66.62899 45.

> y

 [1] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "B" "B" "B" "B" "B" "B" "B" "B" "B" "B" "C"
[25] "C" "C" "C" "C" "C" "D" "D" "D" "D" "D" "D" "D" "D" "D" "D" "E" "E" "E" "E" "E"
[49] "E" "E"

> gM <- tapply(x, y, mean)
> gM

       A        B        C        D        E
52.96075 46.17779 46.06588 50.33724 49.35969

> gCV <- tapply(x, y, function(z) 100 * sd(z)/mean(z))
> gCV
```

```
       A        B        C        D        E
16.19106 17.17599 23.08328 20.65681 25.77284
```

Para organizar os dados em um `data-frame`:

```
> xy <- data.frame(x = rnorm(50, mean = 50, sd = 10), y = rep(LETTERS[1:5], each = 10))
> gM <- with(xy, tapply(x, y, mean))
> gM

       A        B        C        D        E
49.91571 51.03091 45.26204 47.45439 47.25661

> gCV <- with(xy, tapply(x, y, function(z) 100 * sd(z)/mean(z)))
> gCV

       A        B        C        D        E
16.13100 11.97707 17.35279 18.67300 16.03077
```

5. Considere gerarmos uma matrix $1000 \times 300$ representando 1000 amostras de tamanho 300. O que desejamos é calcular a média de cada uma das amostras Os códigos a seguir mostras três formas alternativas de fazer isto. Encapsulamos os comandos com a função `system.time()` que compara os tempos de execução.

```
> x <- matrix(rnorm(1000 * 300), nc = 300)
> system.time({
+     f <- function(x) {
+         mx <- numeric(1000)
+         for (i in 1:1000) mx[i] <- mean(x[i, ])
+         mx
+     }
+     mx <- f(x)
+ })

[1] 0.200 0.004 0.209 0.000 0.000

> system.time(mx <- apply(x, 1, mean))

[1] 0.224 0.024 0.253 0.000 0.000

> system.time(mx <- rowMeans(x))

[1] 0.016 0.000 0.016 0.000 0.000
```

A função `rowMeans()` é substancialmente mais eficiente (menos tempo de execução. Outras funções simulares são `colMeans()`, `rowSums()` e `colSums()`.

6. Considere o seguinte problema:
Sejam `li` e `ls` vetores com os limites superiores e inferiores definindo intervalos. Inicialmente vamos simular estes valores.

```
> li <- round(rnorm(500, m = 70, sd = 10))
> ls <- li + rpois(li, lam = 5)
```

O que queremos montar um vetor com os valores únicos que definem estes intervalos, e testar a pertinência de cada elemento a cada um dos intervalos. Ao final teremos uma matrix incidando, para cada elemento do vetor de valores únicos, a pertinência a cada intervalo. Inicialmente vamos fazer um código usando "loops"guardando os resultados no objeto `B`.

```
> system.time({
+     aux <- sort(c(li, ls))
+     m <- length(table(aux))
+     all <- rep(min(aux), m)
+     for (j in 1:(m - 1)) {
+         all[j + 1] <- min(aux[aux > all[j]])
+     }
+     n <- length(li)
+     aij <- matrix(0, nrow = n, ncol = m)
+     for (i in 1:n) {
+         for (j in 1:m) {
+             aij[i, j] <- ifelse(all[j] >= li[i] & all[j] <= ls[i], 1, 0)
+         }
+         B <- aij
+     }
+ })

[1]  9.544  0.140 14.116  0.000  0.000
```

Agora, usando a estrutura vetorial da linguagem R vamos reimplementar este código de maneira mais eficiente e adequada para a linguagem, usando `sapply()`, guardando os resultados no objeto `A`. Ao final usamos `identical()` para testar se os resultados numéricos são exatamente os mesmos. Note a diferença nos tempos de execução.

```
> system.time({
+     all <- sort(unique(c(li, ls)))
+     interv1 <- function(x, inf, sup) ifelse(x >= inf & x <= sup, 1, 0)
+     A <- sapply(all, interv1, inf = li, sup = ls)
+ })

[1] 0.108 0.000 0.144 0.000 0.000

> identical(A, B)

[1] TRUE
```

7. Considere agora uma extensão do problema anterior. Queremos montar o vetor com os valores únicos que definem estes intervalos como no caso anterior, e depois usar este vetor montar intervalos com pares de elementos consecutivos deste vetor e testar se cada um destes intervalos está contido em cada um dos intervalos originais. O resultado final é uma matrix indicando para cada intervalo obtido desta forma a sua pertinência a cada um dos intervalos originais. Da mesma forma que no caso anterior implementamos com um "loop"e depois usando a estrutura vetorial da linguagem, e testando a igualdade dos resultados com `identical()`.

```
> li <- round(rnorm(500, m = 70, sd = 10))
> ls <- li + rpois(li, lam = 5)
```

```
> system.time({
+     aux <- sort(c(li, ls))
+     m <- length(table(aux))
+     all <- rep(min(aux), m)
+     for (j in 1:(m - 1)) {
+         all[j + 1] <- min(aux[aux > all[j]])
+     }
+     n <- length(li)
+     aij <- matrix(0, nrow = n, ncol = m - 1)
+     for (i in 1:n) {
+         for (j in 1:m - 1) {
+             aij[i, j] <- ifelse(all[j] >= li[i] & all[j + 1] <= ls[i], 1, 0)
+         }
+         B <- aij
+     }
+ })

[1] 10.892  0.132 13.910  0.000  0.000

> system.time({
+     all <- sort(unique(c(li, ls)))
+     all12 <- cbind(all[-length(all)], all[-1])
+     interv1 <- function(x, inf, sup) ifelse(x[1] >= inf & x[2] <= sup, 1, 0)
+     A <- apply(all12, 1, interv1, inf = li, sup = ls)
+ })

[1] 0.120 0.000 0.156 0.000 0.000

> identical(A, B)

[1] TRUE
```

# 28 Ajuste de modelos não linares

## 28.1 Exemplo: modelo de van Genutchen

Este exemplo mostra o ajuste de um modelo não linear. Primeiro discutimos como efetuar um único ajuste para um conjunto de dados e algumas sugestões para examinar resultados. Ao final mostramos como efetuar vários ajustes de uma só vez de forma eficiente e extrair alguns resultados de particular interesse.

O exemplo mostrado aqui foi motivado por um questão levntada pelo Prof. Álvaro Pires da Silva do Departamento de Ciência do Solo da ESALQ/USP e refere-se ao ajuste da equação de van Genutchen para a *curva de retenção de água no solo* (ou *curva de retenção de água no solo*).

Informalmente falando, a equação de van Genutchen é um dos modelo matemático utilizados para descrever a curva característica de água no solo que caracteriza a armazenagem de água através de relação entre a umidade e o potencial matricial da água no solo. Para determinação da curva característica de água em um determinado solo o procedimento usual é o de se tomar uma amostra que é submetida em condições de laboratório a diferentes tensões. Para cada tensão aplicada mede-se a umidade da amostra. A curva então é um modelo ajustado a estes pontos que descreve a umidade em função da tensão. A equação de van Genutchen é dada por:

$$\theta = \theta_R + (\theta_S - \theta_R)\left[\frac{1}{1+(\alpha\Psi_m)^n}\right]^{1-(1/n)} \tag{5}$$

em que $\theta$ é a umidade volumétrica, $\Psi_m$ o potencial matricial, $\theta_S$ e $\theta_R$ a umidade volumétrica na saturação e residual, respectivamente.

Para exemplificar o ajuste utilizamos dados cedidos pelo Prof. Álvaro que podem ser obtidos usando o comando mostrado a seguir. Este conjunto de dados refere-se a apenas duas amostras que são um subconjunto dos de dados original que contém diversas amostras. O objetivo é determinar da curva de retenção de água no solo estimada segundo modelo de van Genutchen para cada uma das amostras. Portanto, os dados consistem, para cada amostra, de uma série de valores de tensão aplicada e umidade do solo. No objeto `cra` a primeira coluna (`am`) indica o número da amostra, a segunda (`pot`) o potencial aplicado e a terceira (`u`) a umidade do solo. Vemos a seguir que dispomos de 15 pontos medidos da curva de retenção da primeira amostra e 13 para a segunda.

```
> cra <- read.table("http://www.leg.ufpr.br/~paulojus/aulasR/dados/cra.csv",
+     header = T, sep = ",")
> head(cra)

  am pot      u
1 30  10 0.3071
2 30  19 0.2931
3 30  30 0.2828
4 30  45 0.2753
5 30  63 0.2681
6 30  64 0.2628

> cra <- transform(cra, am = as.factor(am))
> summary(cra)

  am          pot                u
 30:15   Min.   :   10.0   Min.   :0.0636
 41:13   1st Qu.:   58.5   1st Qu.:0.1199
```

```
 Median :  107.5   Median :0.1969
 Mean   : 2139.8   Mean   :0.1879
 3rd Qu.: 1550.0   3rd Qu.:0.2436
 Max.   :26300.0   Max.   :0.3071
```

Inicialmente vamos nos concentrar na discussão do ajuste do modelo e para isto, vamos isolar os dados referentes a uma única amostra.

```
> cra30 <- subset(cra, am == 30)
> cra30
```

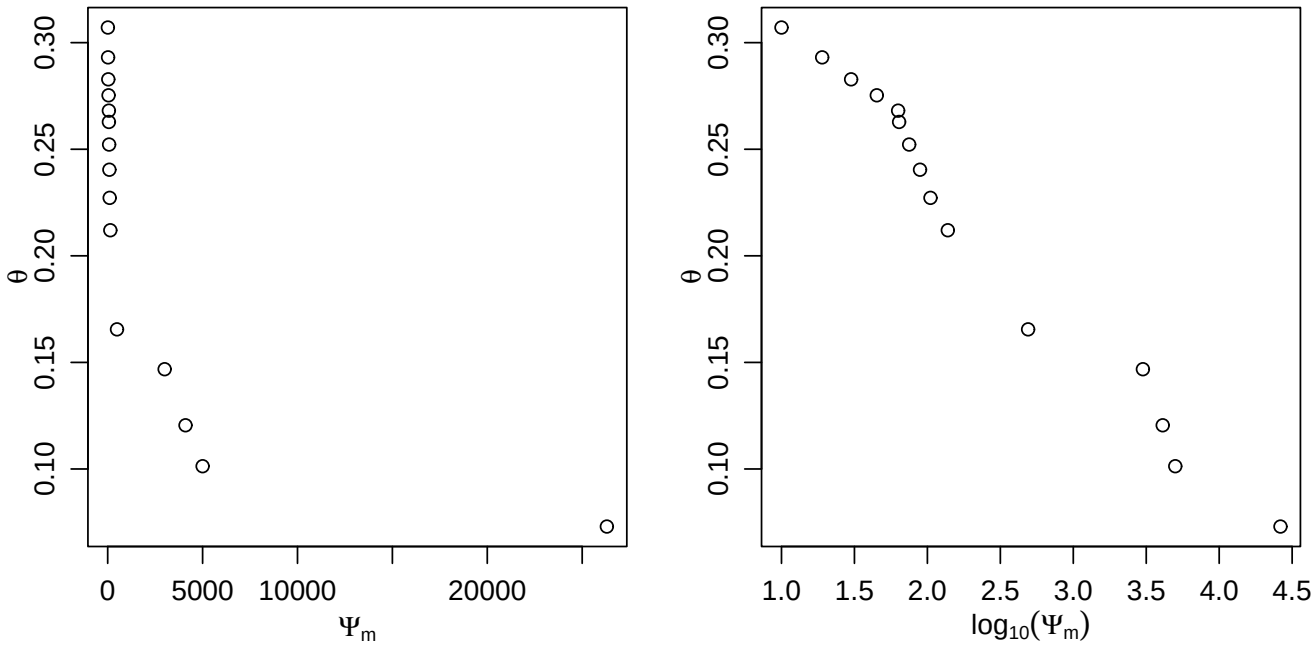
```
   am   pot      u
1  30    10 0.3071
2  30    19 0.2931
3  30    30 0.2828
4  30    45 0.2753
5  30    63 0.2681
6  30    64 0.2628
7  30    75 0.2522
8  30    89 0.2404
9  30   105 0.2272
10 30   138 0.2120
11 30   490 0.1655
12 30  3000 0.1468
13 30  4100 0.1205
14 30  5000 0.1013
15 30 26300 0.0730
```

Na Figura 28.1 visualizamos os dados no gráfico de umidade *versus* pressão aplicada. O gráfico da esquerda mostra os dados originais e no da direita um gráfico utilizado na prática para melhor visualização onde no eixo horizontal utiliza-se o logarítmo (base 10) dos valores das pressões aplicadas.

```
> with(cra30, plot(u ~ pot, xlab = expression(Psi[m]), ylab = expression(theta)))
> with(cra30, plot(u ~ log10(pot), xlab = expression(log[10](Psi[m])),
+     ylab = expression(theta)))
```

Portanto, os dados nas colunas `u` e `pot` correspondem à $\theta$ e $\psi_m$ na equação 5, e as demais quantidades $(\theta_R, \theta_R, m, \alpha)$ são parâmetros (coeficientes) a serem estimados a partir do ajuste do modelo teórico aos dados. Este é um modelo não linear e para o ajuste utilizamos a função `nls()` que ajusta modelos são lineares pelo método de mínimos quadrados. A função possui três argumentos obrigatórios: (i) o primeiro é utilizado para declarar a expressão do modelo a ser ajustado, (ii) o segundo informa o objeto contendo o conjunto de dados cujas nomes das colunas relevantes devem ter o mesmo nome utilizado na declaração do modelo e, (iii) valores iniciais para os parâmetros a serem ajustados que devem ser passados por uma *named list*, isto é, uma lista com nomes dos elementos, e estes nomes devem coincidir com os utilizados na declaração do modelo. Há argmentos adicionais para ocntrolaro algorítimo e para maios detalhes veja a documentação da função. Na visualização dos resultados no restante desta seção optamos por omitir símbolos para indicara significância dos resultados

```
> fit30 = nls(u ~ ur + (us - ur)/((1 + (alpha * pot)^n)^(1 - 1/n)),
+     data = cra30, start = list(us = 0.2236, ur = 0.0611, alpha = 0.056,
+         n = 1.5351))
> options(show.coef.Pvalues = T)
> summary(fit30)
```



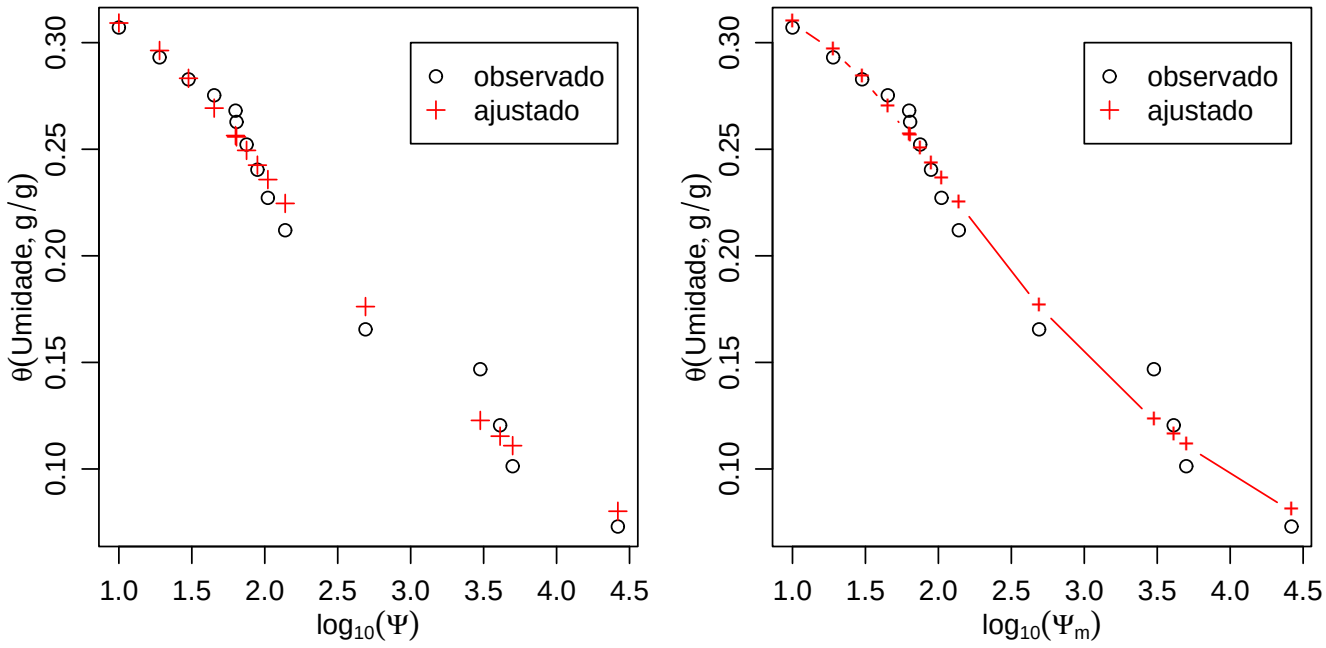

```
Formula: u ~ ur + (us - ur)/((1 + (alpha * pot)^n)^(1 - 1/n))

Parameters:
      Estimate Std. Error t value Pr(>|t|)
us    0.324120   0.017744   18.27 1.41e-09 ***
ur    0.007083   0.071083    0.10    0.922
alpha 0.038780   0.026202    1.48    0.167
n     1.211817   0.105207   11.52 1.77e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.01104 on 11 degrees of freedom
```

A partir do modelo ajustado pode-se calcular quantidades de interesse. Por exemplo, nesta caso calculamos uma quantidade denotada por $S$ que é um indicador da qualidade física do solo. Quanto maior o valor de $S$, melhor a sua qualidade física.

```
> S = with(as.list(coef(fit30)), abs((-n * (us - ur) * (((2 * n -
+     1)/(n - 1))^(1/n - 2)))))
> S

[1] 0.04097132
```

Os valores preditos são obtidos de forma direta com qualquer um dos comandos abaixo:

```
> fitted(fit30)

 [1] 0.3092108 0.2963160 0.2832760 0.2692882 0.2564222 0.2558002 0.2494653 0.2425230
 [9] 0.2357620 0.2245840 0.1761765 0.1227784 0.1153897 0.1109397 0.0801680
attr(,"label")
[1] "Fitted values"

> predict(fit30)
```




```
 [1] 0.3092108 0.2963160 0.2832760 0.2692882 0.2564222 0.2558002 0.2494653 0.2425230
 [9] 0.2357620 0.2245840 0.1761765 0.1227784 0.1153897 0.1109397 0.0801680
```

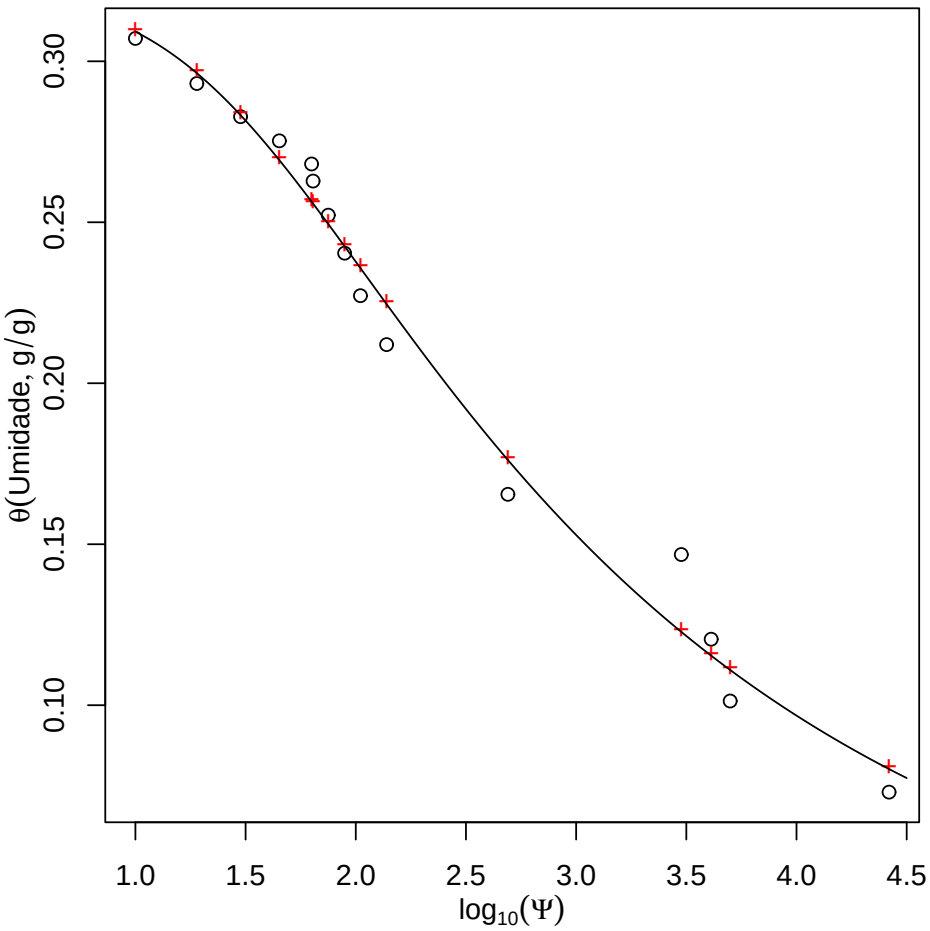
Para visualização e avaliação do modelo ajustado podemos fazer diferentes gráficos. A Figura 28.1 mostra os pontos ajustados no gráfico da esquerda, e a união destes pontos no gráfico da direita.

```
> with(cra30, plot(log10(pot), u, xlab = expression(log[10](Psi)),
+     ylab = expression(theta(Umidade, g/g))))
> with(cra30, points(log10(pot), fitted(fit30), pch = 3, col = "red"))
> legend(3, 0.3, c("observado", "ajustado"), pch = c(1, 3), col = c(1,
+     2))
> with(cra30, plot(log10(pot), u, xlab = expression(log[10](Psi[m])),
+     ylab = expression(theta(Umidade, g/g))))
> with(cra30, points(log10(pot), fitted(fit30), type = "b", pch = "+",
+     col = "red"))
> legend(3, 0.3, c("observado", "ajustado"), pch = c(1, 3), col = c(1,
+     2))
```

Entretanto, para obter uma melhor visualização do modelo ajustado é recomendado obter a curva ajustada não apenas nos pontos observados, mas em uma sequência de valores ao longo do gráfico. Para isto, obtemos os valores preditos para esta sequência de valores como ilustrado a seguir. Note que neste exemplo consideramos que o interesse é de visualização na escala logarítmica do potencial. A Figura 28.1 mostra o modelo ajustado.

```
> with(cra30, plot(log10(pot), u, xlab = expression(log[10](Psi)),
+     ylab = expression(theta(Umidade, g/g))))
> with(cra30, points(log10(pot), fitted(fit30), pch = "+", col = "red"))
> pp <- 10^seq(1, 4.5, l = 201)
> lines(log10(pp), predict(fit30, list(pot = pp)))
```

**C**omentários: é importante lembrar que certos modelos não lineares são *parcialmente linearizáveis* e neste caso o ajuste pode ser mais preciso e numericamente estável se beneficiando disto para reduzir a dimensão do problema de otimização numérica. Entretanto não vamos os ater a este ponto




aqui. Há também que se pensar em fazer o ajuste na escala de $log_{10}(\Psi_m)$ já que os resultados são tipicamente visualizados desta forma. Isto reduz a escala dos valores das variáveis e também torna o problema mais estável numericamente. Finalmente cuidados usuais com ajuste de modelos utilizando métodos iterativos devem ser observados, tais como sensibilidade a valores iniciais e verificações de convergência.

## 28.2 Ajustando modelo a vários conjuntos de dados

Agora, vamos considerar que temos várias amostras e que desejamos fazer vários ajustes como ilustrado anteriormente para cada uma das amostras individualmente, porém de forma automática, sem a necesidade e repetir os passos acima a cada ajuste. Neste exemplo temos duas amostras, mas o procedimento a seguir funcionará igualmente para um maior númeo de amostras.

Inicialmente definimos uma função que contém a chamada à `nls()` como acima. Neste função estamos incluindo um argumento `ini` paa passar valores iniciais que caso não fornecido assumirá os valores indicados. A seguir utilizamos a função `by()` para proceder o ajuste para cada amostra individualmente. Esta função retorna uma lista com dois elementos, um para cada amostra, sendo que cada um deles contém o ajuste do modelo não linear.

```
> fit.vG <- function(x, ini = list(us = 0.2236, ur = 0.0611, alpha = 0.056,
+     n = 1.5351)) nlsfit = nls(u ~ ur + (us - ur)/(1 + (alpha * pot)^n)^(1 -
+     1/n), data = x, start = ini)
> allfits <- by(cra, cra$am, fit.vG)
> names(allfits)

[1] "30" "41"
```

Neste caso, o objeto resultante `allfits` é uma *lista de listas* e portanto podemos usar funções como `lapply()`, `sapply()` ou similares para extrair resultados de interesse. Note que a primeira retorna sempre uma lista, enquanto que a segunda "simplifica"o objeto resultante se possível. Por exemplo, quando extraindo coeficientes a função retorna uma matrix $4 \times 2$, já que para cada uma das duas amostras são extraidos quatro coeficientes.

```
> lapply(allfits, summary)

$`30`

Formula: u ~ ur + (us - ur)/(1 + (alpha * pot)^n)^(1 - 1/n)

Parameters:
      Estimate Std. Error t value Pr(>|t|)
us    0.324120   0.017744   18.27 1.41e-09 ***
ur    0.007083   0.071083    0.10    0.922
alpha 0.038780   0.026202    1.48    0.167
n     1.211817   0.105207   11.52 1.77e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.01104 on 11 degrees of freedom


$`41`

Formula: u ~ ur + (us - ur)/(1 + (alpha * pot)^n)^(1 - 1/n)

Parameters:
       Estimate Std. Error t value Pr(>|t|)
us     0.243148   0.009446  25.741 9.71e-10 ***
ur    -0.122405   0.171619  -0.713    0.494
alpha  0.035929   0.022324   1.609    0.142
n      1.113319   0.079472  14.009 2.04e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.006207 on 9 degrees of freedom

> lapply(allfits, coef)

$`30`
         us          ur       alpha           n
0.324120488 0.007082856 0.038779972 1.211816892

$`41`
         us          ur       alpha           n
 0.24314808 -0.12240506  0.03592887  1.11331890

> sapply(allfits, coef)
```

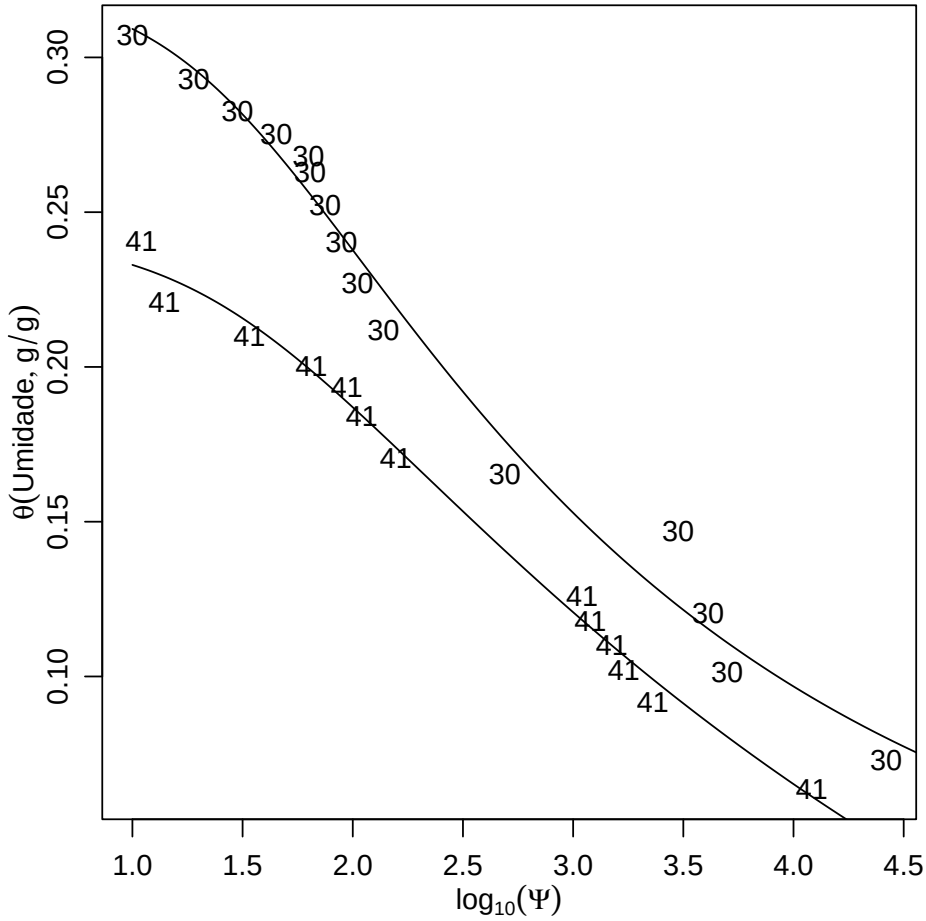
```
               30          41
us    0.324120488  0.24314808
ur    0.007082856 -0.12240506
alpha 0.038779972  0.03592887
n     1.211816892  1.11331890
```

Quando ajustamos o modelo apenas para uma das amostras mostramos como calcular o índice $S$ de qualidade física do solo a partir dos coeficientes estimados. Vamos então aqui obter este índice para cada uma das amostra. Para isto simplesmente definimos uma função que recebe o modelo ajustado e usa os coeficiente para calcular o valor de $S$. Passamos o objeto (lista) contendo todos os ajustes e a função que calcula $S$ para `sapply()` que neste caso vai simplificar o resultado para formato de um vetor, já que a função `calculaS` retorna um escalar para cada amostra.

```
> calculaS <- function(fit) with(as.list(coef(fit)), abs((-n * (us -
+     ur) * (((2 * n - 1)/(n - 1))^(1/n - 2)))))
> Sall <- sapply(allfits, calculaS)
> Sall

        30         41
0.04097132 0.02950316
```

Finalmente, para encerrar este exemplo, vamos mostrar uma possível forma de combinar a visualização dos ajustes em em um único gráfico. Começamos definindo uma sequência de valores para os quais queremos visualizar os ajustes. Armazenamos os valores preditos para cada amostra no objeto `allpred` e optamos aqui por mostrar os ajustes para as duas amostras no mesmo gráfico.

```
> lpsimax <- with(cra, max(log(pot)))
> pp <- 10^seq(1, lpsimax, l = 501)
> allpred <- lapply(allfits, predict, list(pot = pp))
> with(cra, plot(log10(pot), u, type = "n", , xlab = expression(log[10](Psi)),
+     ylab = expression(theta(Umidade, g/g))))
> with(cra, text(log10(pot), u, as.character(am)))
> lapply(allpred, function(yp) lines(log10(pp), yp))
```




# 29 Interface com códigos compilados

O R pode utilizar códigos compilados em Fortran, C, C++ e Delphi.

Abaixo apresentamos um exemplo simples de como fazer tal interface. Maiores detalhes estão disponíveis no manual *Wrinting R Extensions*.

As instuções a seguir são direcionadas para o sistema operacional LINUX. Assume-se que exista um compilador C (por exemplo `gcc` disponível no sistema. O mesmo recurso também pode ser usado em outros sistemas operacionais tais como Windows.

Considere o seguinte código em C que gravaremos no arquivo `test.c`

```
=========================================================================
#include <math.h>
#include <R.h>
#include <Rmath.h>

void cormatern(int *n, double *uphi, double *kappa, double *ans)
{
 int register i;
 double cte;
 for (i=0; i<*n; i++){
  if (uphi[i]==0) ans[i] = 1;
  else{
   if (*kappa==0.5)
   ans[i] = exp(-uphi[i]);
```

```
   else {
    cte = R_pow(2, (-(*kappa-1)))/gammafn(*kappa);
    ans[i] = cte * R_pow(uphi[i], *kappa) * bessel_k(uphi[i],*kappa,1);
}}}}
=============================================================================
```

Compilamos o código em C na linha de comando do LINUX com uma ferramenta do próprio R. O comando a seguir vai prodizir ambos: `test.o` e `test.so`

```
$ R CMD SHLIB teste.c
$ R
```

Uma vez criado o objeto compartilhado `test.so` (seria um `test.dll` no Windows) vamos usar uma função do R para acessar função disponibilidadas por este objeto. No caso de código C mostrado a seguir usamos `C()`. Para código Fortran usa-se `.Fortran()` e para C++ `.Call`. A seguir iniciamos o R e vamos definir fazer uma função "wrapper" em R que vai chamar, passar dados e receber resultados da rotina em C.

```
"matern" <- function(u, kappa){
  out <- .C("cormatern",
            as.integer(length(u)),
            as.double(u),
            as.double(kappa),
            res = as.double(rep(0,length(u))))$res
  return(out)
}
```

Depois basta carregar o objeto compartilhado ("shared object") e usar a sua função em R como no exemplo a seguir.

```
> dyn.load('teste.so')
> matern(0.1, 1)
> matern(seq(0,1,l=11), 1)
```

# 30 (Re)-direcionando saídas texto e gráficas

Por "default"o R em uma sessão interativa produz saídas texto na janela do programa e saídas gráficas um uma janela gráfica. Portanto, a tela texto e a janela gráficas devem ser entendidas como saídas padrão, cujos conteúdos podem ser redirecionados pelo usuário para outro local (dispositivo) como, por exemplo, um arquivo.

## 30.1 Texto

**Usando `sink()`** As saídas em formato texto podem ser redirecionadas para um arquivo usando

```
> sink("nome_do_arquivo")
```

que recebe como argumento o nome do arquivo (entre aspas) para onde queremos direcionar as saídas. Depois de digitarmos este comando os resultados deixam de ser mostrado na tela sendo enviados para o arquivo. Para encerrar o envio de conteúdo para o arquivo basta chamar a função sem argumento algum, e o conteúdo volta a ser mostrado na tela.

```
> sink()
```

A função recebe tem ainda outros argumentos que podem controlar o envio de conteúdo para o arquivo. Por exemplo, o argumento `echo` recebe os valores `TRUE` ou `FALSE` indicando se os comandos devem ser incluídos no arquivo, ou somente os resultados dos comandos. Para mais detalhes veja `args(sink)` e `help(sink)`.

**Outras ferramentas para redirecionamento de conteúdo texto** A função `sink()` redireciona as saídas para um arquivo em formato texto. Há ainda outras funções que podem redirecionar as saídas em outros formatos. Alguns (mas não todos!) exemplo são citados a seguir.

`xtable()` do pacote (xtable) prepara tabelas em LaTeX

`HTML()` do pacote (R2HTML) e diversas outras funções deste pacote preparam saídas em HTML

e `html()` do pacote **Hmisc** preparam, respectivamente, saídas em LaTeXe HTML.

## 30.2 Gráficos

**Abrindo e redirecionando janelas gráficas** A janela gráfica é tipicamente aberta quando o usuário chama alguma função que produza um gráfico. Além desta forma, ela também pode ser aberta em branco quando o usuário chama a função de parâmetros gráficos `par()` ou por um dos seguintes comandos:

`x11()` no LINUX/UNIX

`indows()` no Windows

`quartz()` no Macintosh

Para fechar a janela gráfica usamos:

```
> dev.off()
```

Da mesma forma que `sink()` redireciona conteúdo texto, os função listadas a seguir redirecionam para os respectivos formatos gráficos.

- `postscript()`
- `pdf()`
- `png()`
- `jpeg()`

Existem ainda outros dispositivos que podem ser específicos de uma detarminada plataforma (sistema operacional) Cada uma desta funções recebe argumentos específicos, mas todas elas recebem um argumento obrigatório, o nome do arquivo para onde o gráfico deve ser enviado. Os comandos a seguir exemplificam o uso de uma destas funções para gerar um arquivo do tipo **.jpg** que chamamos de `exemplohist.jpg` contendo um histograma de um conjunto de dados.

```
> jpeg("exemplohist.jpg")
> hist(rexp(150, rate=5))
> dev.off()
```

Duas observações importantes:

1. é obrigatório o uso de `dev.off()` ao final para "fechar"o arquivo
2. a maioria dos dispositivos gera apenas 1 (um) gráfico por arquivo sendo necessário portanto gerar um arquivo para cada gráfico desejado.

**Múltiplas janelas gráficas** É possível abrir váriоa janelas gráficas ao mesmo tempo, ou seja, dispositivos ("devices") gráficos múltiplos. Para abri-los basta usar as funções mencionadas acima (por ex. `x11()` no LINUX) Neste caso uma das janelas será a "ativa"onde novos gráficos serão produzidos e as demais ficam "inativas". Há funções para controlar o comportamento destas janelas

- `dev.list()` lista os dispositivos gráficos
- `dev.next()` torna ativo o próximo dispositivo gráfico
- `dev.prev()` torna ativo o dispositivo gráfico anterior
- `dev.set(which=k)` torna ativo o $k$-ésimo dispositivo gráfico
- `dev.copy(device, ..., which=k)` e `dev.print(device, ..., which=k)` redirecionam o conteúdo do dispositivo gráfico ativo para impressora ou arquivo.
- `graphics.off()` fecha todos os dispositivos gráficos que estão abertos

Por exemplo, suponha que voce esteja com uma janela gráfica aberta e queira enviar o gráfico que está sendo mostrado na tela (na janela ativa) para um arquivo **meugrafico.jpeg**. Para isto pode usar os comandos:

```
> dev.copy(jpeg, file="meugrafico.jpeg")
> dev.off()
```

# 31 R, ambiente e o sistema de arquivos

O R pode interagir com o sistema de arquivos e o sistema operacional. Nesta seção vamos ver algumas das funcionalidades que informam sobre o ambiente de trabalho no R e também utilidades que podem facilitar o manuseio do programa.

Algumas implementações podem ser específicas para cada sistema operacional (SO). Por exemplo o diretório de trabalho ("workspace") pode ser definido via menu no Windows. Entretando vamos aqui dar preferência a funções que independem do SO. Os exemplos a seguir foram rodados em LINUX mas também podem ser usados em outros SO.

## 31.1 Ambiente de trabalho

Informações detalhadas sobre a versão do R e plataforma (sistema operacional) são retornadas pelo objeto abaixo. Note que é sempre útil informar a saída deste objeto quando utilizando listas de emails do projeto. As saídas retornadas na forma de uma `list` podem ainda ser úteis para escrever programas/rotinas que dependam destas informações

```
> R.version
```

```
               _
platform       i686-pc-linux-gnu
arch           i686
os             linux-gnu
system         i686, linux-gnu
status         Patched
major          2
minor          4.1
year           2007
month          02
day            17
svn rev        40757
language       R
version.string R version 2.4.1 Patched (2007-02-17 r40757)
```

Outros comandos relevantes sobre o sistema e recursos, cuja saída não mostramos aqui incluem:

- `getRversion()`retorna string com a versão do R.
- `.Platform`retorna ista com detalhes sobre a plataforma onde o R foi compilado, disponibilizando informação para trechos de código dependentes de informações sobre o sistema operacional.
- `Sys.info()`lista com informações dosbre o sistema e usuário.
- `.Machine`detalhes sobre atirimética usada, tal como manor e maior representação de números, etc, etc.

Outro comando útil é `SessionInfo()` que informa sobre o sistema operacional e *locales* (linguagem utilizada pelo sistema), a versão do R, pacotes carregados e e também os recursos (pacotes) disponíveis. As saídas das funções mencionadas podem ser usada quando informando/reportando problemas encontrados em aplicações e/ou quando escrevendo funções que possuam funcionalidades e opções que dependam destas informações.

```
> sessionInfo()
```

```
R version 2.4.1 Patched (2007-02-17 r40757)
i686-pc-linux-gnu

locale:
LC_CTYPE=pt_BR;LC_NUMERIC=C;LC_TIME=pt_BR;LC_COLLATE=pt_BR;LC_MONETARY=pt_BR;LC_MESSAGES=pt_BR

attached base packages:
[1] "tools"     "stats"     "graphics"  "grDevices" "utils"     "datasets"
[7] "methods"   "base"
```

## 31.2 Área de trabalho

Ao iniciar o R é aberta ou iniciada uma área de trabalho ("workspace") onde os objetos desta sessão poderão ser gravados. A localização "default"desta área de trabalho depende do sistema operacional, permissões etc. Por exemplo, no LINUX é o diretório de onde o R foi iniciado. No Windows é um diretório onde o R foi instalado.

Nos comandos a seguir mostramos como verificar qual o diretório de trabalho sendo usado, guardamos esta informação num objeto, verificamos qual o diretório onde o R foi instalado e como mudar o diretório de trabalho.

```
> getwd()

[1] "/home/paulojus/DEST/aulasR/Rnw"

> wdir <- getwd()
> wdir

[1] "/home/paulojus/DEST/aulasR/Rnw"

> R.home()

[1] "/usr/local/lib/R"

> setwd(R.home())
> getwd()

[1] "/usr/local/lib/R"

> setwd("/home/paulojus")
> getwd()

[1] "/home/paulojus"
```

O R automaticamente mantém um diretório temporário para uso em cada sessão e dentro deste um arquivo. As funções a seguir mostram como obter o caminho e nome do diretório earquivo temporários.

```
> tempdir()

[1] "/tmp/RtmpThfhe4"

> tempfile()

[1] "/tmp/RtmpThfhe4/file2ad12792"
```

## 31.3 Manipulação de arquivos e diretórios

Há uma diversidade de funções para interagir com o diretórios e arquivos. Por exemplo `dir()` vai listar o conteúdo do diretório, e possui vários argumentos para seleção. Informações sobre cada elemento do diretório podem ser obtidas com `file.info()`

```
> getwd()

[1] "/home/paulojus"

> dir("../")

[1] "paulojus" "shimakur"

> setwd(R.home())
> dir()

 [1] "bin"          "COPYING"      "doc"          "etc"          "include"
 [6] "lib"          "library"      "modules"      "NEWS"         "share"
[11] "SVN-REVISION"

> args(dir)

function (path = ".", pattern = NULL, all.files = FALSE, full.names = FALSE,
    recursive = FALSE)
NULL

> file.info("bin")

    size isdir mode               mtime               ctime               atime uid
bin 4096  TRUE 2755 2007-02-19 16:15:48 2007-02-19 16:15:48 2007-04-11 07:39:41   0
    gid uname grname
bin  50  root  staff

> file.info("bin")$isdir

[1] TRUE

> dir(path = "bin")

 [1] "BATCH"         "build"         "check"         "COMPILE"       "config"
 [6] "exec"          "f77_f2c"       "INSTALL"       "javareconf"    "libtool"
[11] "LINK"          "mkinstalldirs" "pager"         "R"             "Rcmd"
[16] "Rd2dvi"        "Rd2txt"        "Rdconv"        "Rdiff"         "REMOVE"
[21] "Rprof"         "Sd2Rd"         "SHLIB"         "Stangle"       "Sweave"
[26] "texi2dvi"

> dir(pattern = "COPY")

[1] "COPYING"

> dir(path = "doc")
```

```
 [1] "AUTHORS"          "COPYING"          "COPYING.LIB"      "COPYRIGHTS"
 [5] "CRAN_mirrors.csv" "FAQ"              "html"             "KEYWORDS"
 [9] "KEYWORDS.db"      "manual"           "NEWS"             "RESOURCES"
[13] "THANKS"
```

```
> dir(path = "doc", full = TRUE)
```

```
 [1] "doc/AUTHORS"          "doc/COPYING"          "doc/COPYING.LIB"
 [4] "doc/COPYRIGHTS"       "doc/CRAN_mirrors.csv" "doc/FAQ"
 [7] "doc/html"             "doc/KEYWORDS"         "doc/KEYWORDS.db"
[10] "doc/manual"           "doc/NEWS"             "doc/RESOURCES"
[13] "doc/THANKS"
```

É possivel efetuar operações do sistema operacional tais como criar, mover, copiar e remover arquivos e/ou diretórios a partir do R.

```
> file.exists("foo.txt")
```

```
[1] FALSE
```

```
> file.create("foo.txt")
```

```
[1] FALSE
```

```
> file.exists("foo.txt")
```

```
[1] FALSE
```

```
> file.rename("foo.txt", "ap.txt")
```

```
[1] FALSE
```

```
> file.exists("foo.txt")
```

```
[1] FALSE
```

```
> file.exists(c("foo.txt", "ap.txt"))
```

```
[1] FALSE FALSE
```

```
> file.copy("ap.txt", "foo.txt")
```

```
[1] FALSE
```

```
> file.exists(c("foo.txt", "ap.txt"))
```

```
[1] FALSE FALSE
```

```
> file.remove("ap.txt")
```

```
[1] FALSE
```

```
> file.exists(c("foo.txt", "ap.txt"))
```

```
[1] FALSE FALSE
```

Da mesma forma é também possível criar e manipular diretórios. Note que a opção `recursive=TRUE` deve ser usada com muito cuidado pois apaga todo o conteúdo do diretório.

```
> getwd()

[1] "/usr/local/lib/R"

> dir.create("~/meu.dir")
> file.copy("foo.txt", "~/meu.dir")

[1] FALSE

> dir("~/meu.dir")

[1] "foo.txt"

> unlink("~/meu.dir", recursive = TRUE)
```

Os exemplos acima são na verdade funções que passam comandos para o sistema operacional, seja ele qual for. De forma mais geral comandos do sistema operacional podem ser executados diretamento do R com a função `system()`, mas a sintaxe do comando fica obviamente dependente do sistema operacional usado (linux, unix, Mac, etc). A seguir ilustramos comandos usados no LINUX. Uma opção interessante é dada pelo argumento `intern = TRUE` que faz com que o resultado do comando seja convertido num objeto do R, como no exemplo abaixo onde o objeto `mdir` para a ser um vetor de caracteres com nomes de diretório de trabalho e mais abaixo o objeto `arqs` é um vetor com os nomes de todos os arquivos existerntes no diretório de trabalho.

```
> system("pwd")
> mdir <- system("pwd", intern = TRUE)
> mdir
> system("mkdir FStest.dir")
> system("touch FStest.dir/arquivo.txt")
> system("ls FStest.dir")
> arqs <- system("ls d*.Rnw", intern = TRUE)
> arqs
> system("rm -rf FStest.dir")
```

# 32 Usando 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.

Os passos 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)
   ```

4. rodar a função `Sweave()` no seu documento com um comando do tipo:

   ```
   > Sweave("foo.Rnw")
   ```

   Fazendo isto o Sweave deve gerar um documento `foo.tex`. Se tudo correr bem voce deverá ver uma mensagem do seguinte tipo na tela:

   ```
   You can now run LaTeX on 'foo.tex'
   ```

   Caso outra mensagem apareça, que não esta, voce deve ter tido algum problema com seu documento. Leia a mensagem, revise o seu documento para encontrar o erro.

5. Compile e visualize o documento LaTeX de forma usual.

## 32.1 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, bastando digitar o comando a seguir (ou com 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*.

## 32.2 Exemplos de arquivos em Sweave

1. Arquivo visto durante a aula
2. Arquivo com o conteúdo da aula sobre distribuições de probabilidades. 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

## 32.3 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

# 33 Instalando e usando pacotes (*packages*) do R

O programa R é composto de 3 partes básicas:

1. o **R-base**, o "coração" do R que contém as funções principais disponíveis quando iniciamos o programa,

2. os **pacotes recomendados** (*recommended packages*) que são instalados junto com o R-base mas não são carregados quando iniciamos o programa. Por exemplo os pacotes MASS, lattice, nlme são pacotes recomendados – e há vários outros. Para usar as funções destes pacotes deve-se carregá-los antes com o comando `library()`. Por exemplo o comando `library(MASS)` carrega o pacote MASS.

3. os pacotes contribuídos (*contributed packages*) não são instalados junto com o R-base. Estes pacotes disponívies na página do R são *pacotes oficiais*. Estes pacotes adicionais fornecem funcionalidades específicas e para serem utilizados devem ser copiados, instalados e carregados, conforme explicado abaixo. Para ver a lista deste pacotes com uma descrição de cada um deles acesse a página do R e siga os links para CRAN e Package Sources.

Antes de instalar o pacote voce pode ver se ele já está instalado/disponível. Para isto inicie o R e digite o comando:

```
> require(NOME_DO_PACOTE)
```

Se ele retornar `T` é porque o pacote já está instalado/disponível e voce não precisa instalar. Se retornar `F` siga os passos a seguir.

A instalação e uso dos pacotes vai depender do seu sistema operacional e os privilégios que voce tem no seu sistema. Nas explicações a seguir assume-se que voce está em uma máquina conectada à internet. O comando mostrado vai copiar o arquivo para seu computador, instalar o pacote desejado e ao final perguntar se voce quer apagar o arquivo de instalação (responda `Y` (*yes*))

1. **Instalação em máquinas com Windows98 ou em máquinas NT/XP/LINUX com senha de administrador (instalação no sistema)**.
   Neste caso basta usar o comando `install.packages()` com o nome do pacote desejado entre aspas. Por exemplo para instalar o pacote `CircStats` digite:

   ```
   > install.packages('CircStats')
   ```

   O pacote vai ser instalado no sistema e ficar disponível para tudos os usuários. Para usar o pacote basta digitar `library(CircStats)` ou `require(CircStats)`.

2. **Instalação em máquinas NT/XP/LINUX na conta do usuário, sem senha de administrador (instalação na conta do usuário)**

Neste caso o usuário deve abrir um diretório para instalar o pacote e depois rodar o comando de instalação especificando este diretório. Por exemplo, suponha que voce queira instalar o pacote `CircStats` na sua conta no sistema Linux do LABEST. Basta seguir os seguintes passos.

1. Na linha de comando do Linux abra um diretório (se já não existir) para instalar os pacotes. Por exemplo, chame este diretório `Rpacks`:

   ```
   % mkdir -p ~/Rpacks
   ```

2. Inicie o R e na linha de comando do R digite:

```
> install.packages("CircStats", lib="~/Rpacks")
```

3. Neste caso o pacote vai ser instalado na área do usuário e para carregar o pacote digite:

```
> library(CircStats, lib="~/Rpacks")
```

**NOTA:** no Windows voce pode, alternativamente, instalar usando o menu do R selecionando a opção `PACKAGES - INSTALL FROM CRAN`.

## 33.1 Pacotes não-oficiais

Além dos pacotes contribuídos existem diversos pacotes *não-oficiais* disponívies em outros locais na web. Em geral o autor fornece instruções para instalação. As instruções gerais para instalação são as seguintes:

- **Linux:** Os pacotes para Linux em geral vem em um arquivo tipo `PACOTE.tar.gz` e são instalados com um dos comandos abaixo (use o primeiro se for administrador do sistema e o segundo como usuário comum).

```
R INSTALL PACOTE.tar.gz
ou
R INSTALL -l ~/Rpacks PACOTE.tar.gz
```

- **Windows:** No menu do R use a opção `PACKAGES - INSTALL FROM LOCAL .ZIP FILE`

# 34 Construindo pacotes

Os passos básicos para contrução de um pacote são listados a seguir.

1. Abra uma sessão do R e coloque na sua área de trabalho todas as funções e conjunto de dados que deseja incluir no pacote. Tome o cuidade de remover todos os demais objetos que não devem ser incluídos no pacote.

2. No "prompt"do R use `package.skeleton()` para gerar um diretório com a estrutura de diretórios mínima requirida para pacotes. Por exemplo, se o seu pacote for se chamar `meupack` use o comando abaixo. Certifique-se que o diretório a ser criado ainda não existe no seu diretório de trabalho.

   ```
   > package.skeleton(name="meupack")
   ```

3. No diretório criado voce vai encontrar:

   - O arquivo `DESCRIPTION` que contém uma descrição básica do seu pacote. Edite este arquivo tomando cuidado para não alterar a estrutura do mesmo
   - O subdiretório `data` que contém os conjuntos de dados que estavam em seu "workspace". Voce não precisa fazer nada neste diretório.
   - O subdiretório `man` que contém a documantação de seu pacote com um arquivo para cada função e conjunto de dados de seu pacote. Abra cada um dos arquivos em um editor de arquivos texto e edite a documentação, preservando o formato do arquivo.
   - O subdiretório `R` contém arquivos com as funções em R de seu pacote. Voce não precisa fazer nada neste diretório a menos que vá usar código compilado em seu pacote (ver mais detalhes a seguir).
   - O subdiretório `src` somente será usado caso o seu pacote vá usar códigos em C, C++ ou Fortran. Se este for o caso voce deve colocar neste subdiretório os arquivos fontes nestas linguagens.

4. Caso o seu pacote vá usar códigos em C, C++ ou Fortran coloque um arquivo com o nome `zzz.R` no sibdiretório `R` com o seguinte conteúdo

   ```
   ".First.lib" <- function(lib, pkg)
   {
     library.dynam("Embrapa", package = pkg, lib.loc = lib)
     return(invisible(0))
   }
   ```

5. Para testar o seu pacote voce pode usar na linha de comando:

   ```
   R CMD check meupack
   ```

6. Para montar o arquivo fonte `.tar.gz` de distribuição co pacote use o comando a seguir. O arquivo criando pode ser usado de forma padrão para instalar pacotes no R a partir do arquivo fonte do pacote.

   ```
   R CMD build meupack
   ```

Durante o curso foi demonstrado como construir pacotes no R. O pacote montado durante as aulas está disponível neste link e voce pode inspecionar o conteúdo para ver um exemplo de criação de pacote.

As passos listados aqui são bastante simplificados e são simplesmente o mínimo necessário para criação de pacotes. Diversos outros recursos estão disponíveis e para maiores e mais detalhadas informações sobre como construir pacotes consulte o manual *Writing R Extensions*.

# 35 Rodando o R dentro do xemacs

Esta página contém instruções sobre como rodar o programa estatístico R dentro do editor xemacs que tem versões disponíveis para LINUX e Windows. Para obter o xemacs vá em `http://www.xemacs.org`

Este procedimento permite um uso ágil do programa R com facilidades para gravar o arquivo texto com os comandos de uma sessão e uso das facilidades programadas no pacote ESS (Emacs Speaks Statistics) que é um complemento do editor xemacs.

Para utilizar esta funcionalidade deve-se seguir os seguintes passos:

1. Instalar o programa R. *(clique para baixar programa de instalação)*
   Para usuários do Windows assume-se aqui que o R esteja instalado em:
   `C:\ARQUIVOS DE PROGRAMAS\rw`

   Note que na instalação do R é sugerido um nome do diretório de instalação do tipo `rw2010`. Sugiro que voce mude para `rw` apanes para não ter que alterar a configuração abaixo toda vez que atualizar a sua versão do R.

2. Instalar o programa xemacs. As versões mais recentes já veem com o pacote ESS incluído. *(clique para baixar programa de instalação)*

3. Modifique a variável `PATH` do seu computador adicionando a ela o caminho para o diretório `bin` do R. No **Windows 98** isto é feito modificando o arquivo `C:\AUTOEXEC.BAT` inserindo a seguinte linha no final do arquivo
   `SET PATH=%PATH%;C:\ARQUIVOS DE PROGRAMA\rw\bin`
   No **Windows XP** isto é feito adicionado este diretório à esta variável de ambiente.

4. Inicie o programa xemacs e clique na barra de ferramentas em:
   `Options --> Edit init file`

5. Adicionar a seguinte linha:
   `(require 'ess-site)`

6. Gravar o arquivo e sair do xemacs.

7. Se usar o **Windows 98**: reinicialize o seu computador.

8. Tudo pronto! Para começar a utilizar basta iniciar o programa xemacs. Para iniciar o R dentro do xemacs use a combinação de teclas:
   `ESC SHIFT-X SHIFT-R`

9. Use sempre a extensão `.R` para os seus arquivos de comandos do R.

10. Lembre-se que voce pode usar `CTRL-X-2` para dividir a tela em duas.

# 36 Arquivos `.Rhistory`

O endereço `http://www.leg.ufpr.br/~paulojus/embrapa/history` possui alguns arquivos `.Rhistory` que foram criados durante o curso.

# 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.