

Uma Introdução ao Ambiente Estatístico R

Paulo Justiniano Ribeiro Junior

Última atualização: 10 de setembro de 2026

Estas notas¹ foram inicialmente escritas como material para um curso de *introdução ao sistema estatístico R* ministrado para profissionais da EMBRAPA em Brasília, 30/05 a 03/06 de 2005. Desde sua versão inicial o material foi modificado com adição, correção e inclusão de tópicos.

O objetivo é ilustrar o uso do R com ênfase na compreensão de aspectos básicos da linguagem, a estrutura e a forma de operar o programa. Nenhum método e/ou modelo estatístico em particular é discutido em detalhes, seja em seus fundamentos ou alternativas para análises. Métodos estatísticos são usados ao longo do texto com foco em ilustrar aspectos do uso da linguagem.

Na maior parte do texto assume-se familiaridade apenas com conceitos e métodos básicos de estatística. Alguns tópicos especializados são usados em algumas das Seções e, não sendo de interesse do leitor, podem ser deixados de lado sem prejuízo ao acompanhamento das demais partes do texto. Não será assumido nenhum conhecimento prévio do R. O curso foi preparado e ministrado em ambiente LINUX porém não faz uso de nenhum recurso específico deste sistema operacional. O material pode ser acompanhado utilizando o R instalado em outros sistemas operacionais, tal como Windows® ou Macintosh.

O texto começa com uma Seção que tem como objetivo "experimentar o R", que permite ter uma ideia de seus recursos e a forma de trabalhar com este programa. Sugere-se reproduzir e estudar os comandos indicados bem como inspecionar e interpretar os resultados produzidos por tais comandos, o que vai permitir uma familiaridade com aspectos básicos do uso do programa. Espera-se que ao final desta Seção o leitor se sinta à vontade para iniciar o programa e experimentar o seu uso em outros contextos de análises. Ao longo do material mais detalhes sobre o uso do programa R serão apresentados, na maior parte das vezes motivados por exemplos de análise de dados.

Para utilizar o R siga os seguintes passos:

1. inicie o R em seu computador;
2. você verá uma janela de comandos com o símbolo `>`; que é chamado de *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 ou seus próprios comandos.

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 `>`;

¹Estas notas estão disponíveis em formato HTML em <http://www.leg.ufpr.br/~paulojus/Rembrapa> e também em [arquivo no formato PDF](#).

- 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 ideia geral da aparência e forma de operação do programa. Os comandos abaixo motivam explicações sobre características básicas de linguagem e serão reproduzidos, comentados e discutidos com os participantes 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] -0.2068510 -1.2693623 0.5574482 2.0031405 0.3570333
> print(x)
[1] -0.2068510 -1.2693623 0.5574482 2.0031405 0.3570333
> print(x, dig=3)
[1] -0.207 -1.269 0.557 2.003 0.357
> y <- rnorm(x)
> y
[1] 0.3027666 0.1811900 0.9989794 -0.9324467 -0.2098432
> 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)** não 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 você 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 ideias e conceitos: *objeto, atribuição de valores, vetores, impressão de objetos, função, argumentos de funções, defaults, geração de números aleatórios e controle de semente*.

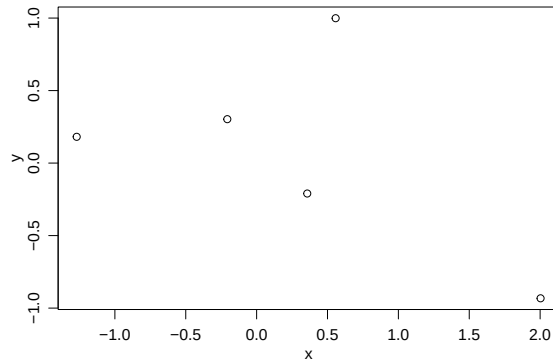
Agora vamos colocar num gráfico os pontos gerados usando o comando

```
> 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 especificadas 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. Os argumentos básicos são mostrados a seguir.

```
> args(plot.default)
function (x, y = NULL, type = "p", xlim = NULL, ylim = NULL,
  log = "", lim2 = FALSE, main = NULL, sub = NULL, xlab = NULL,
  ylab = NULL, ann = par("ann"), axes = TRUE, frame.plot = axes,
  panel.first = NULL, panel.last = NULL, asp = NA, xgap.axis = NA,
  ygap.axis = NA, ...)
NULL
```

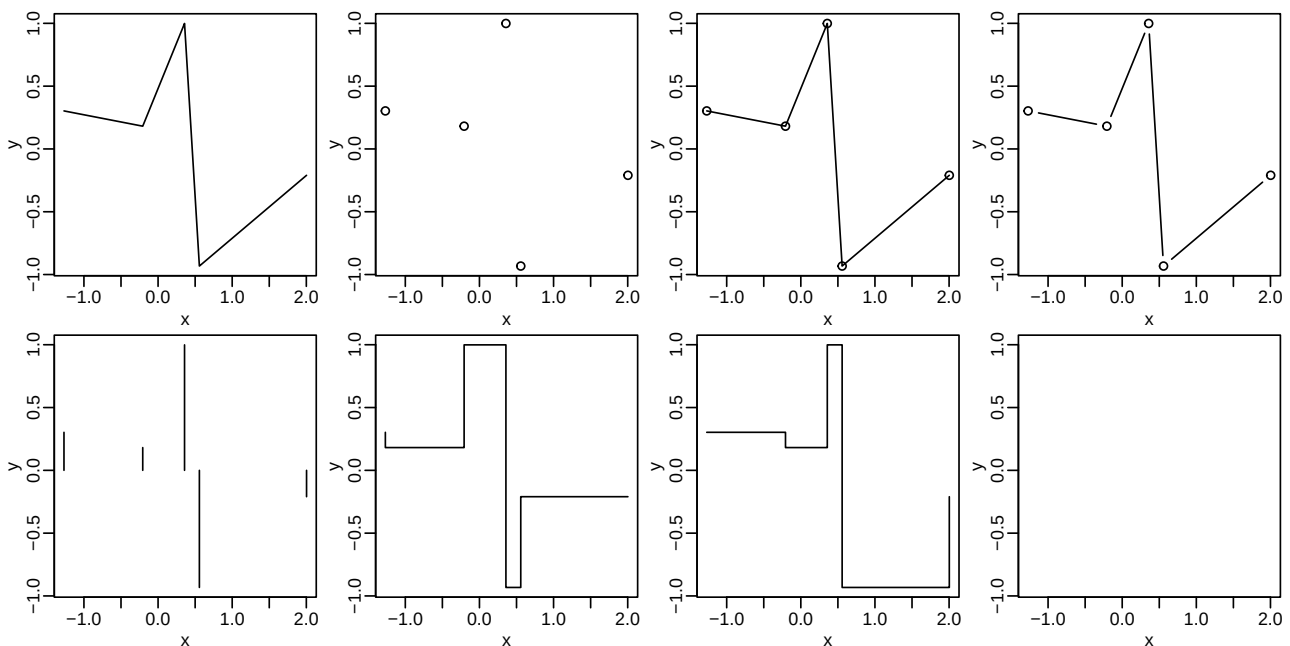


Para ilustração, no exemplo a seguir mostramos o uso do argumento `type`. Para facilitar esta ilustração vamos primeiro ordenar os valores de `x` e `y` na sequência crescente dos valores de `x`.

```
> 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.

```
> par(mfrow=c(2,4), mar=c(2.5,2.5,0.3,0.3), mgp=c(1.5, 0.5, 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))
```

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

```

> x
[1] -1.2693623 -0.2068510  0.3570333  0.5574482  2.0031405
> x[1]
[1] -1.269362
> x[3]
[1] 0.3570333
> x[2:4]
[1] -0.2068510  0.3570333  0.5574482
> round(x, dig=1)
[1] -1.3 -0.2  0.4  0.6  2.0
> ceiling(x)
[1] -1  0  1  1  3
> floor(x)
[1] -2 -1  0  0  2
> trunc(x)
[1] -1  0  0  0  2

```

Os objetos existentes 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`. Inspeccionando 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 0.6493015 1.500000
2  2 4.0734352 1.707107
3  3 -0.3563404 1.866025
4  4 3.7202244 2.000000
5  5 6.0038068 2.118034
6  6 6.9167381 2.224745
7  7 5.1757240 2.322876
8  8 5.9294215 2.414214
9  9 5.1512831 2.500000
10 10 13.5005700 2.581139
11 11 12.9993392 2.658312
12 12 12.7587511 2.732051

```

```

13 13 13.9086405 2.802776
14 14 11.0851592 2.870829
15 15 14.4183039 2.936492
16 16 14.4569368 3.000000
17 17 14.3904028 3.061553
18 18 17.3722979 3.121320
19 19 20.9927450 3.179449
20 20 13.3542118 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
-5.0262	-1.5341	0.1448	1.6452	4.1258

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	0.36916	1.14083	0.324	0.75
x	0.90056	0.09524	9.456	2.1e-08 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.456 on 18 degrees of freedom

Multiple R-squared: 0.8324, Adjusted R-squared: 0.8231

F-statistic: 89.42 on 1 and 18 DF, p-value: 2.097e-08

```

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

```

Call:

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

Weighted Residuals:

Min	1Q	Median	3Q	Max
-1.74670	-0.62110	0.05046	0.60417	1.60510

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	0.13682	0.87043	0.157	0.877
x	0.92207	0.08875	10.389	4.95e-09 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.9619 on 18 degrees of freedom

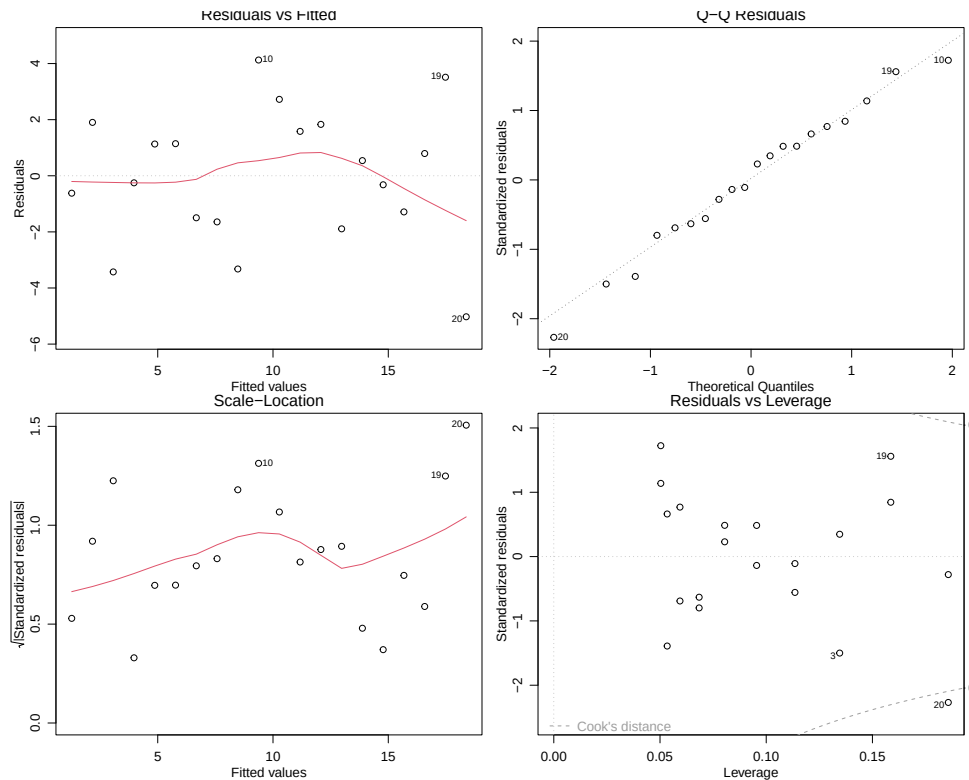
Multiple R-squared: 0.8571, Adjusted R-squared: 0.8491

F-statistic: 107.9 on 1 and 18 DF, p-value: 4.948e-09

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:knitr"    "package:stats"    "package:graphics"
[5] "package:grDevices" "package:utils"    "package:datasets" "package:methods"
[9] "Autoloads"       "package:base"

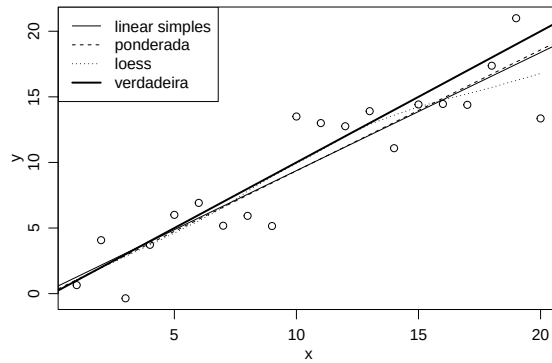
> attach(dummy)
> search()
[1] ".GlobalEnv"      "dummy"            "package:knitr"    "package:stats"
[5] "package:graphics" "package:grDevices" "package:utils"    "package:datasets"
[9] "package:methods" "Autoloads"         "package:base"
```

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("topleft", c("linear simples", "ponderada", "loess", "verdadeira"), lty=c(1,2,3,1), lwd=c(1,1,1,2))
```

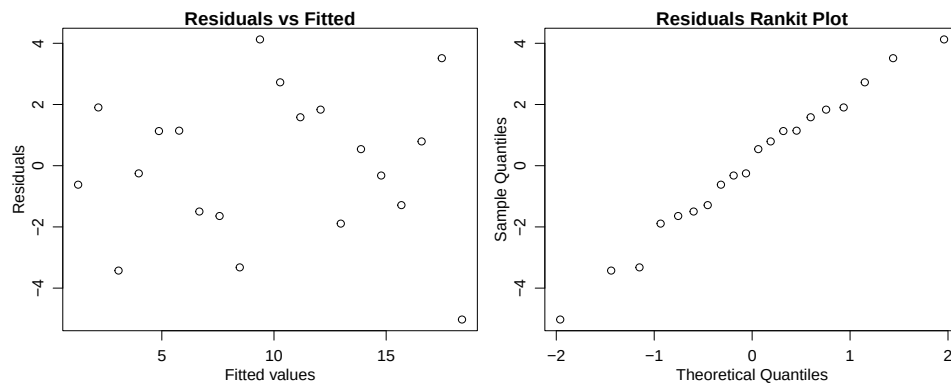
Ao final das análises removemos o objeto *dummy* do caminho de procura.

```
> 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 você 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")
```

Vamos importar os dados como um *data.frame* e inspecionar seu conteúdo listando as seis primeiras linhas do objeto criado com o comando `head()`. Há 5 experimentos (coluna *Expt*) e cada um com 20 “rodadas”(coluna *Run*) e *s1* é o valor medido da velocidade da luz numa escala apropriada

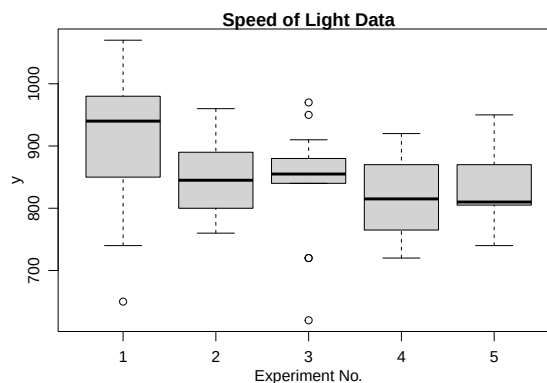
```
> mm <- read.table("http://www.leg.ufpr.br/~paulojus/dados/morley.tab")
> head(mm)
  Expt Run Speed
001   1  1   850
002   1  2   740
003   1  3   900
004   1  4  1070
005   1  5   930
006   1  6   850
```

Devemos definir `Expt` e `Run` como fatores e 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	1.105	0.36321
Expt	4	94514	23629	4.378	0.00307 **
Residuals	76	410166	5397		

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

> names(fm)
```

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

```
> fm$coef
```

	Run2	Run3	Run4	Run5	Run6
(Intercept)	-5.20000e+01	-2.80000e+01	6.00000e+00	-7.60000e+01	-1.04000e+02
Run7	-4.00000e+01	-1.00000e+01	-3.80000e+01	4.00000e+00	-8.70878e-14
Run13	-9.40000e+01	-6.00000e+01	-6.60000e+01	-6.00000e+00	-3.80000e+01
Run19	-4.40000e+01	-5.30000e+01	-6.40000e+01	-8.85000e+01	-7.75000e+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 segundo 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))
      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,] -3.510833e-17 -0.5345225  1.755417e-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 estiver claro experimente para cada uma destas examinar a matriz 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)
```

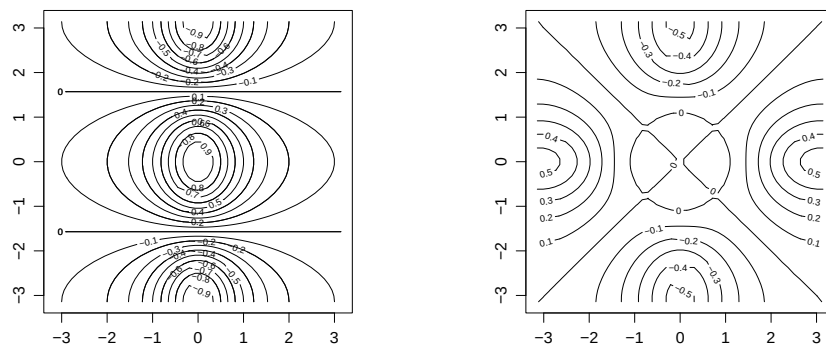
Vamos agora ver alguns gráficos gerados pelas funções `contour()` e `image()`.

No próximo exemplo `x` é um vetor de 50 valores igualmente espaçados no intervalo $[-\pi, \pi]$. O objeto `y` é uma cópia de `x`. O objeto `f` é uma matriz 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))
```

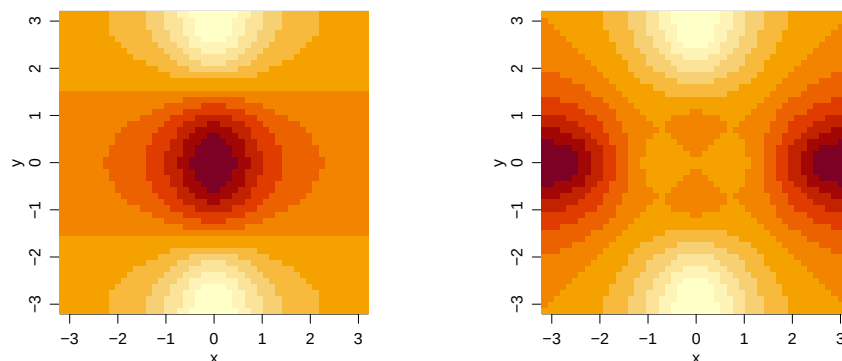
Agora gravamos parâmetros gráficos definindo a região gráfica como quadrada e fazemos um mapa de contorno de `f`. Depois adicionamos mais linhas para melhor visualização. `fa` é a “parte assimétrica” e `t()` é transposição. Ao final restauramos os parâmetros gráficos iniciais.

```
> 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.

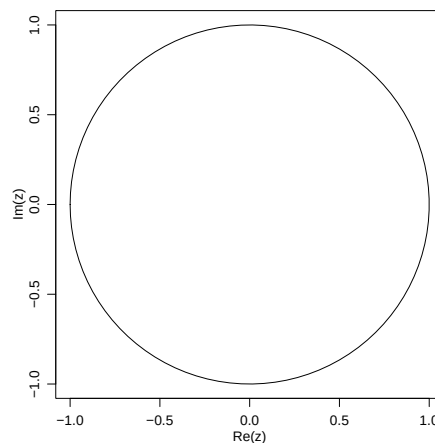
```
> objects()
[1] "f"      "fa"      "mds"      "mm"      "oldpar" "x"      "y"
> rm(x, y, f, fa)
```

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(1i*th)
```

Plotando a sequência de complexos com a parte imaginária *versus* a real obtém-se 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. Depois mapeamos qualquer externo ao círculo no seu recíproco:

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



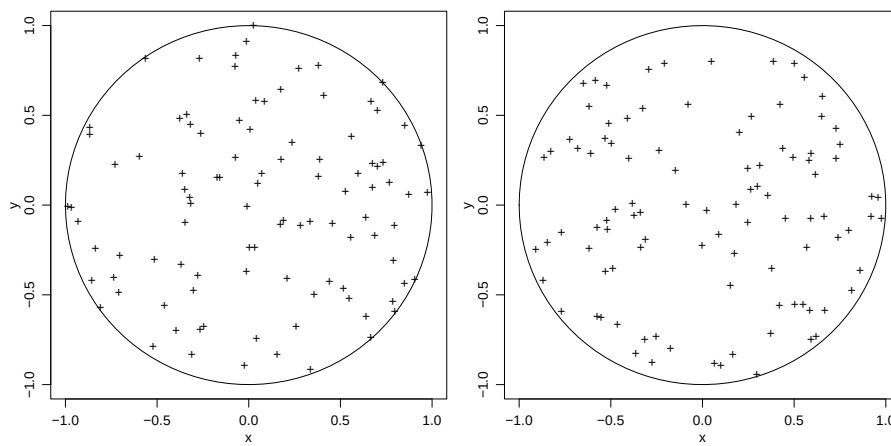
Desta forma todos os pontos estão dentro do círculo unitário, mas a distribuição não é uniforme. Um segundo método usa a distribuição uniforme. Os pontos devem estar melhor distribuídos sobre o círculo

```
> 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)*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 Computação estatística com o sistema R

Nesta seção iremos seguir a apresentação disponível no arquivo <http://www.leg.ufpr.br/Rembrapa/estcompR.pdf>

3 Introdução

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

3.1 O projeto R

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

<http://www.r-project.org>.

O programa pode ser copiado livremente pela internet. Há alguns espelhos (*mirrors*) brasileiros da área de *downloads* do programa chamada de CRAN (Comprehensive R Archive Network), entre eles um situado no C3SL/UFPR que pode ser acessado em <https://cran-r.c3sl.ufpr.br>.

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

3.2 Utilizando o R

Siga os seguintes passos.

1. Inicie o R em seu computador. Para iniciar o R no LINUX basta digitar R na linha de comando de um terminal, como a seguir

```
paulojus@pjLEG:~$ R
```

```
R version 4.6.1 (2026-06-24) -- "Happy Hop"
Copyright (C) 2026 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu
```

```
R é um software livre e vem sem GARANTIA ALGUMA.
Você pode redistribuí-lo sob certas circunstâncias.
Digite 'license()' ou 'licence()' para detalhes de distribuição.
```

```
R é um projeto colaborativo com muitos contribuidores.
Digite 'contributors()' para obter mais informações e
'citation()' para saber como citar o R ou pacotes do R em publicações.
```

```
Digite 'demo()' para demonstrações, 'help()' para o sistema on-line de ajuda,
ou 'help.start()' para abrir o sistema de ajuda em HTML no seu navegador.
Digite 'q()' para sair do R.
```

```
[Área de trabalho anterior carregada]
```

```
>
```

2. Você verá no final o símbolo > indicando onde você irá digitar comandos do R. Este é o *prompt* do R indicando que o programa está pronto para receber seus comandos.

Para cada comando digitado o R irá executar uma ação, o que caracteriza o R como uma **linguagem interpretada**.

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 e suas saídas são sempre mostrados em fontes do tipo `typewriter` como `esta`;
- linhas iniciadas pelo símbolo `#` são comentários e são ignoradas pelo R.

3.3 Cartão de referência

Para operar o R é necessário conhecer e digitar comandos. Isto pode trazer alguma dificuldade no início 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 duas opções (há outras na internet):

- **Cartão de Referência** em formato PDF preparado por Jonathan Baron.
- **Cartão de Referência** em formato PDF preparado por Tom Short.

3.4 Rcmdr - Pacote “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 e possibilidades.

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.

Atenção: 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](#).

3.5 Rodando o R pela web

Ao longo do tempo foram propostos e disponibilizados diversos mecanismos que permitem rodar o R pela web, sem que você precise ter o programa instalado no seu computador. Para usá-lo basta estar conectado na internet.

Algumas opções, dentre várias disponíveis, são listadas a seguir.

- **webR**: o interpretador do R compilado para *WebAssembly*, rodando inteiramente no navegador, sem depender de nenhum servidor.
- **R Snippets**: roda código R diretamente no navegador, com milhares de pacotes já disponíveis, sem necessidade de instalação.
- **Posit Cloud**: ambiente de desenvolvimento (RStudio) completo rodando no navegador, mantido pela Posit.
- **Google Colab**: voltado principalmente para Python, mas também permite rodar R através de *notebooks* específicos.

4 Aritmética e Objetos

4.1 Operações aritméticas

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

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

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

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

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

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

```
> sin(pi)
[1] 1.224647e-16
```

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

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 logaritmo natural
log10() log2()	logaritmo base-10 e base-2
gamma()	função Gamma de Euler
factorial()	fatorial ($n!$)
choose()	número de combinações $\left(\frac{n!}{x!(n-x)!}\right)$
combn()	todas as combinações de um certo número de elementos

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

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

4.2 Valores faltantes e especiais

Vimos nos exemplos anteriores que `pi` é um valor especial, que armazena o valor desta constante matemática. O número de Euler e é obtido com `exp(1)`.

```
> pi
[1] 3.141593
> exp(1)
[1] 2.718282
```

Existem ainda alguns outros valores especiais usados pelo R:

- `NA` : *Not Available*, denota dados faltantes. Note que deve utilizar maiúsculas.
- `NaN` : *Not a Number*, 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.

```
> sqrt(-1)
Warning in sqrt(-1): NaNs produzidos
[1] NaN
> Inf/10
[1] Inf
> Inf/Inf
[1] NaN
> c(-1,0,1)/0
[1] -Inf NaN Inf
```

No último comando usamos a função `c()` para criar um *vetor* de elementos -1 , 0 e 1 e a operação é dita ser *vetorial* por dividir cada um desses valores por 0 .

4.3 Objetos

O R é uma linguagem funcional e vetorial que manipula *objetos*, no sentido em que variáveis, dados, matrizes, funções, etc, são armazenados na memória ativa do computador na forma de *objetos*. Por exemplo, se um objeto `x` tem o valor 10 , ao digitarmos o seu nome, o programa exibe o valor do objeto:

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

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

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

```
> x <- sin(pi)
> x
[1] 1.224647e-16
> sin(pi) -> x
> x
[1] 1.224647e-16
> x = sin(pi)
> x
[1] 1.224647e-16
```

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

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

Note que ao atribuir um valor a um objeto o programa não imprime nada na tela. Digitando o nome do objeto o programa imprime seu conteúdo na tela. Digitando uma operação aritmética, sem atribuir o resultado a um objeto, faz com que o programa imprima o resultado na tela. Nomes de variáveis devem começar com uma letra e podem conter letras, números e pontos. Um fato importante é que o R distingue letras maiúsculas e minúsculas nos nomes dos objetos, por exemplo `dados`, `Dados` e `DADOS` serão interpretados como nomes de três objetos diferentes pela linguagem.

DICA: tente atribuir nomes que tenham um significado lógico, relacionado ao trabalho e dados em questão. Isto facilita lidar com um grande número de objetos. Ter nomes como `a1` até `a20` pode causar confusão. A seguir estão alguns exemplos válidos ...

```
> x <- 25
> x * sqrt(x) -> x1
> x2.1 <- sin(x1)
> xsq <- x2.1**2 + x2.1**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 parêntese 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. No último caso, `sqrt(x) <- 10` tem a forma de uma *função de substituição* (`nome<-`), mecanismo usado por comandos como `names(x) <- ...` ou `dim(x) <- ...`; como não existe uma função `sqrt<-`, o R retorna erro dizendo que não conseguiu encontrar esta função.

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

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

Nomes reservados: O R, como qualquer outra linguagem, possui nomes reservados, isto é, 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 `any()` verifica a presença de *algum* valor que satisfaça a condição e `all()` verifica se *todos* os valores satisfazem a condição.

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

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

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

Agora vamos ver outros valores especiais.

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

5 Tipos de objetos

Os tipos básicos de objetos do R são:

- 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()`.

5.1 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.

No exemplo a seguir mostramos algumas operações com vetores. A função `c()` ("c" de concatenar) é usada para criar um vetor. Os colchetes `[ ]` são usados para indicar seleção de elementos. As funções `rep()`, `seq()` e o símbolo `:` são usadas para facilitar a criação de vetores que tenham alguma lei de formação.

```
> x1 <- 10
> x1
[1] 10
> x2 <- c(1, 3, 6)
> x2
[1] 1 3 6
> x2[1]
[1] 1
> x2[2]
[1] 3
> length(x2)
[1] 3
> is.vector(x2)
[1] TRUE
> is.matrix(x2)
[1] FALSE
> is.numeric(x2)
[1] TRUE
> is.character(x2)
[1] FALSE
> (x3 <- 1:10)
[1] 1 2 3 4 5 6 7 8 9 10
```

```

> (x4 <- seq(0,1, by=0.1))
[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))
[1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> 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))
[1] 1 1 1 1 1
> (x7 <- rep(c(1, 2), c(3, 5)))
[1] 1 1 1 2 2 2 2 2
> (x8 <- rep(1:3, rep(5,3)))
[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 de caracteres.

```

> 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)))
[1] 71 71 67 66 76 87 66 87 58 73

```

```

> sum(x9)
[1] 722
> mean(x9)
[1] 72.2
> var(x9)
[1] 84.62222
> min(x9)
[1] 58
> max(x9)
[1] 87
> summary(x9)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 58.00  66.25   71.00   72.20   75.25   87.00
> fivenum(x9)
[1] 58 66 71 76 87

```

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. Os comandos a seguir 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). No terceiro comando são extraídos elementos de posições arbitrárias.

```

> x9[1:3]
[1] 71 71 67
> x9[2*(1:5)]
[1] 71 66 87 87 73
> x9[c(2,4,7)]
[1] 71 66 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] FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE  TRUE FALSE  TRUE
> as.numeric(ind.72)
[1] 0 0 0 0 1 1 0 1 0 1
> x10 <- x9[ind.72]
> x10
[1] 76 87 87 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 são `paste()` e `paste0()`. 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] "fulano-2"
> paste("fulano", 2, sep="")
[1] "fulano2"
> paste0("fulano", 2)
[1] "fulano2"
> paste(letters[1:8], 2, sep="")
[1] "a2" "b2" "c2" "d2" "e2" "f2" "g2" "h2"
```

Portanto `paste0()` é um caso especial de `paste()` quando não se deseja caracter separador. 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 neste caso incluem os níveis do fator. Nos comandos abaixo esta distinção fica mais clara onde um vetor é criado inicialmente como *caracteres* e depois convertido para *fator*.

```
> estados <- c("PR", "SC", "RS")
> estados
[1] "PR" "SC" "RS"
> class(estados)
[1] "character"
> attributes(estados)
NULL
> estados <- as.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"
```

Fatores podem ser convertidos e interpretados como números. A associação é feita por ordem alfabética por *default*. Examine o exemplos a seguir

```
> estados <- c("PR", "SC", "RS")
> as.numeric(estados)
Warning: NAs introduzidos por coerção
[1] NA NA NA
> estados <- factor(c("PR", "SC", "RS"))
> as.numeric(estados)
[1] 1 3 2
```

Um fato relevante a respeito da manipulação de um 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, a menos que a opção **drop** seja usada.

```
> estados.sel <- estados[-3]
> estados.sel
[1] PR SC
Levels: PR RS SC
> estados.seldrop <- estados[-3, drop=TRUE]
> estados.seldrop
[1] PR SC
Levels: PR SC
```

Da mesma forma pode-se criar um vetor e definir para ele níveis, mesmo que estes níveis não estejam entre os elementos atualmente existentes no vetor. Note no exemplo abaixo 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

```

Reagrupando níveis de fatores Pode-se reagrupar níveis de um fator usando-se `levels`. No exemplo a seguir começamos definindo um novo fator como cópia do original, listando os níveis originais dos fatores e depois o modificando de forma a agrupar os estados por regiões respeitando a ordem dos níveis originais.

```

> regioao <- sesul
> levels(regiao)
[1] "PR" "SC" "RS" "MG" "SP" "RJ" "ES"
> levels(regiao) <- c("Sul", "Sul", "Sul", "Sudeste", "Sudeste", "Sudeste", "Sudeste")
> regioao
[1] Sul      Sul      Sul      Sul      Sul      Sudeste Sul      Sudeste Sudeste
[10] Sul      Sudeste Sudeste
Levels: Sul Sudeste
> table(regiao)
regiao
      Sul Sudeste
      7       5

```

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 que 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] 68 78 62 61 65 64 72
> rev(vec)
[1] 72 64 65 61 62 78 68
> sort(vec)
[1] 61 62 64 65 68 72 78
> order(vec)
[1] 4 3 6 5 1 7 2
> vec[order(vec)]
[1] 61 62 64 65 68 72 78
> rank(vec)
[1] 5 7 2 1 4 3 6
> vec %% 5
[1] 3 3 2 1 0 4 2
> vec %/% 5
[1] 13 15 12 12 13 12 14
```

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

```
> which(vec > 70)
[1] 2 7
> which.max(vec)
[1] 2
> which.min(vec)
[1] 4
```

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

```
> vec
[1] 68 78 62 61 65 64 72
> vec[-5]      ## exclui o 5o elemento
[1] 68 78 62 61 64 72
> vec[-(2:4)]  ## exclui o 2 e o 4o elemento
[1] 68 65 64 72
```

5.2 Matrizes

Matrizes são montadas a partir da reorganização de elementos de um vetor em linhas e colunas. Por *default* a matriz 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 matriz 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 *totaux marginaux*, 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 esta 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

```

Estes valores podem ainda ser representados por proporções, com opções para total por linha, por coluna ou geral da matriz, usando a função `prop.table()`.

```

> prop.table(m1, margin = 1)

```

```

      C1      C2      C3
L1 0.06666667 0.3333333 0.6000000
L2 0.11111111 0.3333333 0.5555556
L3 0.14285714 0.3333333 0.5238095
L4 0.16666667 0.3333333 0.5000000
> prop.table(m1, margin = 2)
      C1      C2      C3
L1 0.1 0.1923077 0.2142857
L2 0.2 0.2307692 0.2380952
L3 0.3 0.2692308 0.2619048
L4 0.4 0.3076923 0.2857143
> prop.table(m1)
      C1      C2      C3
L1 0.01282051 0.06410256 0.1153846
L2 0.02564103 0.07692308 0.1282051
L3 0.03846154 0.08974359 0.1410256
L4 0.05128205 0.10256410 0.1538462

```

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 $X'X$ enquanto `crossprod(X,Y)` retorna $X'Y$. 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 1 e considere obter os coeficientes da regressão linear dados por:

$$\hat{\beta} = (X'X)^{-1}X'y \quad , \quad (1)$$

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

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

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 (ii) usando uma forma computacionalmente mais adequada de obter o mesmo resultado.

```

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

```

```
> drop(solve(crossprod(X), crossprod(X,y)))
[1] 8.06 2.78
```

Notas:

1. existem formas ainda mais computacionalmente eficientes de obter o resultado acima no R, como por exemplo usando decomposições como a de Cholesky (`chol()`) e **QR** (`qr()`), mas o uso dessas 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.
3. As operações retornam uma matriz de dimensão 2×1 . A função `drop` reduz a dimensão do objeto (se possível) para um vetor de comprimento 2.

5.3 Arrays

O conceito de *array* generaliza a ideia de matriz. Enquanto em uma matriz 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

```

5.4 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.

A seguir criamos um *data-frame*, vemos os nomes de suas colunas e extraímos cada coluna.

```

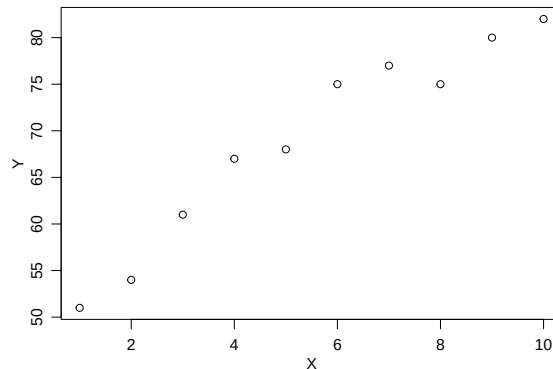
> d1 <- data.frame(X = 1:10, Y = c(51, 54, 61, 67, 68, 75, 77, 75, 80, 82))
> d1
   X  Y
1  1 51
2  2 54
3  3 61
4  4 67
5  5 68
6  6 75
7  7 77
8  8 75
9  9 80
10 10 82
> names(d1)
[1] "X" "Y"
> d1$X
[1] 1  2  3  4  5  6  7  8  9 10
> d1$Y

```

```
[1] 51 54 61 67 68 75 77 75 80 82
```

Para fazer um gráfico com os dados pode-se fazer

```
> plot(d1)
```



Se houverem mais de duas colunas o comando `plot()` irá produzir diagramas de dispersão duas a duas. Para especificar as colunas pode-se usar um dos comandos a seguir que produzem o mesmo gráfico visto anteriormente.

```
> plot(d1$X, d1$Y)
> plot(X ~ Y, data = d1)
> plot(Y ~ X, data = d1)
> with(d1, plot(X, Y))
```

Vejamos a seguir uma estrutura de dados com grupos.

```
> d2 <- data.frame(Y= c(10+rmnorm(5, sd=2), 16+rmnorm(5, sd=2), 14+rmnorm(5, sd=2)))
> d2$lev <- gl(3,5)
> d2
```

	Y	lev
1	9.747865	1
2	11.329726	1
3	8.700255	1
4	8.348289	1
5	11.437184	1
6	16.776116	2
7	16.844483	2
8	16.648475	2
9	18.716013	2
10	15.377763	2
11	16.296121	3
12	15.376397	3
13	12.337456	3
14	10.785550	3
15	14.901564	3

A função `gl()` é um atalho para o uso de `seq()` e `rep()` vistos anteriormente e que já retorna um fator. Tipicamente é usada para criar fatores indicadores da estrutura de experimentos.

O seguinte comando produz um resumo dos dados de cada grupo.

```
> with(d2, by(Y, lev, summary))
```

```
lev: 1
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 8.348   8.700   9.748   9.913  11.330  11.437
```

```
-----
lev: 2
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
15.38	16.65	16.78	16.87	16.84	18.72

lev: 3

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
10.79	12.34	14.90	13.94	15.38	16.30

Há funções que facilitam a criação de variáveis. Na criação de *data-frame*, `expand.grid()` pode ser muito útil gerando automaticamente combinações de valores.

```
> 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

Outro exemplo:

```
> 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

5.5 Listas

Listas são estruturas genéricas e flexíveis que permitem armazenar diversos formatos em um único objeto. A lista a seguir contém um vetor, um caractere e uma matriz.

```
> 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

Muitas funções retornam uma lista. A função `lm()` ajusta um modelo linear. No exemplo a seguir é feito um ajuste de uma regressão linear simples.

```
> 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"
[5] "fitted.values" "assign"       "qr"           "df.residual"
[9] "xlevels"      "call"        "terms"       "model"
> lis2$fitted
      1      2      3      4      5      6      7      8
53.50909 56.95152 60.39394 63.83636 67.27879 70.72121 74.16364 77.60606
      9     10
81.04848 84.49091
> lis2$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

```

No próximo exemplo a função `aov()` faz o ajuste de um modelo de análise de variância e também retorna uma lista.

```

> lis3 <- aov(Y ~ lev, data=d2)
> lis3
Call:
aov(formula = Y ~ lev, data = d2)

Terms:
                lev Residuals
Sum of Squares 122.09738  35.02747
Deg. of Freedom      2      12

Residual standard error: 1.708495
Estimated effects may be unbalanced
> summary(lis3)
          Df Sum Sq Mean Sq F value    Pr(>F)
lev         2 122.10   61.05   20.91 0.000123 ***
Residuals   12  35.03    2.92
---
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. 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

```

5.6 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.

5.7 Que tipo de objeto eu tenho?

As funções do tipo `is.*()` mencionadas no início desta seção podem ser usadas para obter informações sobre a natureza de um objeto, o que pode ser 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 objetos que criamos anteriormente: `x1` é um vetor numérico, `estados` é um fator com 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 matriz de seis elementos e de dimensão 3×2 .

```
> str(x1)
num 10

> str(estados)
Factor w/ 3 levels "PR","RS","SC": 1 3 2

> str(ar1)
int [1:3, 1:4, 1:2] 1 2 3 4 5 6 7 8 9 10 ...
- attr(*, "dimnames")=List of 3
..$ : chr [1:3] "Baixo" "Médio" "Alto"
..$ : chr [1:4] "col1" "col2" "col3" "col4"
..$ : chr [1:2] "Masculino" "Feminino"

> str(d1)
'data.frame':      10 obs. of  2 variables:
 $ X: int   1 2 3 4 5 6 7 8 9 10
 $ Y: num  51 54 61 67 68 75 77 75 80 82

> str(lis1)
List of 2
 $ nomes: chr [1:3] "Pedro" "Joao" "Maria"
 $ mat  : int [1:3, 1:2] 1 2 3 4 5 6
```

5.8 Exercícios

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

- (a) o vetor

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

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

- (c) `[1] 10 11 12 13 14 15 16 17 18 19 20`

- (d) o vetor com a sequência de valores

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

- (e) o vetor com a sequência 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 sequência repetida

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

- (i) o vetor de sequência 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 sequência 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
[26] 51 53 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
[11] Trat_4 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
[10] Bloco_1 Bloco_2 Bloco_3
Levels: Bloco_1 Bloco_2 Bloco_3
```

- Mostre comando(s) para construir uma matriz 10×10 tal que as entradas são iguais a $i * j$, sendo i a linha e j a coluna.
- Construa um data-frame com uma tabela com três colunas: x , x^2 e $\exp(x)$, com x variando de 0 a 50.
- 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 sequência de 10 vetores x . Use estas e outras funções para calcular a soma dos 100 primeiros termos das séries:

- (a) $1 + 1/2 + 1/3 + 1/4 + \dots$
- (b) $1 + 1/2^2 + 1/4^2 + 1/6^2 + 1/8^2 + \dots$
- (c) $1/(1 + 1/1!)^2 + 1/(1 + 1/2!)^2 + 1/(1 + 1/3!)^2 + \dots$

- (d) $1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + \dots$
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 frequências a seguir. Entre com os dados usando o tipo de objeto adequado e mostre os comandos para responder as perguntas abaixo.

Idade	Fumante		Não Fumante	
	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?

6 Miscelânea de funcionalidades do R

6.1 O R como calculadora

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

(a) $10^2 + 11^2 + \dots + 20^2$

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

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

6.2 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 1 e 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 ideia 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", xlab = "x", ylab = "f(x)")
```

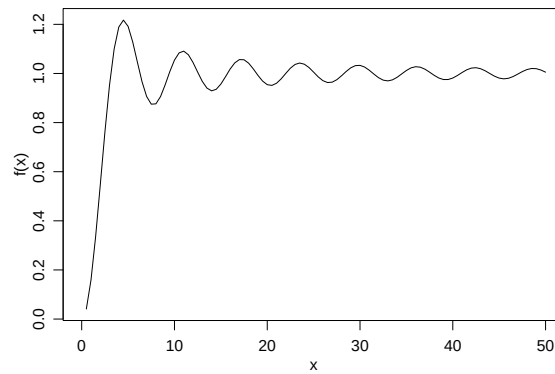


Figura 1: Gráfico da função dada em (a).

Note que este procedimento é o mesmo que aprendemos para fazer esboços de gráficos a mão em uma folha de papel, calculando e unindo ponto a ponto uma sequência de pontos!

Há ainda uma outra maneira de fazer isto no R utilizando `plot.function()` ou `curve()` conforme pode ser visto nos comandos a seguir que nada mais fazem do que combinar os três comandos anteriores em apenas um comando.

```
> plot(function(x) 1 - (1/x) * sin(x), from = 0, to = 50)
> curve(1 - (1/x) * sin(x), from = 0, to = 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", xlab = "x", ylab = "f(x)")
```

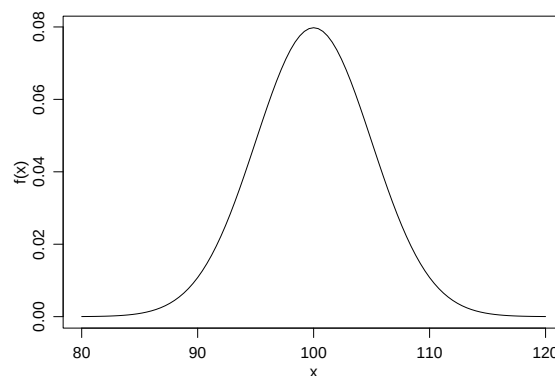


Figura 2: Gráfico da função dada em (b).

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)
```

6.3 Integração numérica

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

$$I = \int_{-3}^3 x^2 dx. \quad (2)$$

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 3 obtida com os comandos a seguir. Inicialmente é desenhada a função como nos exemplos anteriores. A área que corresponde à integração é desenhada por `polygon()` que utiliza uma sequência de coordenadas (x, y) que define o polígono. A função `rev()` reverte a ordem dos elementos de uma sequência.

```
> 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')
```

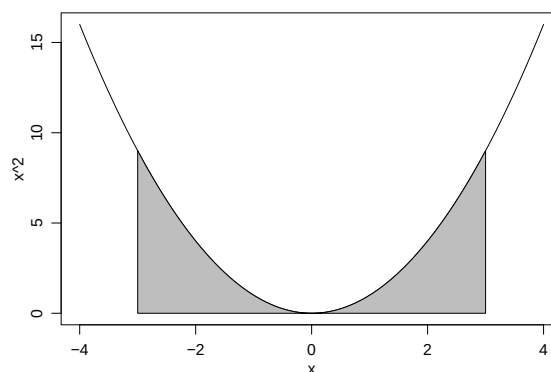


Figura 3: Gráfico onde a área indicada corresponde à integral definida na equação 2.

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 uma 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 mesmos resultados conforme 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
```

6.4 Matemática simbólica no R

Embora o R seja um programa predominantemente para operações numéricas, é possível obter alguns resultados simbólicos, em particular para expressões de derivadas que podem ser informadas para algoritmos de otimização numérica. A forma básica de utilização consiste em:

- (i) defina a expressão desejada dentro de `quote()`,
- (ii) use `D()` para obter a expressão da derivada desejada informando a expressão e o termo em relação ao qual deseja-se derivar a expressão,
- (iii) use `eval()` caso queira obter o valor numérico de uma determinada expressão. A documentação `help(D)` fornece mais detalhes.

Vejamos um exemplo.

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

Existem programas computacionais especializados em matemática simbólica dentre os quais destacam-se os projetos *axiom*, *maxima*, *yacas* e *SymPy* e ainda o site *WolframAlfa*.

- o programa *axiom* está disponível em <https://github.com/daly/axiom>
- o programa *maxima* está disponível em <https://maxima.sourceforge.io>
- o programa *yacas* está disponível em <https://yacas.sourceforge.net>
- o *SymPy* (biblioteca de matemática simbólica em Python) está disponível em <https://github.com/sympy/sympy>

Todos estes programas estão disponíveis para várias plataformas/sistemas operacionais. Também estão incluídos nas distribuições de sistemas LINUX podendo ser instalados a partir de versões pré-compiladas. Alguns ainda oferecem interfaces gráficas.

Para o YACAS há o pacote do R **Ryacas** (cran.r-project.org/package=Ryacas) que permite acessar suas funcionalidades diretamente de dentro de uma sessão. O pacote **rSymPy**, que fazia o mesmo para o SymPy, foi removido do CRAN e não é mais mantido.

6.5 Exercícios

1. Calcule o valor das expressões abaixo

- (a) Seja $x = (12, 11, 14, 15, 10, 11, 14, 11)$.

Calcule $E = -n\lambda + (\sum_1^n x_i) \log(\lambda) - \sum_1^n \log(x_i!)$, onde n é o número de elementos do vetor x e $\lambda = 10$.

Dica: o fatorial de um número pode ser obtido utilizando a função `prod`. Por exemplo o valor de $5!$ é obtido com o comando `prod(1:5)`.

Há ainda uma outra forma usando a função Gama e lembrando que para a inteiro, $\Gamma(a+1) = a!$. Portanto podemos obter o valor de $5!$ com o comando `gamma(6)`.

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

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

2. Obtenha o gráfico das seguintes funções:

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

(b) Para $\phi = 4$,

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

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

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

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

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

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

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

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

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

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

(a) $1 + 1/3^2 + 1/5^2 + 1/7^2 + 1/9^2 + \dots$

(b) $1 - 1/2^2 + 1/3^2 - 1/4^2 + 1/5^2 - 1/6^2 + \dots$

7 Dados no R

Pode-se entrar com dados no R de diferentes formas. O formato mais adequado vai depender do formato e tamanho do conjunto de dados. Uma importante diferença é se os dados já existem em formato do R ou 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. Iniciamos com a entrada de dados diretamente no R como se não tivéssemos ainda dados em algum formato eletrônico. A seguir descreve-se como importar dados já disponíveis eletronicamente de um arquivo texto, em outro sistema ou no próprio R. Ao final discutimos diferentes formas de exportar/importar objetos do R.

7.1 Entrando com dados diretamente no R

7.1.1 Definindo vetores

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

```
> a1 <- c(2,5,8)      # cria vetor a1 com os dados 2, 5 e 8
> a1                  # exibe os elementos de a1
[1] 2 5 8
> (a1 <- c(2,5,8))    # cria e exibe vetor a1 com os dados 2, 5 e 8
[1] 2 5 8
> (a2 <- c(23,56,34,23,12,56))
[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 têm algum "padrão" tal como elementos repetidos ou números sequenciais pode-se usar mecanismos do R para facilitar a entrada dos dados na forma de vetores. Examine os seguintes exemplos.

```
> (a3 <- 1:10)          # vetor com números sequenciais de 1 a 10
[1] 1 2 3 4 5 6 7 8 9 10
> (a4 <- (1:10)*10)      # vetor com elementos 10, 20, ..., 100
[1] 10 20 30 40 50 60 70 80 90 100
> (a5 <- rep(3, 5))      # vetor com elemento 3 repetido 5 vezes
[1] 3 3 3 3 3
> (a6 <- rep(c(5,8), 3))  # vetor repetindo 3 vezes 5 e 8 alternadamente
[1] 5 8 5 8 5 8
> (a7 <- rep(c(5,8), each=3)) # vetor repetindo 3 vezes 5 e depois 8
[1] 5 5 5 8 8 8
```

7.1.2 Entrada via teclado

Usando a função `scan()` Esta função lê dados diretamente 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 em mais detalhes mais adiante.

Usando `textConnection()` Esta função generaliza a anterior permitindo que se entre com mais de um campo por linha, gravando o resultado em um `data.frame()`. O seguinte exemplo foi postado por Henrique Dallazuanna na lista R-STAT e ilustra este uso.

```
d <- read.table(textConnection("trat resposta
a 10
a 12
b 15
b 20
c 12
c 5
d 8
d 10"), header = TRUE, stringsAsFactors = TRUE)
str(d)
'data.frame':      8 obs. of  2 variables:
 $ trat      : Factor w/ 4 levels "a","b","c","d": 1 1 2 2 3 3 4 4
 $ resposta: int  10 12 15 20 12 5 8 10
```

Uso da função `readLines()` Este função é particularmente útil para ler entradas na forma de texto (*strings*). Por exemplo, para ler uma linha a ser digitada na tela do R siga o comando abaixo e digite o texto indicado. Ao terminar pressione a tecla ENTER e o texto será armazenado no objeto `texto`.

```
> texto <- readLines(n=1)
> ## Este é um texto de exemplo
> texto
```

Um possível uso é dentro de funções que solicitem que o usuário responda e/ou entre com informações à medida que são solicitadas. Experimente definir e rodar a função a seguir.

```
> rL.ex <- function(){
+   cat("digite o nome do time de futebol de sua preferencia (em letras minúsculas)\n")
+   time <- readLines(n=1)
+   if(time == "cruzeiro") cat("BOA ESCOLHA!!!\n")
+   else cat("Ihh, tá mal de escolha...\n")
+   return(invisible())
+ }
> rL.ex()
```

Nesse exemplo `readLines()` foi utilizada para efetuar via teclado mas a função permite ainda entrada de dados por conexões com outros dispositivos de *input*. Por exemplo, pode ser utilizada para ler texto de um arquivo. Consulte a documentação da função para maiores detalhes e exemplos.

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 todos dados iguais ou acima de 30 têm seu valor alterado para 30.

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

7.1.3 Usando a função `edit()`

O comando `edit(data.frame())` abre uma planilha para digitação de dados que são armazenados como *data.frames*. Data-frames são o análogo no R a 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 você precisar abrir novamente planilha com os dados, para fazer correções e/ou inserir mais dados use o comando `fix()`. No exemplo acima você digitaria `fix(a8)`.

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

7.2 Lendo dados de um arquivo texto

Se os dados já estão disponíveis em formato eletrônico, isto é, já foram digitados em outro programa, você 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, você 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()` (e variantes) 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 e visualizar os dados deste arquivo usamos:

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

Exemplo 2: Em um segundo 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 `header=TRUE`. Portanto para importar este arquivo usamos:

```
(ex02 <- read.table("exemplo02.txt", header=TRUE))
```

Exemplo 3: Para o terceiro 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 pelo caractere `:` separador de valores. Além 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=TRUE, sep=":", dec=","))
```

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 em determinadas situações.

- `read.fwf()` é conveniente para ler *fixed width formats*, arquivos em que cada campo é delimitado por um certo número de posições.
- `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.csv2()`, `read.delim()` e `read.delim2()`. Estas funções são praticamente iguais a `read.table()` porém com diferentes opções *default*. Em geral (mas não sempre) dados em formato `csv` usado no Brasil são lidos diretamente com `read.csv2()`.

Exemplo 4: Um caso particular é quando os dados estão em formato de *largura fixa* (*fixed width format*), sem nenhum caractere separando os campos – cada variável ocupa sempre a mesma quantidade de colunas no arquivo. Nestes casos `read.table()` não funciona e deve-se usar `read.fwf()`, informando a largura de cada campo através do argumento `widths`. Considere por exemplo o arquivo [fwf01.txt](#) com o seguinte conteúdo:

```
001M23
002F31
003M45
004F29
```

onde as três primeiras colunas correspondem a um código, a quarta ao sexo e as duas últimas à idade, sem nenhum espaço ou símbolo separando os campos. Para importar estes dados usamos:

```
> read.fwf("fwf01.txt", widths = c(3, 1, 2),
+         col.names = c("codigo", "sexo", "idade"))
```

Exemplo 5: As funções permitem ler ainda dados diretamente disponíveis na *web*. Por exemplo os dados dos exemplos 1, 2 e 3 poderiam ser lidos diretamente com o comando a seguir, sem a necessidade de copiar primeiro os dados para algum local no computador do usuário:

```
> ex1 <- read.table("http://www.leg.ufpr.br/Rembrapa/dados/gam01.txt")
> ex02 <- read.table("http://www.leg.ufpr.br/Rembrapa/dados/exemplo02.txt", header=TRUE)
> ex03 <- read.table("http://www.leg.ufpr.br/Rembrapa/dados/dadosfic.csv",
+                   head=TRUE, sep=":", dec=",")
> ex03 <- read.csv2("http://www.leg.ufpr.br/Rembrapa/dados/dadosfic.csv", sep = ":")
```

7.3 Lendo dados através da área de transferência

Um mecanismo comum para copiar dados de um programa para o outro é usando a *área de transferência*. Tipicamente isto é feito com o mecanismo de *recorta-e-cola*, ou seja, marca-se os dados desejados em algum aplicativo (editor, planilha, página web, etc), usa-se o mecanismo de COPIAR (opção no menu do programa que muitas vezes corresponde ao teclar CTRL-C), o que transfere os dados para a *área de transferência*. Funções como `scan()`, `read.table()` e outras podem ser usadas para ler os dados diretamente da área de transferência passando-se a opção "clipboard" ao primeiro argumento. Por exemplo, os seguintes dados:

ID	Grupo	Gasto	Ano
23	A	25,4	11
12	B	12,3	09
23	A	19,8	07

podem ser marcados e copiados para área de transferência e lidos diretamente com:

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

7.4 Carregando dados já disponíveis no R

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

```
> data(mtcars)
> head(mtcars)
```

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

```
> require(MASS)
Carregando pacotes exigidos: MASS
> data(topo)
> find("topo")
[1] ".GlobalEnv" "package:MASS"
> 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 R e portanto deve ser carregado com `require()` para depois podermos acessar os dados.

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

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

7.5 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)
```

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 e `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*.

Um mecanismo genérico O pacote `rio` contém uma função `import()` que é capaz de importar dados de uma diversidade de formatos incluindo todas as opções mencionadas anteriormente e muitas outras. A **vinheta de apresentação do pacote** descreve este poderoso e versátil recurso.

7.6 Acesso a planilhas e bancos de dados relacionais

É comum que dados estejam armazenados em planilhas eletrônicas tais como *MS-Excel* ou *LibreOffice 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 formatos. 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 este pacote está implementado apenas para o sistema operacional Windows.
- O pacote **readxl** lê arquivos *.xls* e *.xlsx* diretamente, sem depender de sistema operacional, do *Perl* ou de drivers ODBC. Suponha que você possua um arquivo de uma planilha MS-Excel já no seu diretório (pasta) de trabalho do R chamado `planilha.xls`, 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(readxl)
> dados1 <- read_excel("planilha.xls", sheet="Planilha1")
> 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 `xls2csv` combinada com `read.csv()` ou `read.csv2()`, sendo esta última recomendada para dados com *vírgula* como caractere separador de decimais. O *Perl* é tipicamente instalado em sistemas Linux/Unix e também livremente disponível para outros sistemas operacionais. Note que `xls2csv` é um comando do sistema operacional e não do R.

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

- O pacote **gdata** possuía uma função `read.xls()` que encapsulava opções mencionadas anteriormente, mas a função não está mais disponível no pacote.

Estruturas de dados mais complexas são tipicamente armazenadas em DBMS's (database management system) ou RDBMS's (relational database management system). Alguns exemplos são Oracle, Microsoft SQL server, MySQL, PostgreSQL, Microsoft Access, dentre outros. O R possui ferramentas implementadas em pacotes para acesso a estes sistemas gerenciadores e esse é um tópico especializado que não abordaremos aqui. Para mais detalhes consulte o manual *R Data Import/Export* e a documentação dos pacotes que implementam tal funcionalidade. Alguns deles disponíveis por ocasião da redação deste texto eram: **odbc**, **DBI**, **RMySQL**, **RPostgreSQL**, **ROracle**, **RNetCDF**, **RSQLite**, dentre outros. Certamente as opções atuais são diferentes.

7.7 Exportando dados para arquivos

Assim como podemos importar dados de arquivos externos, muitas vezes é necessário o processo inverso: gravar em um arquivo os dados que estão em um objeto do R, por exemplo um *data.frame* resultante de alguma manipulação ou análise.

As funções mais usadas para isto são `write.table()`, `write.csv()` e `write.csv2()`, análogas às funções `read.table()`, `read.csv()` e `read.csv2()` vistas anteriormente e que seguem as mesmas convenções. Por exemplo, `write.csv2()` grava o arquivo usando ponto-e-vírgula como separador de campos e vírgula como separador decimal, mais comum no formato `csv` usado no Brasil.

```
> df <- data.frame(trat = c("A","A","B","B"), resposta = c(10, 12, 15, 20))
> write.table(df, "dadosExp.txt")
> write.csv(df, "dadosExp.csv", row.names = FALSE)
> write.csv2(df, "dadosExp2.csv", row.names = FALSE)
```

Os arquivos com os noems entre aspas nos comandos acima são gravados no diretório local, a menos que algum outro caminho seja especificado no nome dos arquivos. Note que por *default* estas funções gravam também os nomes das linhas (*row names*) do objeto, o que na maioria das vezes não é desejado. Para evitar isto usa-se o argumento `row.names = FALSE` como no exemplo acima.

Para exportar diretamente para uma planilha do MS-Excel pode-se usar, por exemplo, a função `write_xlsx()` do pacote **writexl**, que é o equivalente para gravação do pacote **readxl** mencionado anteriormente.

```
> require(writexl)
> write_xlsx(df, "planilhaExp.xlsx")
```

7.8 Salvando e restaurando objetos e o *workspace* do R

Além de importar e exportar dados em formato texto ou de outros programas, o R possui mecanismos próprios para salvar e restaurar objetos. Isto é conveniente por preservar a classe e os atributos do objeto original. Dessa forma fatores continuam fatores e listas continuam listas. Estes mecanismos se distinguem em dois aspectos independentes: o **número de objetos** tratados por vez (um único, vários, ou todo o *workspace*) e o **formato do arquivo** gerado.

Quanto ao formato, `saveRDS()`, `save()` e `save.image()` gravam em formato **binário**. É um formato compacto e rápido de gravar/ler, porém ilegível diretamente por humanos e pouco amigável a sistemas de controle de versão como o `git` já que não consegue mostrar diferenças significativas entre duas versões do arquivo.

Alternativamente `dput()` e `dump()`, suas contrapartes vistas a seguir junto com `saveRDS()` e `save()`, gravam em formato **texto** (ASCII). Nesse formato o arquivo contém o próprio código R (obtido por *deparse*) que, ao ser executado, recria o objeto. Por isso pode ser lido, editado e comparado como qualquer outro *script*, embora seja mais lento e ocupe mais espaço para objetos grandes, mas pode ser comprimido usando ferramentas como `zip`, `tar` ou similar.

Exemplo 1 Vamos considerar que desejamos exportar para uma representação do R um único objeto usando `saveRDS()/readRDS()` (binário) ou `dput()/dget()` (texto)

Estas funções gravam e restauram um único objeto do R por vez, tipicamente em um arquivo com extensão `.rds` no caso de `saveRDS()`. Uma característica importante é que o nome do objeto **não** é gravado junto com os dados. Ao restaurar, o resultado deve ser atribuído a um objeto, que pode inclusive ter um nome diferente do original.

```
> saveRDS(df, "df.rds")
> rm(df)
> df.novo <- readRDS("df.rds")
> df.novo
```

	trat	resposta
1	A	10
2	A	12
3	B	15
4	B	20

O equivalente em formato texto é dado por `dput()` e `dget()`, que funcionam da mesma forma:

```
> dput(df.novo, "df.R")
> d.novo2 <- dget("df.R")
> d.novo2
```

	trat	resposta
1	A	10
2	A	12
3	B	15
4	B	20

Note no arquivo `d.R` gerado que o seu conteúdo é o próprio comando R que constrói o objeto `d.novo` (uma chamada a `structure()`), e não os dados em formato de tabela.

Exemplo 2 Vamos agora considerar que desejamos exportar uma representação do R de vários objetos: `save()/load()` (binário) e `dump()/source()` (texto).

As funções `save()` e `load()` permitem gravar vários objetos de uma só vez, tipicamente em um arquivo com extensão `.Rdata` ou `.rda`. Diferentemente de `saveRDS()`, aqui os nomes dos objetos **são** gravados junto com os dados e restaurados automaticamente com estes mesmos nomes ao usar `load()`. Não há como atribuir o resultado a um nome diferente e objetos já existentes com o mesmo nome são sobrescritos.

```
> a1 <- 1:5
> a2 <- c("x", "y", "z")
> save(a1, a2, file = "objetos.Rdata")
> rm(a1, a2)
> load("objetos.Rdata")
> a1
[1] 1 2 3 4 5
> a2
[1] "x" "y" "z"
```

O equivalente em formato texto é dado por `dump()` e `source()`: a primeira grava, em um único arquivo texto, o código R que recria cada um dos objetos indicados; a segunda simplesmente executa este código, recriando-os com os mesmos nomes originais – exatamente como fazemos ao rodar (`source`) qualquer outro *script* do R.

```
> dump(c("a1", "a2"), file = "meus-objetos.R")
> rm(a1, a2)
> source("meus-objetos.R")
> a1
[1] 1 2 3 4 5
> a2
[1] "x" "y" "z"
```

Exemplo 3 Por fim, é possível salvar **todos** os objetos criados na sessão de uma só vez com `save.image()`, que por *default* grava um arquivo chamado `.RData` no diretório de trabalho. Este é o mesmo arquivo que o R oferece para gravar ao ser fechado (a pergunta *"Save workspace image?"*) e que, se existir, é automaticamente recarregado na próxima vez que o R for iniciado no mesmo diretório.

```
> save.image() # grava .RData no diretório de trabalho
```

Nota: embora conveniente, esse carregamento automático do `.RData` prejudica a reprodutibilidade do trabalho, pois o resultado de uma sessão passa a depender de objetos “invisíveis” criados em sessões anteriores. Por isso muitos usuários preferem desabilitar este mecanismo, recriando sempre os objetos necessários a partir dos *scripts*.

8 Análise descritiva

8.1 Descrição univariada

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

Uma boa forma de iniciar uma análise descritiva adequada é verificar os tipos 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.

8.2 O conjunto de dados “milsa”

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 2. Consulte o livro para mais detalhes sobre estes 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 quando 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 3. Lembre-se que se você precisar corrigir algo na digitação você pode fazê-lo abrindo a planilha novamente com o comando `fix(milsa)`.

```
> milsa <- edit(data.frame()) # abra a planilha para entrada dos dados
> milsa                      # visualiza os dados digitados
> fix(milsa)                 # comando a ser usado para correções, se necessário
```

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 categóricas são definidas usando o comando `factor()`, que vamos usar para redefinir

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

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 `instrucao` 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. Inspeção os comandos mostrados a seguir e os resultados por elas 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) # conferindo se é um data-frame
[1] TRUE
> names(milsa)          # vendo o nome das variáveis
[1] "funcionario" "civil"          "instrucao"   "filhos"      "salario"
[6] "ano"          "mes"          "regiao"      "idade"
> dim(milsa)            # vendo as dimensões do data-frame
[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)          # anexando ao caminho de procura
```

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

8.2.1 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 estão 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 diferentes 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 4.

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

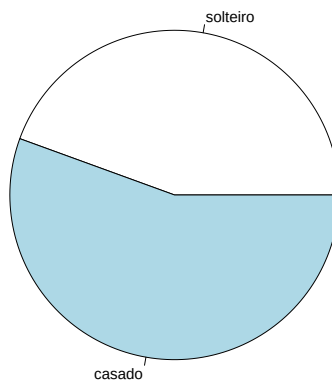


Figura 4: Gráfico de setores para variável `civil`.

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

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

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

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

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

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

```
> 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 conforme mostrado na Figura 5.

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

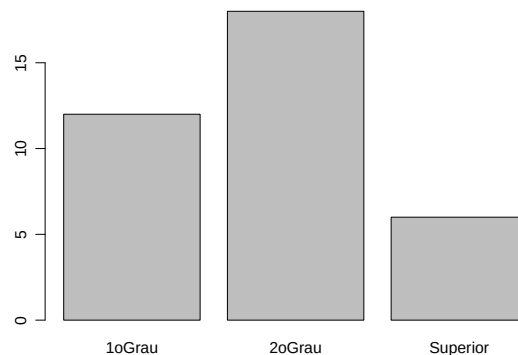


Figura 5: Gráfico de barras para variável `instrucao`.

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)[which.max(instrucao.tb)]
> instrucao.mo
[1] "2oGrau"
> median(as.numeric(instrucao)) # só calcula mediana de variáveis numéricas
[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 6 o obtido com os comandos a seguir.

```
> plot(filhos.tb)           # gráfico das frequências absolutas
```

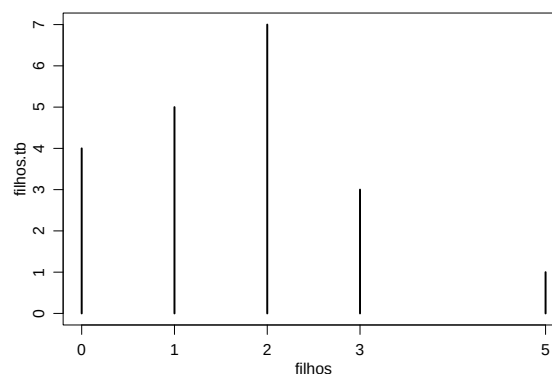


Figura 6: Gráfico de frequências absolutas para variável `filhos`.

Outra possibilidade seria fazer gráficos de frequências relativas e de frequências acumuladas conforme mostrado na Figura 7.

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

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

A seguir mostramos como obter algumas medidas de posição: moda, mediana, média e média aparada. Note que o argumento `na.rm=T` é necessário porque não há informação sobre número de filhos para alguns indivíduos. O argumento `trim=0.1` indica uma média aparada onde foram retirados 10% dos menores e 10% dos maiores dados. Ao final mostramos como obter os quartis, mínimo e máximo.

```
> filhos.mo <- names(filhos.tb)[which.max(filhos.tb)]
> filhos.mo           # moda
```

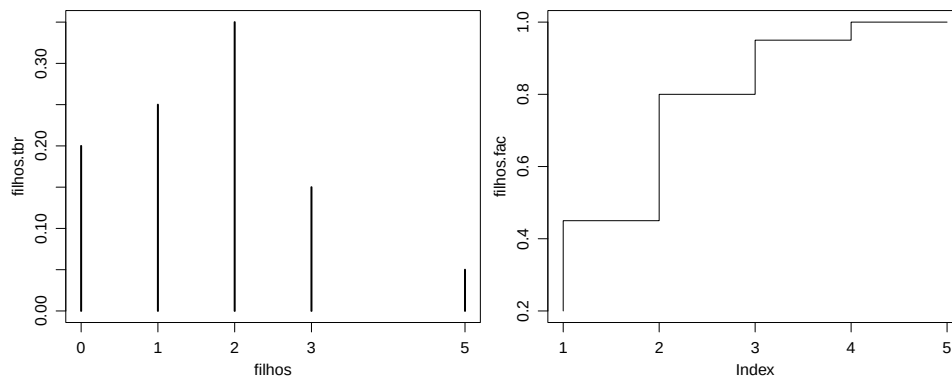


Figura 7: Gráfico de frequências relativas (esquerda) e frequências acumuladas para variável **filhos**.

```
[1] "2"
> filhos.md <- median(filhos, na.rm=T)
> filhos.md           # mediana
[1] 2
> filhos.me <- mean(filhos, na.rm=T)
> filhos.me           # média
[1] 1.65
> filhos.me <- mean(filhos, trim=0.1, na.rm=T)
> filhos.me           # média
[1] 1.5625
> filhos.qt <- quantile(filhos, na.rm=T)
```

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

```
> range(filhos, na.rm=T)
[1] 0 5
> filhos.A <- diff(range(filhos, na.rm=T))
> filhos.A
[1] 5
> var(filhos, na.rm=T)           # variância
[1] 1.607895
> filhos.dp <- sd(filhos, na.rm=T) # desvio padrão
> filhos.dp
[1] 1.268028
> filhos.cv <- 100 * filhos.dp/filhos.me # coeficiente de variação
> filhos.cv
[1] 81.15379
> filhos.qt <- quantile(filhos, na.rm=T)
> filhos.ai <- filhos.qt[4] - filhos.qt[2] # amplitude interquartílica
> filhos.ai
75%
1
```

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

```
> summary(filhos)           # várias medidas
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.     NAs
  0.00   1.00   2.00   1.65   2.00   5.00     16

> 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 absolutas e relativas.

```
> range(salario)           # máximo e mínimo
[1] 4.0 23.3

> nclass.Sturges(salario) # número de classes pelo critério de Sturges
[1] 7

> args(cut)
function (x, ...)
NULL

> args(cut.default)
function (x, breaks, labels = NULL, include.lowest = FALSE, right = TRUE,
  dig.lab = 3L, ordered_result = FALSE, ...)
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 8. Neste comando fazemos `main=` para evitar que a função coloque automaticamente o título no gráfico, o que é o comportamento padrão de `hist()`. É sempre bom lembrar que há várias outras opções fornecidas pelos argumentos das funções. Por exemplo, em `hist()` acrescentando `labels=TRUE` as frequências são mostradas em cima de cada barra, `prob=TRUE` faz com que o gráfico exiba as frequências relativas. Por *default* a função calcula automaticamente o número de classes e os valores limites de cada classe e isto pode ser alterado com o argumento `breaks` que pode receber um vetor definindo os limites das classes definidos

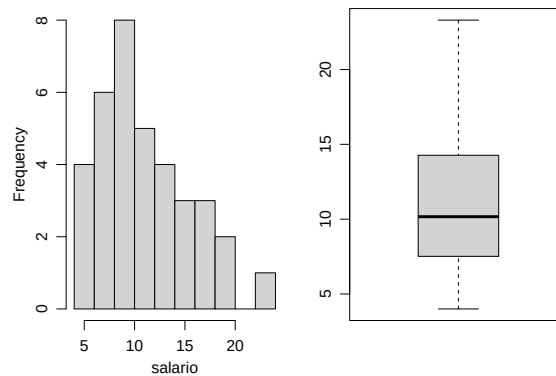


Figura 8: Histograma (esquerda) e boxplot (direita) para a variável `salario`.

pelo usuário, um nome de critério (*default*="Sturges"), número de classes ou mesmo uma função para definir as classes; ou ainda `nclass` pode receber um escalar definindo o número de classes. Além destas há ainda várias outras opções implementadas pelos argumentos da função conforme descrito em `help(hist)`. Da mesma forma argumentos permitem variações em `boxplot` tais como caixas com tamanho proporcional aos tamanhos dos grupos, caixas "acinturadas" (*notched boxplot*) entre outras.

```
> hist(salario, main="")
> 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 | 15701488
10 | 5816
12 | 08269
14 | 77
16 | 0263
18 | 84
20 |
22 | 3
```

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

```
> salario.md <- median(salario, na.rm=T)
> salario.md
# mediana
[1] 10.165
> salario.me <- mean(salario, na.rm=T)
> salario.me
# média
[1] 11.12222
> range(salario, na.rm=T)
[1] 4.0 23.3
> salario.A <- diff(range(salario, na.rm=T)) # amplitude
> salario.A
```

```

[1] 19.3
> var(salario, na.rm=T)      # variância
[1] 21.04477
> salario.dp <- sd(salario, na.rm=T)  # desvio padrão
> salario.dp
[1] 4.587458
> salario.cv <- 100 * salario.dp/salario.me  # coeficiente de variação
> salario.cv
[1] 41.24587
> salario.qt <- quantile(salario, na.rm=T)
> salario.ai <- salario.qt[4] - salario.qt[2] # amplitude interquartílica
> salario.ai
75%
6.5075
> summary(salario)          # várias medidas
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
4.000   7.553  10.165  11.122  14.060  23.300
> fivenum(salario)
[1]  4.000  7.515 10.165 14.270 23.300

```

8.2.2 Análise Bivariada

Na análise bivariada procuramos identificar relações 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 análises 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 consideradas 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* ou *tabela de contingência* e pode ser apresentada de várias formas, conforme discutido a seguir. A forma mais adequada de apresentação vai depender dos objetivos da análise e da interpretação desejada para os dados. Inicialmente obtemos com `table()` a tabela de frequências absolutas. A tabela estendida incluindo os totais marginais pode ser obtida com `addmargins()`.

```

> civ.gi.tb <- table(civil, instrucao) # frequências absolutas
> civ.gi.tb

```

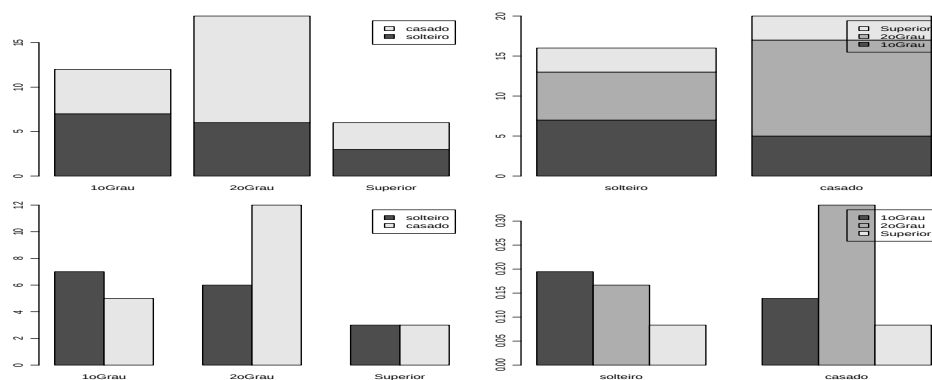


Figura 9: Quatro tipos de gráficos de barras ilustrando o resultado do cruzamento das variáveis civil e instrucao.

```

      instrucao
civil   1oGrau 2oGrau Superior
solteiro    7     6      3
casado      5    12      3
> addmargins(civ.gi.tb)

      instrucao
civil   1oGrau 2oGrau Superior Sum
solteiro    7     6      3    16
casado      5    12      3    20
Sum        12    18      6    36

```

Tabelas de frequências relativas são obtidas com `prop.table()` e tais frequências podem ser globais, por linha (`margin=1`) ou por coluna (`margin=2`).

```

> prop.table(civ.gi.tb)

      instrucao
civil   1oGrau  2oGrau  Superior
solteiro 0.1944444 0.1666667 0.08333333
casado   0.1388889 0.3333333 0.08333333
> prop.table(civ.gi.tb, margin=1)

      instrucao
civil   1oGrau 2oGrau Superior
solteiro 0.4375 0.3750  0.1875
casado   0.2500 0.6000  0.1500
> prop.table(civ.gi.tb, margin=2)

      instrucao
civil   1oGrau  2oGrau  Superior
solteiro 0.5833333 0.3333333 0.5000000
casado   0.4166667 0.6666667 0.5000000

```

Na Figura 9 mostramos quatro diferentes gráficos de barras que podem ser usados para representar o cruzamento das variáveis. A transposição da tabela com `t()` permite alterar a variável que define os grupos no eixo horizontal. O uso de `prop.table()` permite a obtenção de gráficos com frequências relativas.

```

> barplot(civ.gi.tb, legend=T)
> barplot(t(civ.gi.tb), legend=T)
> barplot(civ.gi.tb, beside=T, legend=T)
> barplot(t(prop.table(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 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 linhas e colunas da tabela. Os comandos a seguir mostram como obter todas estas medidas.

```
> summary(civ.gi.tb) # resumo incluindo o teste Chi-quadrado
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)) # resumo incluindo o teste Chi-quadrado
[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
```

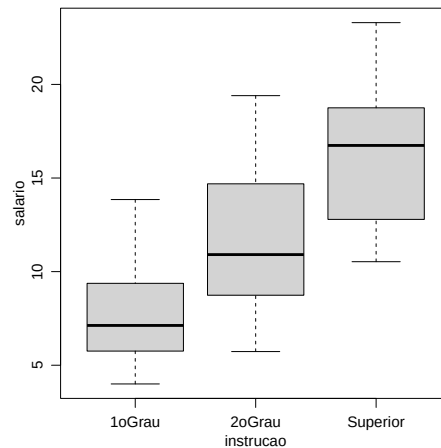


Figura 10: Boxplot da variável `salario` para cada nível da variável `instrucao`.

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()`. Note que as classes são definidas por intervalos abertos à esquerda e então usamos o argumento `include.lowest=TRUE` para garantir que todos os dados, inclusive o menor (mínimo) seja incluído na primeira classe. 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), include.lowest=T)
> 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	7	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    0.5833333 0.2500000 0.1666667 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 informando este resumo para cada nível do fator qualitativo. A seguir mostramos alguns exemplos de como obter a média, desvio padrão e o resumo de cinco números do salário para cada nível de instrução.

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

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

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

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

```
> idade.cl <- cut(idade, quantile(idade), include.lowest=T)
> table(idade.cl)
idade.cl
[20.8,30.7] (30.7,34.9] (34.9,40.5] (40.5,48.9]
      9          9          9          9

> salario.cl <- cut(salario, quantile(salario), include.lowest=T)
> table(salario.cl)
salario.cl
 [4,7.55] (7.55,10.2] (10.2,14.1] (14.1,23.3]
      9          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]      4          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] 0.4444444 0.2222222 0.2222222 0.1111111
(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 menor de classes podemos fazer como no exemplo a seguir onde cada variável é dividida em 3 classes e gerando um tabela de cruzamento 3×3 .

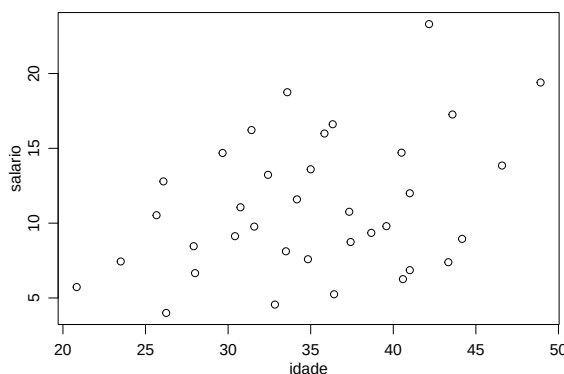


Figura 11: Diagrama de dispersão para as variáveis `salario` e `idade`.

```
> idade.cl1 <- cut(idade, quantile(idade, seq(0,1,len=4)), include.lowest=T)
> salario.cl1 <- cut(salario, quantile(salario, seq(0,1,len=4)), include.lowest=T)
> table(idade.cl1, salario.cl1)
```

	salario.cl1		
idade.cl1	[4,8.65]	(8.65,12.9]	(12.9,23.3]
[20.8,32.1]	5	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]	0.4166667	0.4166667	0.1666667
(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 no 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
```

8.2.3 Análises descritivas multivariadas

Em geral as técnicas descritivas uni e bivariadas mostradas até aqui podem ser estendidas a mais dimensões. Além disto há ferramentas específicas para exploração de dados multidimensionais. Neste momento vamos apenas ilustrar como o comando `tapply()` pode ser estendido.

Anteriormente nesta sessão mostramos como calcular o salário médio para cada nível de instrução. Vamos estender este exemplo calculando o salário médio também para cada estado civil e para combinação dos níveis destes dois fatores. Esta última é mostrada de duas maneiras, uma delas retornando um vetor e outra uma matriz.

```
> tapply(salario, instrucao, mean)
  1oGrau  2oGrau Superior
7.836667 11.528333 16.475000
> tapply(salario, civil, mean)
solteiro  casado
9.870625 12.123500
> tapply(salario, interaction(instrucao, civil), mean)
  1oGrau.solteiro  2oGrau.solteiro Superior.solteiro  1oGrau.casado
      8.402857      8.935000      15.166667      7.044000
  2oGrau.casado  Superior.casado
    12.825000     17.783333
> tapply(salario, list(instrucao, civil), mean)
      solteiro  casado
1oGrau  8.402857  7.044000
2oGrau  8.935000 12.825000
Superior 15.166667 17.783333
```

Além de funções pré-definidas no R, como `mean()` utilizada nos comandos anteriores, podemos definir uma função particular que desejemos. Como exemplo, considere contar em cada grupo, o número de indivíduos que tem salário igual ou superior a 10 unidades, os comandos seriam como mostrado a seguir.

```
> tapply(salario, instrucao, function(x) sum(x >= 10))
  1oGrau  2oGrau Superior
      2      10      6
> tapply(salario, civil, function(x) sum(x >= 10))
solteiro  casado
      7      11
> tapply(salario, interaction(instrucao, civil), function(x) sum(x >= 10))
  1oGrau.solteiro  2oGrau.solteiro Superior.solteiro  1oGrau.casado
      2      2      3      0
  2oGrau.casado  Superior.casado
      8      3
```

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)           # desanexando do caminho de procura
```

8.3 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 úteis para termos uma ideia 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 você vai ter que teclar **ENTER** para ver o próximo gráfico.

- no “prompt” do programa R digite:

```
> demo(graphics)
```

- Você 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 você pode visualizar na janela gráfica o gráfico produzido pelos comandos mostrados anteriormente. Inspeção 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 você 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"))
```

8.4 Outros dados disponíveis no R

Há vários conjuntos de dados incluídos no programa R como, por exemplo, o conjunto `mtcars`. Estes conjuntos são todos documentados, isto é, você 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) # carrega o conjunto de dados women
> women      # mostra os dados
> help(women) # mostra a documentação destes dados
```

8.5 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 a documentação da função `mean()`. Note que no final da documentação há exemplos de uso da função que você pode reproduzir para entendê-la melhor.

8.6 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 capita de cada tipo de crime.
 - (j) encontre os estados com maior e menor ocorrência do total de crimes.

9 Gráficos no R

9.1 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 úteis para termos uma ideia 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 você vai ter que teclar ENTER para ver o próximo gráfico.

- no “prompt” do programa R digite:

```
> demo(graphics)
```

- Você 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 você pode visualizar na janela gráfica o gráfico produzido pelos comandos mostrados anteriormente. Inspeção 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* você 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 produzi-los

9.2 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 permite 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. O comando `layout()` também permite dividir a janela gráfica de forma ainda mais flexível.

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 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` (ou `.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 por exemplo `wmf()` para o sistema operacional WINDOWS. Cada uma das funções possui 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 encerrado.

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)
```

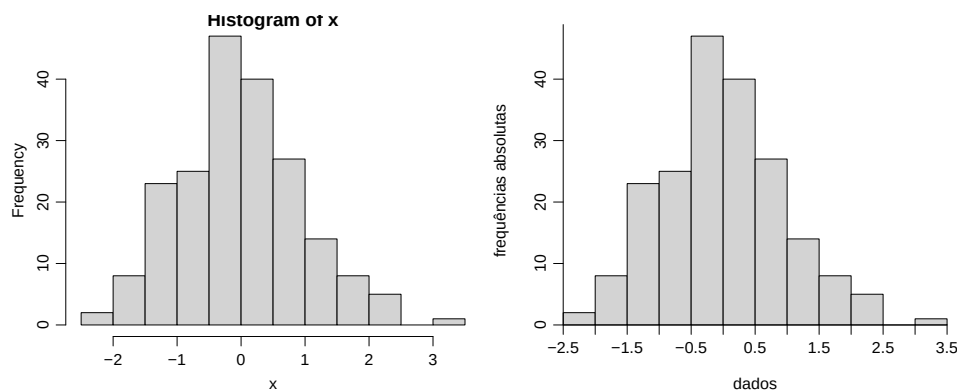


Figura 12: Histograma gerado com opções padrão (esquerda) e modificadas (direita).

Vejamos na Figura 12 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 abscissas e ordenadas, respectivamente.

9.3 Alguns exemplos

Gráfico com dois eixos verticais distintos Considere fazer um gráfico de duas variáveis de grandezas distintas, $Y1$ e $Y2$ contra uma mesma variável X . Isso pode ser útil, por exemplo, para ver se as flutuações são comuns com a variação nos valores de X . Gostaríamos de fazer um gráfico destes colocando eixos distintos para $Y1$ e $Y2$, um à esquerda e outro a direita do gráfico.

Vamos considerar o conjunto de dados `airquality` já disponível no R que possui medidas de Ozônio, radiação solar, velocidade do vento e temperatura em Nova York de Maio a Setembro de 1973.

```
> data(airquality)
> head(airquality)
  Ozone Solar.R Wind Temp Month Day
1   41    190  7.4   67     5   1
2   36    118  8.0   72     5   2
3   12    149 12.6   74     5   3
4   18    313 11.5   62     5   4
5   NA     NA 14.3   56     5   5
6   28     NA 14.9   66     5   6
```

Nesses dados, as informações sobre as datas de coleta estão nas duas últimas colunas. Vamos inicialmente criar uma nova variável com a representação da data.

```
> require(date)
Carregando pacotes exigidos: date
> airquality$dates <- as.date(with(airquality, paste(Month, Day, "2003", sep="/")))
> summary(airquality)
```

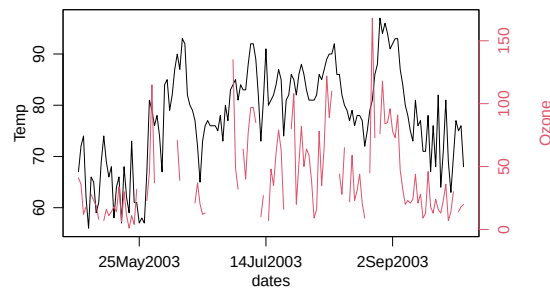


Figura 13: Ilustração de gráfico com duas escalas para o eixo-*y*: evolução dos valores de temperatura e ozônio.

Ozone	Solar.R	Wind	Temp
Min. : 1.00	Min. : 7.0	Min. : 1.700	Min. : 56.00
1st Qu.: 18.00	1st Qu.: 115.8	1st Qu.: 7.400	1st Qu.: 72.00
Median : 31.50	Median : 205.0	Median : 9.700	Median : 79.00
Mean : 42.13	Mean : 185.9	Mean : 9.958	Mean : 77.88
3rd Qu.: 63.25	3rd Qu.: 258.8	3rd Qu.: 11.500	3rd Qu.: 85.00
Max. : 168.00	Max. : 334.0	Max. : 20.700	Max. : 97.00
NAs : 37	NAs : 7		
Month	Day	dates	
Min. : 5.000	Min. : 1.0	First : 1May2003	
1st Qu.: 6.000	1st Qu.: 8.0	Last : 30Sep2003	
Median : 7.000	Median : 16.0		
Mean : 6.993	Mean : 15.8		
3rd Qu.: 8.000	3rd Qu.: 23.0		
Max. : 9.000	Max. : 31.0		

Nos comandos a seguir criamos o gráfico da Figura 13 com as evoluções das medidas de temperatura e níveis de ozônio no período, colocando a escala de cada uma delas em um dos eixos verticais.

```
> par(mar=c(4,4,4,4))
> with(airquality, plot(Temp ~ dates, type="l"))
> par(new=T)
> with(airquality, plot(Ozone ~ dates, type="l", axes=F, frame=T, ann=F, col=2))
> axis(4, col.axis=2, col=2)
> mtext("Ozone", side=4, line=3, col=2)
```

10 Análise descritiva de tabelas de contingência

10.1 Tabelas para dois ou mais fatores

Vamos utilizar aqui o conjunto de dados `milsa` de Bussab & Morettin discutidos na Seção 8.2 e que pode ser obtido conforme comando a seguir. Repetimos aqui o preparo inicial dos dados convertendo as variáveis categóricas em *fatores* do R e criando a variável `idade` a partir de `ano` e `mes`.

```
> milsa <- read.table("http://www.leg.ufpr.br/Rembrapa/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 ocorrência nos cruzamentos de variáveis. A seguir mostramos algumas opções da visualização 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. A classe do objeto resultante vai depender da função utilizada. Nesse 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 estão 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		
civil	1oGrau	2oGrau	Superior
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 <- ftable(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 entre 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
```

O objeto retornado por `table()` não é uma matriz, 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 6×3 onde $6 = 2 \times 3$ indicando o produto do número de níveis 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 matriz de dimensão 6×3 .

```
> 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

10.2 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 apenas 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 `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 classes. Da mesma forma, no caso de duas variáveis, as *margens* das tabelas obtidas de uma ou outra forma são as mesmas. O valor numérico retornado por `margin.table()` é idêntico para `t1` e `t1f`, ainda que, em versões recentes do R, imprimir o resultado de `margin.table()` aplicado a um `ftable` totalmente colapsado (reduzido a uma única dimensão) provoque um erro interno em `print.ftable()`; por isso abaixo mostramos a impressão apenas para o objeto `t1`.

```
> 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)
```

	instrucao	1oGrau	2oGrau	Superior
civil				
solteiro		0.19444444	0.16666667	0.08333333
casado		0.13888889	0.33333333	0.08333333

```
> prop.table(t1, margin=1)
```

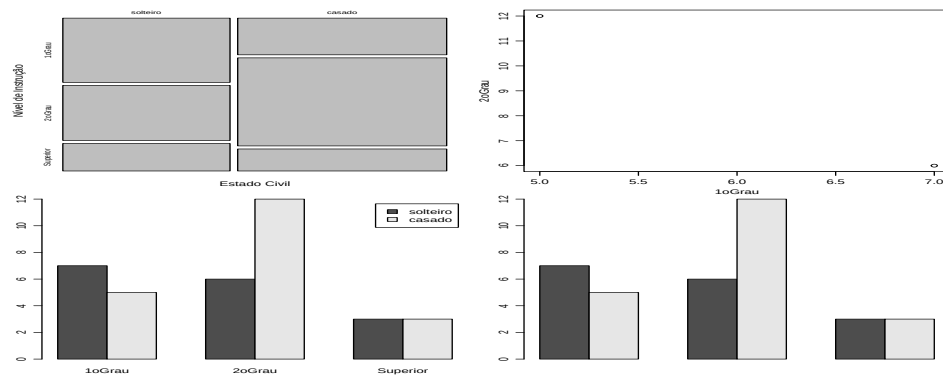


Figura 14: Representações gráficas de tabelas de contingência de duas variáveis obtidas pelas funções `table()` e `fable()`.

```

      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)
      instrucao 1oGrau 2oGrau Superior
civil
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) produz o mesmo resultado numérico de
> ## margin.table(t1, mar=1), mas sua impressão falha em versões
> ## recentes do R devido a um problema em print.fable() ao lidar
> ## com um ftable de uma única dimensão.
> margin.table(t1, mar=2)
Nível de Instrução
  1oGrau  2oGrau Superior
      12      18      6
> ## margin.table(t1f, mar=2) -- mesma limitação de impressão acima.

```

Da mesma forma os gráficos obtidos são os mesmos. A Figura 14 mostra dois tipos de gráficos. Os gráficos na parte superior mostram retângulos cujas áreas são proporcionais às frequências e abaixo um dentre os possíveis gráficos de barras.

```

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

```

Já para três ou mais variáveis os resultados são bem diferentes, em particular para as frequências marginais, uma vez que `fable()` 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

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

, , regiao = outro

      instracao
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  instracao
solteiro 1oGrau      0.05555556 0.05555556 0.08333333
        2oGrau      0.02777778 0.02777778 0.11111111
        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

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

, , regiao = interior

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

, , regiao = outro

      instracao
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  instracao
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

```

```

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

```

```
, , regiao = interior
```

```

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

```

```
, , regiao = outro
```

```

      instracao
civil      1oGrau      2oGrau      Superior
solteiro 0.23076923 0.30769231 0.07692308
casado   0.15384615 0.15384615 0.07692308

```

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

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

```

civil
solteiro  casado
      16      20

```

```
> margin.table(t2, mar=2)
```

```

instracao
  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)
```

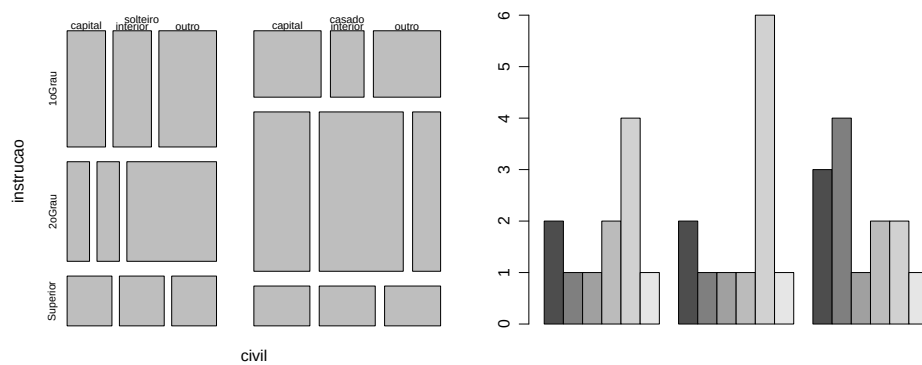


Figura 15: Representações gráficas de tabelas de contingência de três variáveis obtidas pelas funções `table()` (esquerda) e `ftable()` (direita).

11 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)

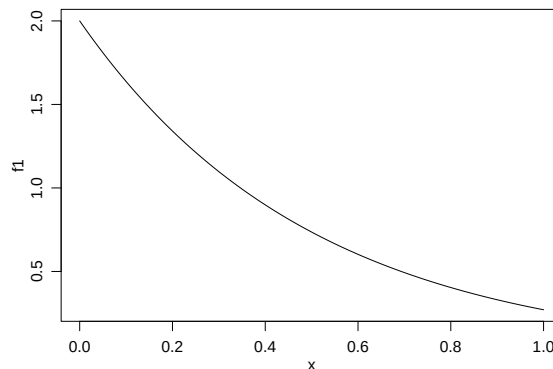
Dada a função

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

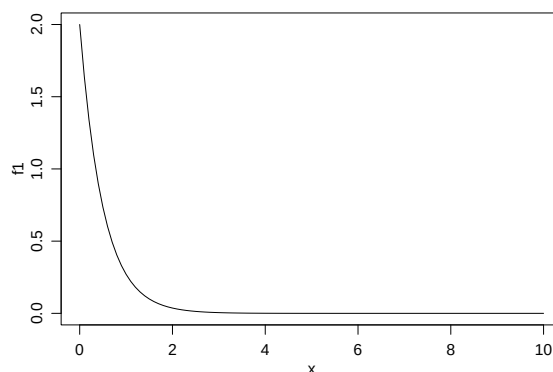
- (a) mostre que esta função é uma f.d.p.
- (b) calcule a probabilidade de que $X > 1$
- (c) calcule a probabilidade de que $0.2 < X < 0.8$

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 16](#) 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)
```

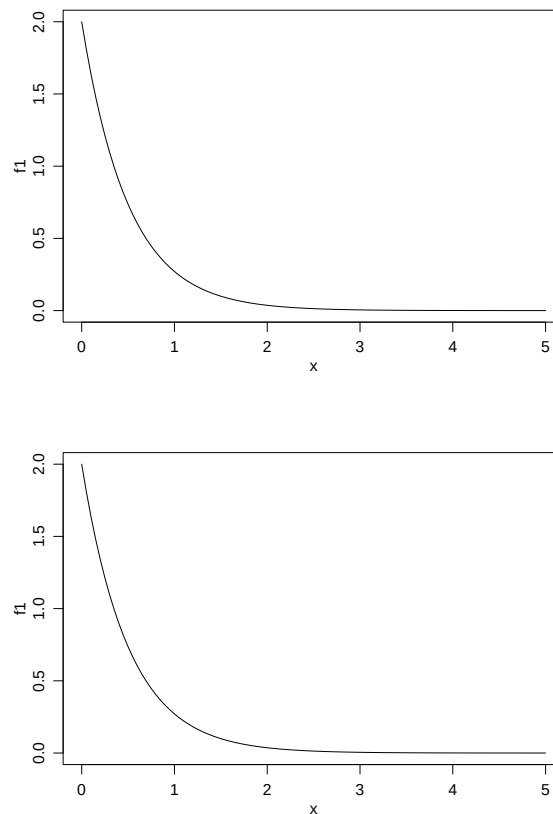


Figura 16: Gráfico da função de probabilidade do Exemplo 1.

Para verificar que 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

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

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

cuja representação gráfica é mostrada na [Figura 17](#). 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 sombreadar 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.

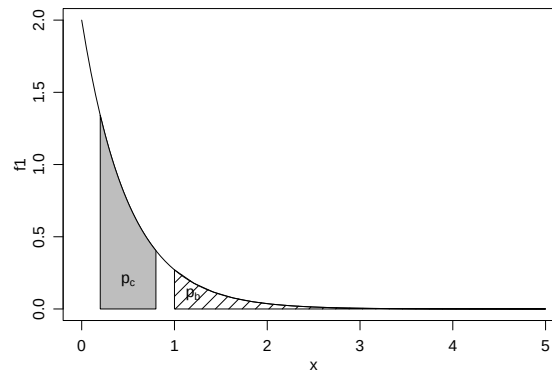


Figura 17: Probabilidades pedidas nos itens (b) e (c) do Exemplo 1.

```
> 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])))
```

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.

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

- Calcular a probabilidade de que sejam vendidos mais que 150 kg.
- Calcular a venda esperada em 30 dias.
- 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 18](#).

1 with absolute error < 1.1e-15

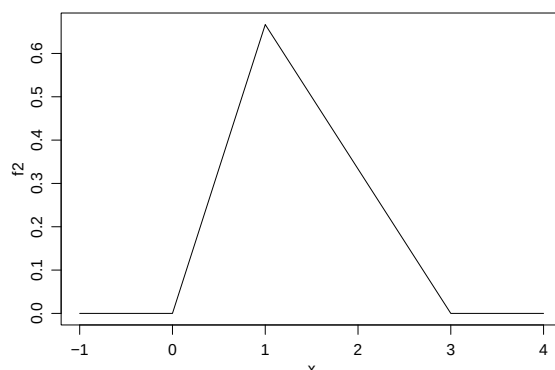


Figura 18: Gráfico da função densidade de probabilidade do Exemplo 2.

```
> integrate(f2, 0, 3) ## verificando que a integral vale 1
1 with absolute error < 1.1e-15
> plot(f2, -1, 4)      ## fazendo o gráfico da função
```

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)
1.333333 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 19 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)
```

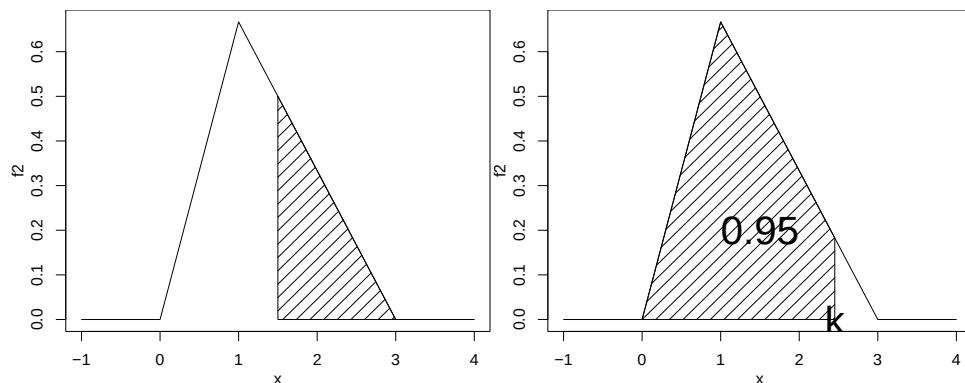


Figura 19: Gráficos indicando as soluções dos itens (a) e (c) do Exemplo 2.

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.

11.1 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?

12 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 as funções deve-se combinar uma das letras acima com uma abreviatura do nome da distribuição. Por exemplo, para calcular probabilidades usamos: `pnorm()` para distribuição normal, `pexp()` para exponencial, `pbinom()` para binomial, `ppois()` para Poisson e assim por diante. Vamos ver com mais detalhes algumas distribuições de probabilidades.

12.1 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] 0.20655549 -0.64716079 -0.25471700 0.03371804 0.11033425 0.98218218
[7] -0.60410167 0.22780774 0.66930840 1.07551727
```

O primeiro valor corresponde ao valor da densidade da normal

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

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 acumulada $P(X \leq -1)$. O comando `qnorm(0.975)` calcula o valor de um quantil 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 você 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 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.

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

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

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

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

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

Note que a última probabilidade foi calculada de duas formas diferentes, a segunda usando o argumento `lower` que implementa um algoritmo 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 a seguir e verifique os gráficos por eles produzidos. A [Figura 20](#) mostra gráficos da densidade (esquerda) e probabilidade acumulada (direita) da normal padrão. Para fazer o gráfico consideramos valores de X entre -3 e 3 que correspondem a $\pm$ três desvios padrões da média, faixa que concentra 99,73% da massa de probabilidade da distribuição normal.

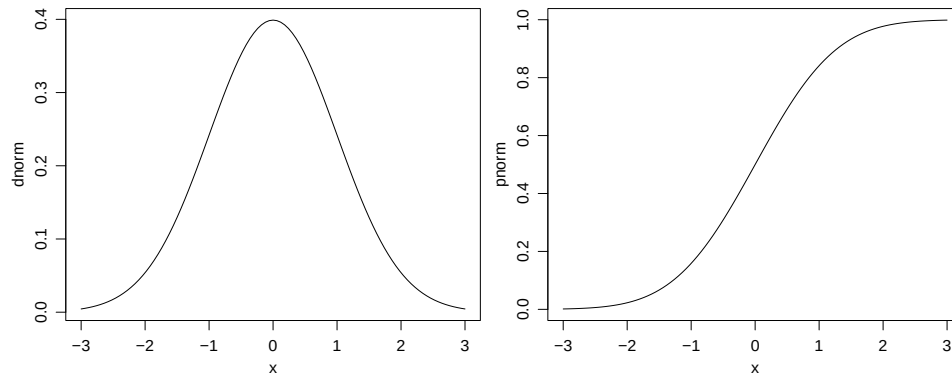


Figura 20: Funções de densidade e probabilidade da distribuição normal padrão.

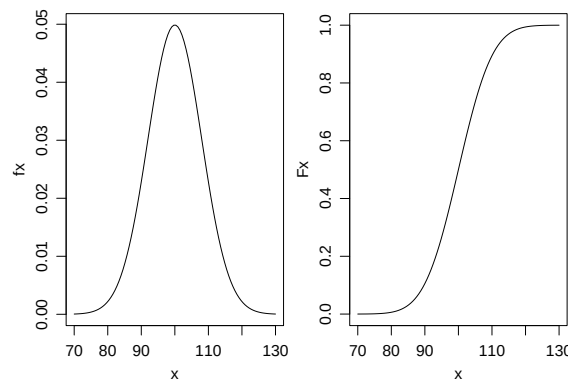


Figura 21: Funções de densidade de probabilidade (esquerda) e função de distribuição acumulada (direita) da $N(100, 64)$.

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

A [Figura 21](#) mostra gráficos da densidade (esquerda) e probabilidade acumulada (direita) da $N(100, 64)$. Para fazer estes gráficos tomamos uma sequência de valores de x entre 70 e 130 e para cada um deles calculamos o valor das funções $f(x)$ e $F(x)$. Depois unimos os pontos $(x, f(x))$ em um gráfico e $(x, F(x))$ no outro.

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

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

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

ou ainda

```
> curve(dnorm(x, 100, 8), 70, 130)
> curve(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 no gráfico da esquerda da [Figura 22](#) e nos dois primeiros comandos abaixo. Os demais comandos mostram

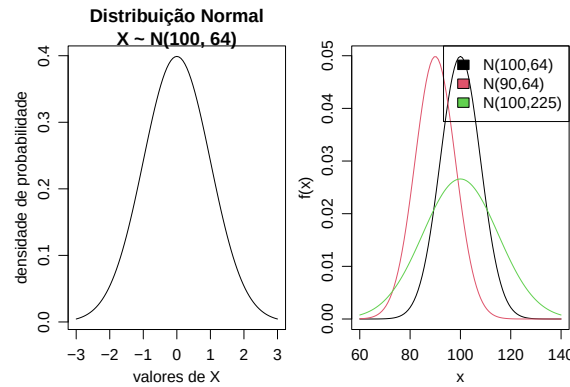


Figura 22: Gráfico com texto nos eixos e título (esquerda) e várias distribuições em um mesmo gráfico (direita).

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('Distribuição Normal\nX ~ N(100, 64)')
>
> curve(dnorm(x, 100, 8), 60, 140, ylab='f(x)')
> curve(dnorm(x, 90, 8), 60, 140, add=T, col=2)
> curve(dnorm(x, 100, 15), 60, 140, add=T, col=3)
> legend("topright", c("N(100,64)", "N(90,64)", "N(100,225)"), fill=1:3)
```

12.2 Distribuição Binomial

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

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

> help(dbinom)
```

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

1. fazer o gráfico da 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 $P[3 < X \leq 6] = P[4 \leq X < 7]$

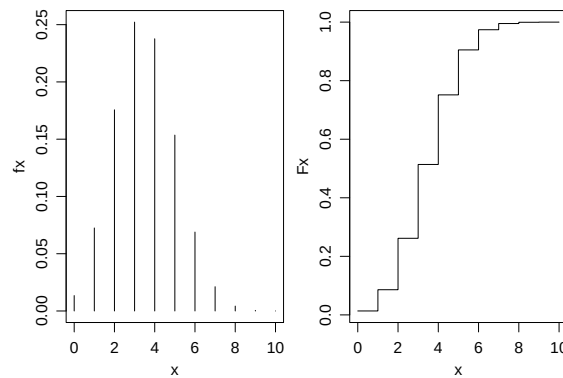


Figura 23: Funções de probabilidade (esquerda) e distribuição acumulada (direita) da $B(10, 0.35)$.

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 23](#).

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

As probabilidades pedidas são obtidas com os comandos a seguir.

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

12.3 Distribuição Uniforme

12.3.1 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] 0.7163657 0.5931028 0.9449806 0.1846381 0.5385011
> 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] 8.737943 10.087838 17.835511 13.626632 16.895792
```

12.3.2 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 *q*. Provavelmente isso é devido a sua simplicidade, embora algumas outras funções possam ser usadas. Por exemplo para sortear números pode-se usar `sample()`. No exemplo a seguir são sorteados 15 valores de uma uniforme discreta com valores (inteiros) entre 1 e 10 ($X \sim U_d(1, 10)$).

```
> sample(1:10, 15, rep=T)
[1] 3 9 7 6 1 6 6 5 8 3 1 2 5 6 9
```

12.4 A função `sample()`

A função `sample()` **não** é restrita à distribuição uniforme discreta, podendo ser usada para sorteios, com ou sem reposição (conforme indicado pelo argumento `replace`, com *default* sem reposição) e ainda 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] 16 14 4
```

- 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] 12 34 23 20 28
```

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

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

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

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

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

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

O 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.

12.5 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:

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

- da função de densidade de uma variável com distribuição de Poisson com parâmetro $\lambda = 5$
 - da densidade de uma variável $X \sim N(90, 100)$
 - sobreponha ao gráfico anterior a densidade de uma variável $Y \sim N(90, 80)$ e outra $Z \sim N(85, 100)$
 - densidades de distribuições χ^2 com 1, 2 e 5 graus de liberdade.
5. A probabilidade de indivíduos nascerem com certa característica é de 0,3. Para o nascimento de 5 indivíduos e considerando os nascimentos como eventos independentes, estude o comportamento da variável *número de indivíduos com a característica* e faça um gráfico de sua função de distribuição.
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.

13 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.

13.1 Probabilidades e integrais

A probabilidade de um evento em uma distribuição contínua é uma área sob a curva da distribuição. Vamos reforçar esta ideia 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

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

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 24 usamos o fato que a maior parte da função está no intervalo entre a média $\pm$ 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)
```

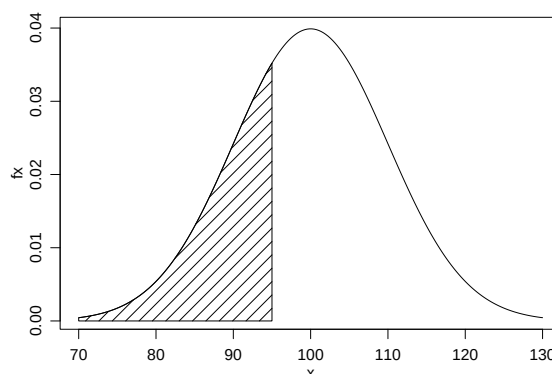


Figura 24: Funções de densidade da $N(100, 100)$ com a área correspondente à $P[X \leq 95]$.

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 o erro de aproximação numérica.

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

Portanto, para os demais itens 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 anteriormente 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()`.

13.2 Distribuição exponencial

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

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

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

A solução 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. O resultado deve coincidir com o 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 25 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)
```

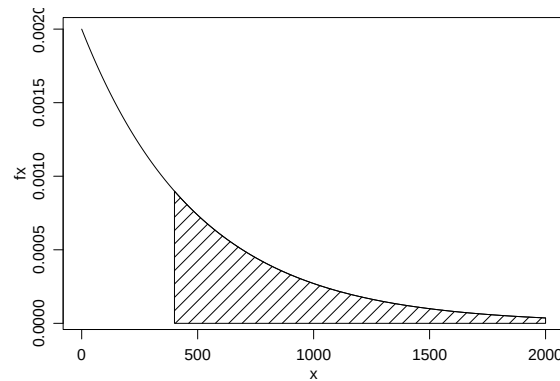


Figura 25: Função de densidade da $Exp(500)$ com a área correspondente à $P[X \geq 400]$.

13.3 Esperança e Variância

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

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

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

13.4 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)      # define o valor da semente
> rnorm(5)           # amostra de 5 elementos
[1] -0.46774980  0.04088223  1.00335193  2.02522505  0.30640096
> rnorm(5)           # outra amostra de 5 elementos
```

```
[1] 0.4257775 0.7488927 0.4464515 -2.2051418 1.9818137
> set.seed(214)      # retorna o valor da semente ao valor inicial
> rnorm(5)           # gera novamente a primeira amostra de 5 elementos
[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 e 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)`. Para assegurar a reprodutibilidade entre sistemas operacionais é recomendado ainda definir o gerador de números aleatórios antes como a seguir.

```
> RNGkind("Mersenne-Twister") # define o algoritmo gerador
> set.seed(214)               # define o valor da semente
> rnorm(5)                    # amostra de 5 elementos
[1] -0.46774980 0.04088223 1.00335193 2.02522505 0.30640096
```

13.5 Argumentos vetoriais e lei da reciclagem

As funções de probabilidades aceitam também vetores em seus argumentos conforme ilustrado nos exemplos a seguir.

```
> 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).

13.6 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 tal aproximação é menos importante, visto que é fácil calcular as probabilidades exatas com o auxílio do computador. Ainda assim 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 26 mostra o gráfico da distribuição binomial e da aproximação pela normal.

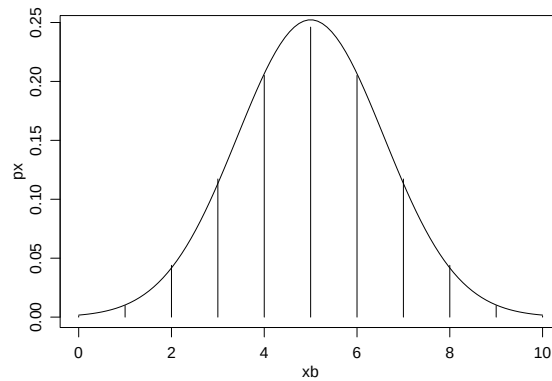


Figura 26: Função de probabilidade da $B(10, 1/2)$ e a aproximação pela $N(5, 2.5)$.

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

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

- $P[X < 6]$

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

```
> pbinom(5, 10, 0.5)
[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 $P[X_B \geq 2] = 1 - P[X_B \leq 1] \approx 1 - P[X_N \leq 1.5]$

```
> 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
```

13.7 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 analiticamente e computacionalmente.

Dica: para resolver você vai precisar da definição da função Gama:

$$\Gamma(a) = \int_0^\infty x^{a-1} e^{-x} dx$$

- (b) Obter $E[X]$ para $\lambda = 5$.

- (c) Obter as probabilidades:

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

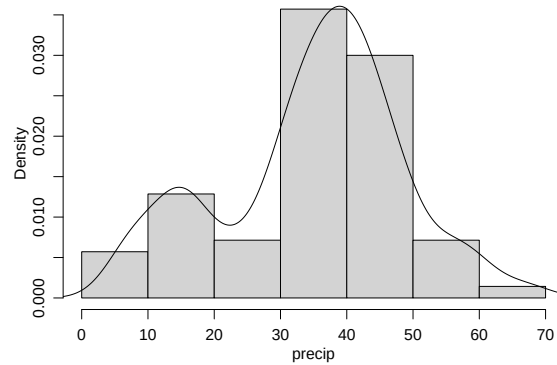


Figura 27: Histograma para os dados `precip` e a densidade estimada usando a função `density`.

14 Explorando distribuições de probabilidade empíricas

Na Sessão 12 vimos como usar distribuições de probabilidade no R. Estas distribuições têm 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 da 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 uma 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.

14.1 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, conforme mostra a Figura 27.

```
> data(precip)
> hist(precip, prob=T, main="")
> 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 28 uma outra forma de

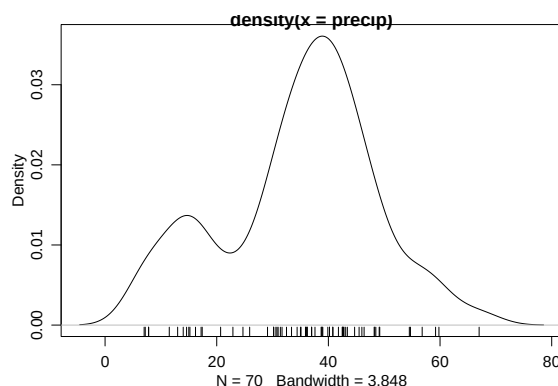


Figura 28: Dados `precip` e a densidade estimada usando a função `density`.

visualizar os dados e sua densidade estimada, agora sem fazer o histograma.

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

Embora os resultados mostrados acima sejam 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.

```
> 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 29. Podemos notar que o grau de suavização aumenta à medida que 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 o tipo de função de pesos, ao que chamamos de núcleo (*kernel*). O R implementa vários núcleos diferentes cujos formatos são mostrados na Figura 30.

```
> (kernels <- eval(formals(density.default)$kernel))
> plot(density(0, bw = 1), xlab = "", main="kernels com bw = 1")
> for(i in 2:length(kernels))
+   lines(density(0, bw = 1, kern = kernels[i]), col = i)
> legend(1.5,.4, legend = kernels, col = seq(kernels),
+       lty = 1, cex = .8, y.int = 1)
```

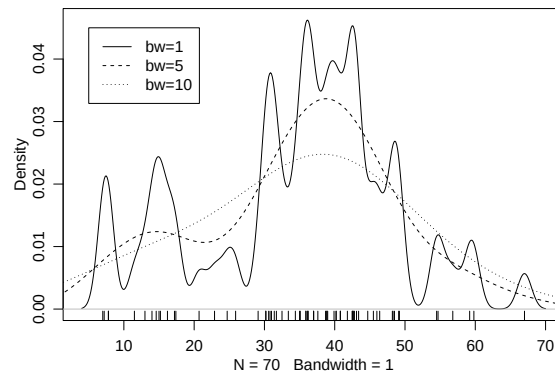


Figura 29: Densidade estimada usando a função `density` com diferentes valores para o argumento `bw`.

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

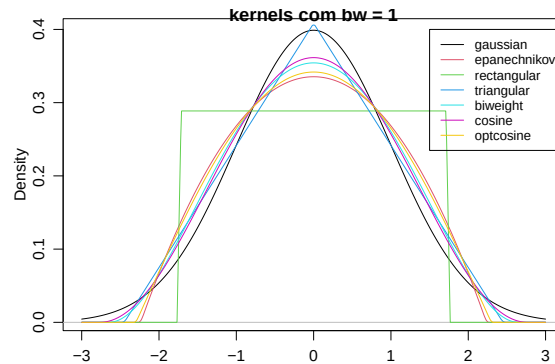


Figura 30: Diferentes núcleos implementados pela função `density`.

Utilizando diferentes núcleos no conjunto de dados `precip` obtemos os resultados mostrados na Figura 31. 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 sensível. O tipo de núcleo (*kernel*) é de importância secundária.

Bem, a esta altura você 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 se comportam bem na maioria das situações práticas. Estes métodos podem ser especificados no mesmo

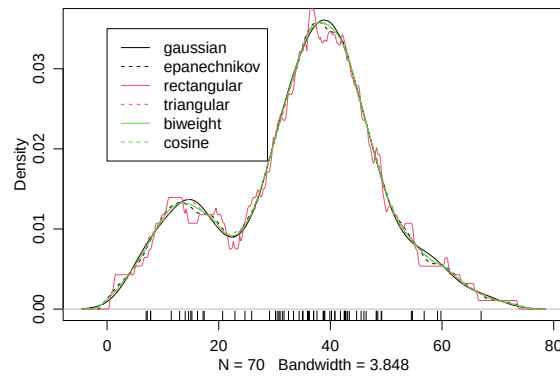


Figura 31: Densidade estimada usando a função `density` com diferentes valores para o argumento `kernel`.

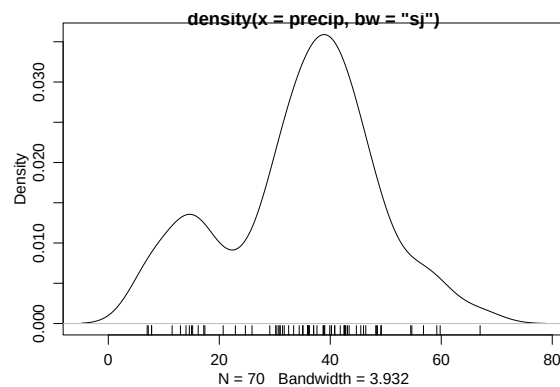


Figura 32: Densidade estimada para os dados `precip` usando a função `density` com critério de Sheather & Jones para seleção da largura de banda.

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 32.

```
> 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 33 ilustramos resultados obtidos com os diferentes métodos.

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

Warning in bw.bcv(x): mínimo ocorreu em uma das extremidades do intervalo

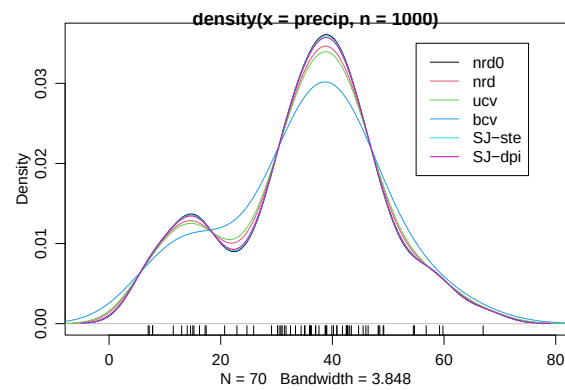


Figura 33: Diferentes métodos para largura de banda implementados pela função `density`.

14.2 Exercícios

1. Carregar o conjunto de dados `faithful` e obter estimacão de densidade para as variáveis tempo de erupção (`waiting`) e duração da erupção (`eruptions`).
2. Carregar o conjunto `airquality` e obter 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()`.

15 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.

15.1 Média de uma distribuição normal com variância desconhecida

Considere o seguinte problema:

Exemplo

O tempo de reação de um novo medicamento pode ser considerado como tendo distribuição Normal e deseja-se fazer inferência sobre a média que é desconhecida obtendo um intervalo de confiança. Vinte pacientes foram sorteados e tiveram seu tempo de reação anotado. Os dados foram os seguintes (em minutos):

2.9	3.4	3.5	4.1	4.6	4.7	4.5	3.8	5.3	4.9
4.8	5.7	5.8	5.0	3.4	5.9	6.3	4.6	5.5	6.2

Entramos com os dados com o comando

```
> tempo <- c(2.9, 3.4, 3.5, 4.1, 4.6, 4.7, 4.5, 3.8, 5.3, 4.9,
+           4.8, 5.7, 5.8, 5.0, 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:

$$\left(\bar{x} - t_t \sqrt{\frac{S^2}{n}} \quad , \quad \bar{x} + t_t \sqrt{\frac{S^2}{n}} \right)$$

em que 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.

15.1.1 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] 0.9920789
```

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
```

15.1.2 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 utilizado várias vezes.

15.1.3 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 seguir, que coincide com os obtidos anteriormente.

```
> tt.tempo <- t.test(tempo)
> tt <- t.test(tempo)
> str(t.test(tempo))
List of 10
 $ statistic   : Named num 21.3
  .. attr(*, "names")= chr "t"
 $ parameter   : Named num 19
  .. attr(*, "names")= chr "df"
 $ p.value     : num 1.01e-14
 $ conf.int    : num [1:2] 4.28 5.21
  .. attr(*, "conf.level")= num 0.95
 $ estimate    : Named num 4.75
  .. attr(*, "names")= chr "mean of x"
 $ null.value  : Named num 0
  .. attr(*, "names")= chr "mean"
 $ stderr      : num 0.223
 $ alternative : chr "two.sided"
 $ method      : chr "One Sample t-test"
 $ data.name   : chr "tempo"
 - attr(*, "class")= chr "htest"
```

```
> tt.tempo$conf.int
[1] 4.278843 5.211157
attr(,"conf.level")
[1] 0.95
```

A chamada da função retorna uma lista que contém também resultados de um teste de hipótese que não é de interesse aqui. Para obter o intervalo de confiança selecionamos o elemento desejado da lista.

15.2 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 esquistossomose. 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 duas 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.

16 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.

16.1 Definições e notações

Seja $L(\theta; y)$ a função de verossimilhança. A notação indica que o argumento da função é θ que pode ser um escalar ou um vetor de parâmetros. Nesta sessão consideraremos que é um escalar. O termo y denota valores realizados de uma variável aleatória Y , isto é os valores obtidos em uma amostra.

O valor que maximiza $L(\theta; y)$ é chamado do estimador de máxima verossimilhança e denotado por $\hat{\theta}$. A função de verossimilhança *relativa* ou *normalizada* $R(\theta; y)$ 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 os modelos dados pelos diferentes valores de θ com o modelo mais plausível (verossível) para a amostra obtida.

O valor que maximiza a função de verossimilhança é também o que maximiza a a função obtida pelo logaritmo da função de verossimilhança, chamada função de log-verossimilhança, uma vez que a função logaritmo 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 logaritmo 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*, $D(\theta; y) = -2 [l(\theta; y) - l(\hat{\theta}; y)]$, onde lembramos que $\hat{\theta}$ é o estimador de máxima verossimilhança de θ . Esta função tem portanto o seu mínimo em zero e quanto maior o seu valor, maior a diferença de plausibilidade entre o modelo considerado e o modelo mais plausível para os dados obtidos na amostra. Esta função combina as vantagens da verossimilhança relativa e da log-verossimilhança sendo portanto conveniente para cálculos computacionais e inferência.

16.2 Exemplo 1: Distribuição normal com variância conhecida

Seja o vetor (12, 15, 9, 10, 17, 12, 11, 18, 15, 13) uma amostra aleatória de uma distribuição normal de média μ e variância conhecida e igual a 4. O objetivo é fazer um gráfico da função de log-verossimilhança.

Solução:

Vejamos primeiro os passos da solução analítica:

1. Temos que $X_1, \dots, X_n$ 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

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

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

Vamos ver agora uma primeira possível forma de fazer a função de verossimilhança no R.

1. Primeiro entramos com os dados que armazenamos no vetor **x**

```
> x <- c(12, 15, 9, 10, 17, 12, 11, 18, 15, 13)
```

2. e calculamos as quantidades $\sum_{i=1}^{10} x_i^2$ e $\sum_{i=1}^{10} x_i$

```
> sx2 <- sum(x^2)
> sx <- sum(x)
```

3. agora tomamos uma sequência de valores para μ . Sabemos que o estimador de máxima verossimilhança neste caso é $\hat{\mu} = 13.2$ (este valor pode ser obtido com o comando `mean(x)`) e portanto vamos definir tomar valores ao redor deste ponto.

```
> mu.vals <- seq(11, 15, l=100)
```

4. e a seguir calculamos os valores de $l(\mu)$ de acordo com a equação acima

```
> lmu <- -5 * log(8*pi) - (sx2 - 2*mu.vals*sx + 10*(mu.vals^2))/8
```

5. e finalmente fazemos o gráfico visto na [Figura 34](#)

```
> plot(mu.vals, lmu, type='l', xlab=expression(mu), ylab=expression(l(mu)))
```

Entretanto podemos obter a função de verossimilhança no R de outras forma mais geral e menos trabalhosa. Sabemos que a função `dnorm()` calcula a densidade $f(x)$ da distribuição normal e podemos usar este fato para evitar a digitação da expressão acima.

- Primeiro vamos criar uma função que calcula o valor da log-verossimilhança para um certo valor do parâmetro e para um certo conjunto de dados,

```
> logvero <- function(mu, dados){
+   sum(dnorm(dados, mean = mu, sd = 2, log = TRUE))
+ }
```

- a seguir criamos uma sequência adequada de valores de μ e calculamos $l(\mu)$ para cada um dos valores

```
> mu.vals <- seq(11, 15.5, l=100)
> mu.vals[1:10]
```

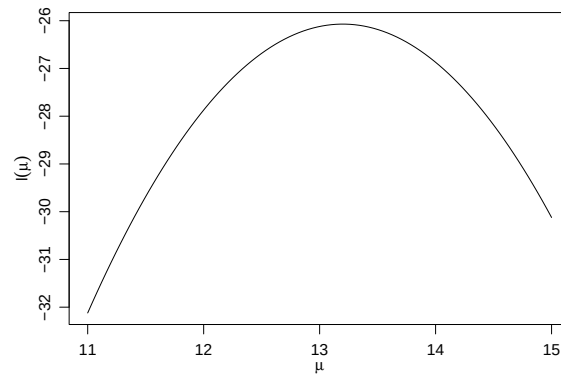


Figura 34: Função de verossimilhança para o parâmetro μ da distribuição normal com variância $\sigma^2 = 4$ com os dados do Exemplo 1.

```
[1] 11.00000 11.04545 11.09091 11.13636 11.18182 11.22727 11.27273 11.31818 11.36364
[10] 11.40909

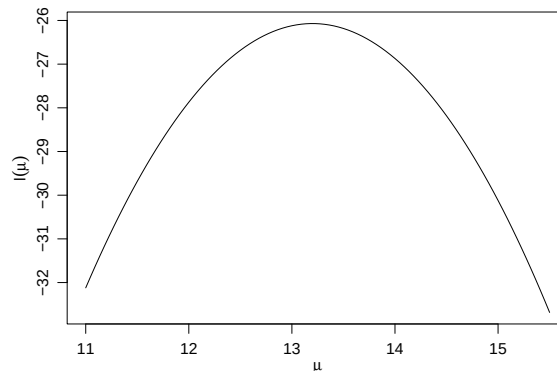
> lmu <- sapply(mu.vals, logvero, dados = x)
> lmu[1:10]

[1] -32.12086 -31.87344 -31.63119 -31.39410 -31.16218 -30.93542 -30.71383 -30.49741
[9] -30.28615 -30.08005
```

Note na sintaxe acima que a função `sapply` aplica a função `logvero` anteriormente definida em cada elemento do vetor `mu.vals`.

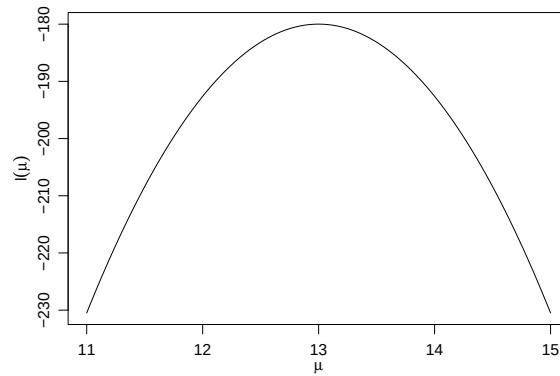
- Finalmente fazemos o gráfico.

```
> 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 μ para qualquer amostra obtida desta distribuição.

```
> vero.normal.v4 <- function(mu, dados){
+   logvero <- function(mu, dados)
+     sum(dnorm(dados, mean = mu, sd = 2, log = TRUE))
+   sapply(mu, logvero, dados = dados)
+ }
> curve(vero.normal.v4(x, dados = x), 11, 15,
+       xlab=expression(mu), ylab=expression(l(mu)))
```



16.3 Exemplo 2: Distribuição Poisson

Considere agora a amostra armazenada no vetor `y`:

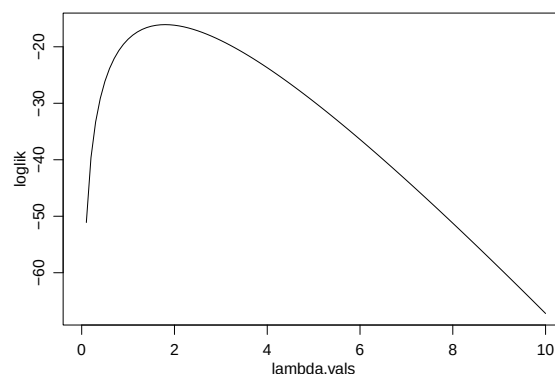
```
> y <- c(5, 0, 3, 2, 1, 2, 1, 1, 2, 1)
```

de uma distribuição de Poisson de parâmetro λ . A função de verossimilhança pode ser definida por:

```
> lik.pois <- function(lambda, dados){
+   loglik <- function(l, dados){sum(dpois(dados, lambda = l, log = TRUE))}
+   sapply(lambda, loglik, dados = dados)
+ }
```

E podemos usar esta função para fazer o gráfico da função de verossimilhança como visto à esquerda da [Figura 35](#)

```
> 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, xlab=expression(lambda),
+       ylab=expression(l(lambda)))
```

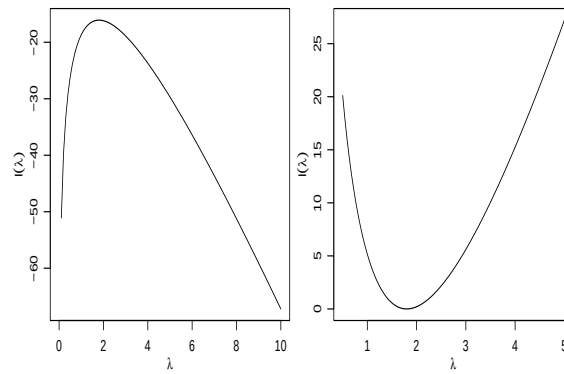
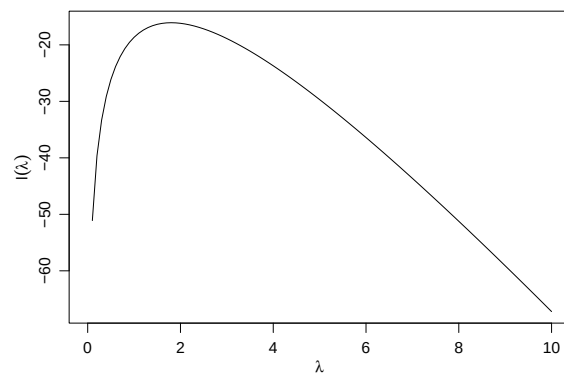
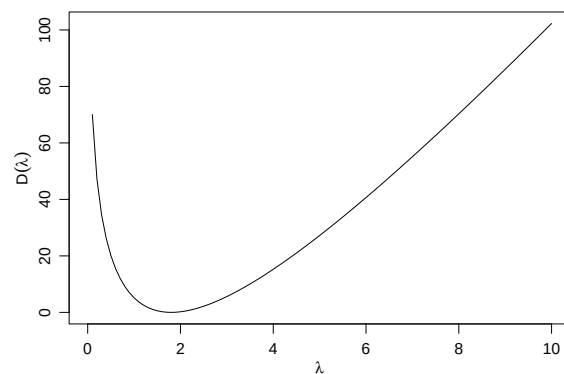


Figura 35: Função de verossimilhança (esquerda) e deviance (direita) para o parâmetro λ da distribuição Poisson.



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, xlab=expression(lambda),
+       ylab=expression(D(lambda)))
```



Ou fazendo novamente em um intervalo menor

```
> curve(dev.pois(x, dados=y), 0.5, 5, xlab=expression(lambda),
+       ylab=expression(l(lambda)))
```

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 pode ser obtido diretamente.

```
> lambda.est <- mean(y)
> lambda.est
[1] 1.8
```

Caso o estimador não tenha expressão fechada pode-se usar maximização (ou minimização) numérica. Para ilustrar isto vamos encontrar a estimativa do parâmetro da Poisson e verificar que o valor obtido coincide com o valor dado pela expressão fechada do estimador. Usamos o função `optimise()` para encontrar o ponto de mínimo da função deviance.

```
> optimise(dev.pois, int=c(0, 10), dados=y)
$minimum
[1] 1.800004

$objective
[1] 1.075264e-10
```

A função `optimise()` é adequada para minimizações envolvendo um único parâmetro. Para dois ou mais parâmetros deve-se usar a função `optim()` ou `nlminb()`.

Finalmente os comandos abaixo são usados para obter graficamente o intervalo de confiança (a 95%) baseado na função deviance.

```
> curve(dev.pois(x, dados=y), 0.8, 3.5,
+       xlab=expression(lambda), ylab=expression(l(lambda)))
> L.95 <- qchisq(0.95, df=1)
> abline(h=L.95)
```

Os limites do intervalo são dados pela interseção dessa função com o valor do quantil da distribuição χ^2 para o nível de significância desejado.

```
> #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])])
> lim.fc <- function(lambda) dev.pois(lambda, dados=y) - L.95
> ic2.lambda <- c(inf=uniroot(lim.fc, c(0, lambda.est))$root,
+               sup=uniroot(lim.fc, c(lambda.est, max(y)))$root)
> ic2.lambda
      inf      sup
1.091267 2.764221
```

E adicionados ao gráfico com

```
> arrows(ic2.lambda, L.95, ic2.lambda, 0, len=0.1)
> text(ic2.lambda, 0, round(ic2.lambda, dig=2), pos=1, cex=0.8, offset=0.3)
```

16.4 Exemplo 3: Distribuição normal com variância desconhecida

Vamos agora revisitar o Exemplo 1 desta seção, usando os mesmos dados porém agora sem assumir que a variância é conhecida. Portanto temos agora dois parâmetros sobre os quais queremos fazer inferência: μ e σ . O objetivo é fazer um gráfico 3-D da função de log-verossimilhança de dois argumentos $l(\mu, \sigma)$.

Solução:

Vejamos primeiro os passos da solução analítica:

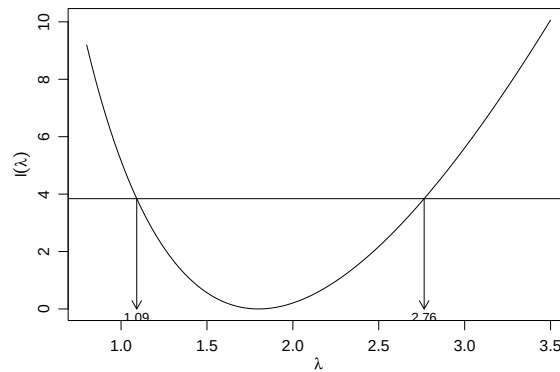


Figura 36: Intervalo de confiança baseado na deviance para o parâmetro λ da distribuição Poisson.

1. Temos que $X_1, \dots, X_n$ onde, neste exemplo $n = 10$, é uma a.a. de $X \sim N(\mu, \sigma^2)$,
2. a densidade para cada observação é dada por $f(x_i) = \frac{1}{\sigma\sqrt{2\pi}} \exp\{-\frac{1}{2\sigma^2}(x_i - \mu)^2\}$,
3. a verossimilhança é dada por $L(\mu, \sigma) = \prod_1^{10} f(\mu, \sigma; x_i)$,
4. e a log-verossimilhança é dada por

$$\begin{aligned} l(\mu, \sigma) &= \sum_1^{10} \log(f(x_i)) \\ &= -5 \log(2\pi\sigma^2) - \frac{1}{2\sigma^2} \left(\sum_1^{10} x_i^2 - 2\mu \sum_1^{10} x_i + 10\mu^2 \right), \end{aligned} \quad (5)$$

5. que é uma função de μ e σ e portanto devemos fazer um gráfico tridimensional de $l(\mu, \sigma)$ versus μ e σ tomando vários valores de pares (μ, σ) e calculando os valores correspondentes de $l(\mu, \sigma)$.

Assim como no Exemplo 1 poderíamos calcular a verossimilhança fazendo as contas "passo a passo" da função acima, ou então usando a função `dnorm()`. Neste exemplo vamos fazer apenas da segunda forma, ficando a primeira como exercício para o leitor.

1. Primeiro entramos com os dados que armazenamos no vetor `x`. Vamos também calcular as estimativas de máxima verossimilhança.

```
> x <- c(12.1, 15.4, 9.8, 10.1, 17.4, 12.3, 11.0, 18.2, 15.4, 13.3, 13.8, 12.7, 15.2, 10.3, 9.9, 1
> pars.MV <- c(mu=mean(x), sd= sqrt(var(x)*(length(x)-1)/length(x)))
> pars.MV

      mu      sd
12.885714 2.248954
```

2. a seguir vamos criar uma função que calcula o valor da log-verossimilhança para um certo par de valores dos parâmetros (média e desvio padrão, nesta ordem) e para um certo conjunto de dados,

```
> logveroN <- function(pars, dados)
+   sum(dnorm(dados, mean = pars[1], sd = pars[2], log = TRUE))
```

3. a seguir criamos uma sequência adequada de pares de valores de (μ, σ) e calculamos $l(\mu, \sigma)$ para cada um dos pares.

```
> par.vals <- expand.grid(mu=seq(5, 20, l=100), sd=seq(1, 12.2, l=100))
> dim(par.vals)

[1] 10000      2

> head(par.vals)

      mu sd
1 5.000000 1
2 5.151515 1
3 5.303030 1
4 5.454545 1
5 5.606061 1
6 5.757576 1

> tail(par.vals)

      mu sd
9995 19.24242 12.2
9996 19.39394 12.2
9997 19.54545 12.2
9998 19.69697 12.2
9999 19.84848 12.2
10000 20.00000 12.2

> par.vals$logL <- apply(par.vals, 1, logveroN, dados = x)
> head(par.vals)

      mu sd      logL
1 5.000000 1 -1208.903
2 5.151515 1 -1167.486
3 5.303030 1 -1126.873
4 5.454545 1 -1087.064
5 5.606061 1 -1048.058
6 5.757576 1 -1009.856
```

Note na sintaxe acima que a função `apply` aplica a função `logveroN` a cada par de valores em cada linha de `par.vals`. Ao final o objeto `|par.vals|` contém na terceira coluna os valores da log-verossimilhança correspondentes aos valores dos parâmetros dados na primeira e segunda colunas.

4. O gráfico 3-D da função pode ser visualizado de três formas alternativas como mostrado na Figura 37: como uma superfície 3D gerada pela função `persp()`, como um mapa de curvas de isovalores obtido com `image()`, ou ainda como um mapa de cores correspondentes aos valores gerado por `image()`.

```
> with(par.vals, persp(unique(mu), unique(sd), matrix(logL, ncol=length(unique(sd))), xlab=expression(mu), ylab=expression(sigma), levels=seq(-120, -75, by=5), ylim=c(0,12)))
> with(par.vals, contour(unique(mu), unique(sd), matrix(logL, ncol=length(unique(sd))), xlab=expression(mu), ylab=expression(sigma), levels=seq(-120, -75, by=5), ylim=c(0,12)))
> points(pars.MV[1], pars.MV[2], pch=4, cex=1.5)
> with(par.vals, image(unique(mu), unique(sd), matrix(logL, ncol=length(unique(sd))), xlab=expression(mu), ylab=expression(sigma), breaks = seq(-120, -75, by=5), col=gray(seq(0.3, 1, length=9))))
> points(pars.MV[1], pars.MV[2], pch=4, cex=1.5)
```

Notas:

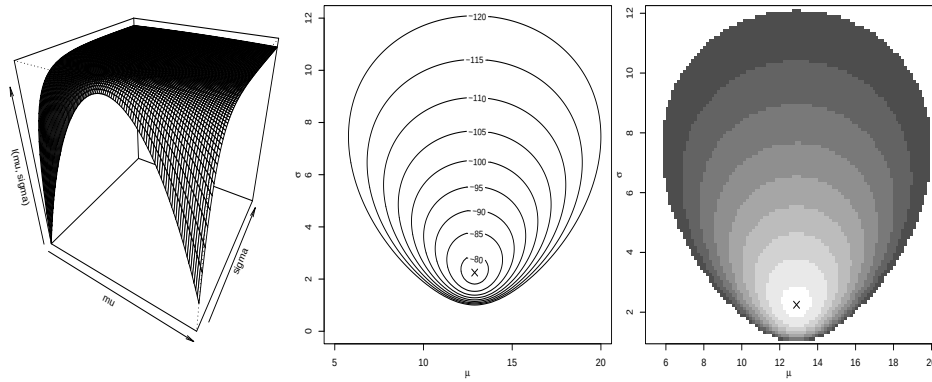


Figura 37: Função de verossimilhança para os parâmetros μ e σ da distribuição normal com os dados do Exemplo 1.

- a obtenção da função foi necessário especificar faixas de valores para μ e σ . A definição desta faixa foi feita após várias tentativas pois depende do problema, em especial do número e variabilidade dos dados.
- as funções gráficas utilizadas requerem: dois vetores de tamanhos n_1 e n_2 com os valores dos argumentos da função e os valores da função em uma matriz de dimensão $n_1 \times n_2$. Por isto usamos `unique()` para extrair os valores dos argumentos, sem repeti-los e `matrix()` para os valores da função.
- na função `persp()` as argumentos `theta` e `phi` são utilizados para rotacionar o gráfico a fim de se obter uma melhor visualização.
- o valor das estimativas de máxima verossimilhança são indicados por `x` nos dois últimos gráficos. Neste caso eles foram encontrados facilmente como mostrado acima no objeto `pars.MV` pois podem ser obtidos analiticamente. De forma mais geral, a função `fitdistr()` do pacote **MASS** pode ser usada para encontrar estimativas de máxima verossimilhança.

```
> require(MASS)

Carregando pacotes exigidos: MASS

> MV <- fitdistr(x, "normal")
> MV

      mean      sd
12.8857143  2.2489544
( 0.3801427) ( 0.2688015)
```

16.5 Exercícios

1. Seja a amostra abaixo obtida de uma distribuição Poisson de parâmetro λ .
5 4 6 2 2 4 5 3 3 0 1 7 6 5 3 6 5 3 7 2
Obtenha o gráfico da função de log-verossimilhança.
2. Seja a amostra abaixo obtida de uma distribuição Binomial de parâmetro p e com $n = 10$.
7 5 8 6 9 6 9 7 7 7 8 8 9 9 9
Obtenha o gráfico da função de log-verossimilhança.

3. Seja a amostra abaixo obtida de uma distribuição χ^2 de parâmetro ν .
8.9 10.1 12.1 6.4 12.4 16.9 10.5 9.9 10.8 11.4
Obtenha o gráfico da função de log-verossimilhança.

17 Intervalos de confiança e função de verossimilhança

Nesta sessão vamos examinar um pouco mais a 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.

Serão utilizados conceitos do método da quantidade pivotal, a propriedade de normalidade assintótica dos estimadores de máxima verossimilhança e a distribuição limite da função deviance.

17.1 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

Desejamos obter:

- o gráfico da função de verossimilhança para p com base nestes dados
- o estimador de máxima verossimilhança de p , a informação observada e a informação de Fisher
- um intervalo de confiança de 95% para p baseado na normalidade assintótica de $\hat{p}$
- compare o intervalo obtido em (b) com um intervalo de confiança de 95% obtido com base na distribuição limite da função deviance
- 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)
```

(a)

Vamos escrever uma função em R para obter a função de verossimilhança usando a função de densidade da distribuição binomial com argumento `log=TRUE` para obter a log-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 a poder utilizar dados disponíveis tanto em um formato quanto em outro.

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

Para obter o gráfico da função de verossimilhança de um conjunto de dados cria-se uma sequência de valores para o parâmetro p e calcula-se o respectivo valor da (log)verossimilhança. O gráfico da função é obtido com os valores fixados dos parâmetros no eixo- x e o respectivos valores da função no eixo- y e unindo-se os pontos assim obtidos. No R isto pode ser feito com os comandos abaixo que produzem o gráfico mostrado na [Figura 38](#). Evitamos os valores nos extremos do espaço paramétrico ($p = 0$ ou $p = 1$) pois nestes casos a verossimilhança é zero e portanto a log-verossimilhança retornada por `dbinom()` é `-Inf`.

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

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)
```

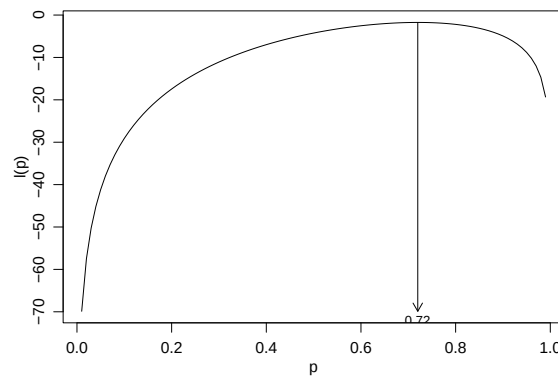


Figura 38: Função de verossimilhança para o parâmetro p da distribuição Bernoulli.

(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 informação 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()` e para obter os valores numéricos para a amostra dada utilizamos os comandos a seguir.

```
> p.est <- mean(y)
> 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 e mostrar em um gráfico o intervalo baseado na função deviance. Lembrando que a deviance é definida pela expressão

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

definimos uma função `dev.binom()` para calcular a deviance. Com o comando `curve()` podemos obter o gráfico de função deviance.

```
> 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
+ }
> curve(dev.binom(x, dados=y), 0.35, 0.95, xlab="p", ylab="D(p)")
```

A função deviance $D(p)$ tem distribuição assintótica χ^2_{n-1} e o intervalo de confiança é dado pelos pontos de intersecção entre a função deviance e o valor de quantil da distribuição χ^2 para o nível de significância desejado como ilustrado na Figura 39. 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()` e os pontos de corte que definem o intervalo são as raízes de uma função definida como a diferença entre a função deviance e o valor do ponto de corte.

```
> L.95 <- qchisq(0.95, df=1)
> #dif <- abs(dev.p - L.95)
> #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])])
> #c(inf,sup)
> lim.fc <- function(x) dev.binom(x, dados=y) - L.95
> ICdev <- c(inf=uniroot(lim.fc, c(0, p.est))$root,
+           sup=uniroot(lim.fc, c(p.est, 1))$root)
> ICdev
      inf      sup
0.5283461 0.8686755
```

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)
```

```

      inf      sup
0.5283461 0.8686755

```

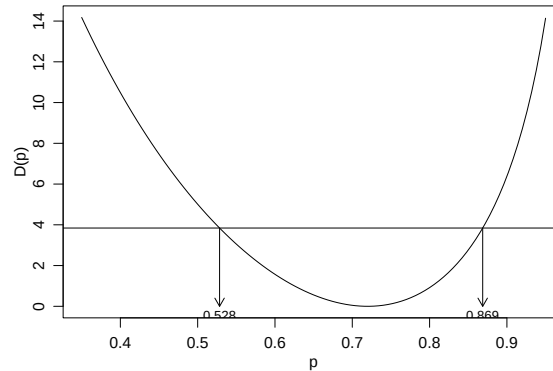


Figura 39: Função deviance para o parâmetro p da distribuição Bernoulli.

```

+   ic <- p.est + qnorm(c(alfa/2, 1-(alfa/2))) * se.p.est
+ }
+ if(tipo == "deviance"){
+   lim.fc <- function(y, ...) dev.binom(y, ...) - qchisq(nivel, df=1)
+   inf <- ifelse(identical(p.est, 0), 0, uniroot(lim.fc, c(0, p.est), n=n, x=x)$root)
+   sup <- ifelse(identical(p.est, 1), 1, uniroot(lim.fc, c(p.est, 1), n=n, x=x)$root)
+   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
0.5283461 0.8686755

```

(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(3214)
> cobertura.binom(n=length(y), p=0.8, nsim=1000)
[1] 0.897
> set.seed(3214)
> cobertura.binom(n=length(y), p=0.8, nsim=1000, tipo = "dev")
[1] 0.96

```

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 aumenta o tamanho da amostra.

17.2 Exercícios

1. Refaça o item (e) do exemplo acima com $n = 10$, $n = 50$ e $n = 200$. Discuta os resultados.
2. Seja $X_1, X_2, \dots, X_n$ uma amostra aleatória da distribuição $U(0, \theta)$. Encontre uma quantidade pivotal e:
 - (a) construa um intervalo de confiança de 90% para θ
 - (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 θ . Verifique se o intervalo cobre o verdadeiro valor de θ .
 - (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 θ . 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 $-2 \sum_i^n \log F(x_i; \theta) \sim \chi_{2n}^2$.

3. Acredita-se que o número de trens atrasados para uma certa estação de trem por dia segue uma distribuição Poisson(θ), 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 θ com base nestes dados
 - (b) o estimador de máxima verossimilhança de θ , 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 ϕ , onde ϕ é a probabilidade de que não haja trens atrasados num particular dia. Construa intervalos de confiança de 95% para ϕ 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 θz_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 $i = 1, \dots, n$:
- (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 θ , $\hat{\theta}$
 - (c) mostre que $\hat{\theta}$ é um estimador não viciado para θ (lembre-se que os valores de z_i são constantes)
 - (d) obtenha um intervalo de aproximadamente 95% de confiança para θ baseado na distribuição assintótica de $\hat{\theta}$

18 Intervalos de confiança baseados na deviance

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

18.1 Média da distribuição normal com variância conhecida

Seja $X_1, \dots, X_n$ a.a. de uma distribuição normal de média θ 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 $\bar{x} \pm \sqrt{c^*/n}$, 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 40 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 pelas 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)))
> L.95 <- qchisq(0.95, df = 1)
> abline(h = L.95, lty=3)
> IC <- 32 + c(-1, 1) * sqrt(L.95/20)
> IC
> arrows(IC, rep(L.95,2), IC, rep(0,2), length=0.1)
```

[1] 31.56174 32.43826

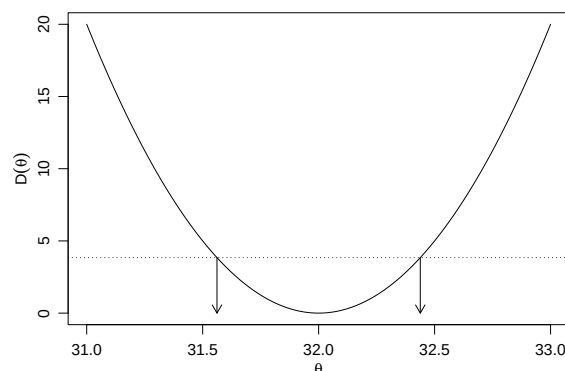


Figura 40: Função deviance para $N(\theta, 1)$ para uma amostra de tamanho 20 e média 32.

Vamos agora examinar o efeito do tamanho da amostra na função. A Figura 41 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.

```
> L.95 <- qchisq(0.95, df = 1)
>
> 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)))
> IC10 <- 32 + c(-1, 1) * sqrt(L.95/10)
> arrows(IC10, rep(L.95/10, 2), IC10, rep(0, 2), length=0.1)
>
> dev20.vals <- dev.norm.v1(thetaN.vals, n=20, xbar=32)
> lines(thetaN.vals, dev20.vals, lty=2)
> IC20 <- 32 + c(-1, 1) * sqrt(L.95/20)
> arrows(IC20, rep(L.95/20, 2), IC20, rep(0, 2), length=0.1, lty=2)
>
> dev50.vals <- dev.norm.v1(thetaN.vals, n=50, xbar=32)
> lines(thetaN.vals, dev50.vals, lwd=2)
> IC50 <- 32 + c(-1, 1) * sqrt(L.95/50)
> arrows(IC50, rep(L.95/50, 2), IC50, rep(0, 2), length=0.1, 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)
```

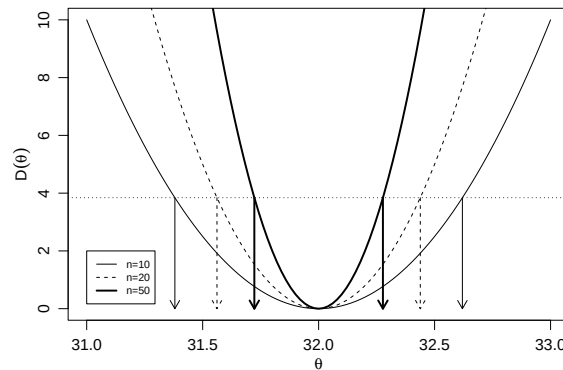


Figura 41: Funções deviance para o parâmetro θ da $N(\theta, 1)$ para amostras de média 32 e tamanhos de amostra $n = 10, 20$ e 50 .

18.2 IC para o parâmetro da distribuição exponencial

Seja $x_1, \dots, x_n$ a.a. de uma distribuição exponencial de parâmetro θ 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 42 e obtida de forma análoga ao exemplo anterior. O estimador de máxima verossimilhança pode ser obtido analiticamente neste exemplo $\hat{\theta} = 1/\bar{x} = 1/10 = 0.1$.

```
> dev.exp <- function(theta, n, xbar){
+   2*n*(log((1/xbar)/theta) + xbar*(theta-(1/xbar)))
+ }
> thetaE.vals <- seq(0.04,0.20, 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)))
```

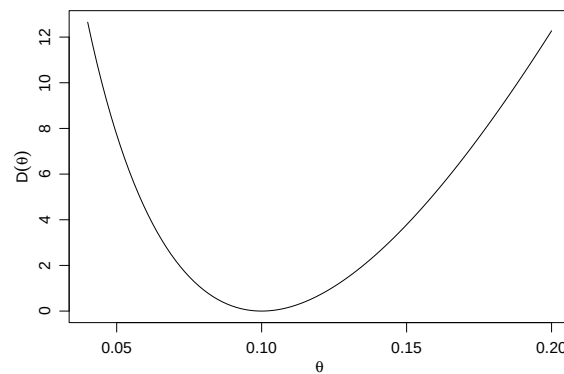


Figura 42: Função deviance da $\text{Exp}(\theta)$ para uma amostra de tamanho 20 e média 10.

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 χ^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, diferentemente do exemplo anterior, esta equação não tem solução analítica vamos examinar a seguir duas possíveis soluções para encontrar os limites do intervalo.

18.2.1 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 θ . A cada um destes valores corresponde um valor de $D(\theta)$. Vamos então localizar os valores de θ 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 43.

```
> plot(thetaE.vals, dev.vals, ty="l", xlab=expression(theta), ylab=expression(D(theta)))
> L.95 <- qchisq(0.95, df = 1)
> abline(h = L.95, lty=3)
```

```

> dif <- abs(dev.vals - L.95)
> #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
> theta.est <- 1/10
> lim.fc <- function(x) dev.exp(x, n=20, xbar=10) - L.95
> ICdev <- c(uniroot(lim.fc, c(0, theta.est))$root,
+           uniroot(lim.fc, c(theta.est, 1))$root)
> arrows(ICdev, rep(L.95,2), ICdev, rep(0,2), len=0.1)
> text(ICdev, 0, round(ICdev, dig=3), pos=1, cex=0.8, offset=0.3)

```

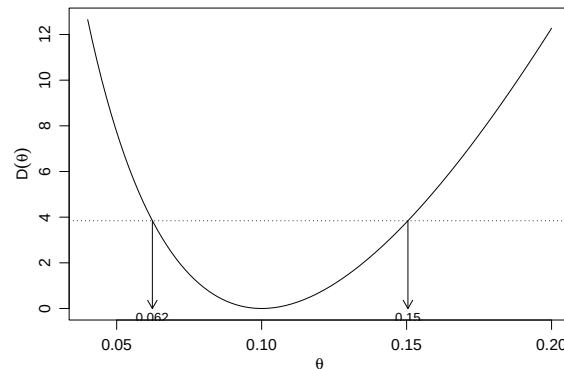


Figura 43: Obtenção gráfica do IC para o parâmetro θ da $\text{Exp}(\theta)$ para uma amostra de tamanho 20 e média 10.

Note que neste código procuramos primeiro o 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. Para isto usamos a função `uniroot()` que fornece raízes unidimensionais de uma função que definimos como a diferença entre a função deviance e o valor de corte definido pela distribuição χ^2 para o nível de significância desejado.

18.2.2 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 44 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)))
> L.95 <- qchisq(0.95, df = 1)
> abline(h = L.95, lty=3)
> ICdevap <- (1/10)*(1 + c(-1,1) * sqrt(L.95/20))
> ICdevap

```

```
[1] 0.05617387 0.14382613
> arrows(ICdevap, rep(L.95,2), ICdevap, rep(0,2), len=0.1)
> text(ICdevap, 0, round(ICdevap, dig=3), pos=1, cex=0.8, offset=0.3)
```

```
[1] 0.05617387 0.14382613
```

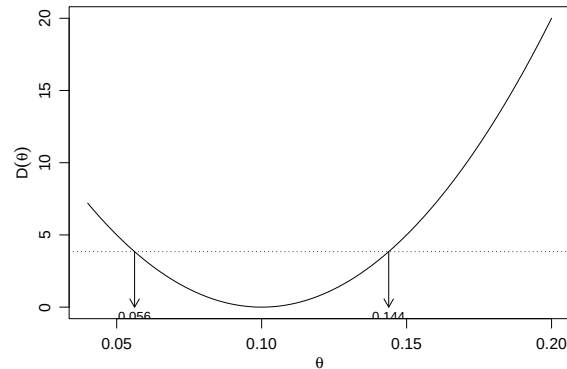


Figura 44: Função deviance obtida pela aproximação quadrática para $\text{Exp}(\theta)$ e uma amostra de tamanho 20 e média 10.

18.3 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 45) 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 = L.95, lty=3)
> arrows(ICdev, rep(L.95,2), ICdev, rep(0,2), len=0.1)
> arrows(ICdevap, rep(L.95,2), ICdevap, rep(0,2), lty=2, len=0.1)
> legend(0.07, 12, c('deviance', 'aproximação quadrática'), lty=c(1,2), cex=0.8)
```

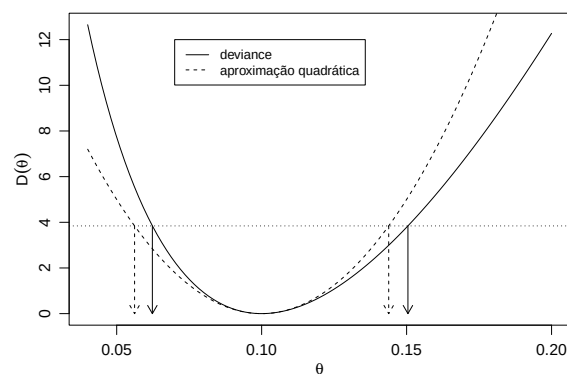


Figura 45: Comparação dos IC's de confiança obtidos pela solução gráfica/numérica (linha sólida) e pela aproximação quadrática (linha tracejada) para o parâmetro θ da $\text{Exp}(\theta)$ para uma amostra de tamanho 20 e média 10.

Vamos agora examinar o efeito do tamanho da amostra na função deviance e sua aproximação quadrática. A Figura 46 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, 0.20, 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)
```

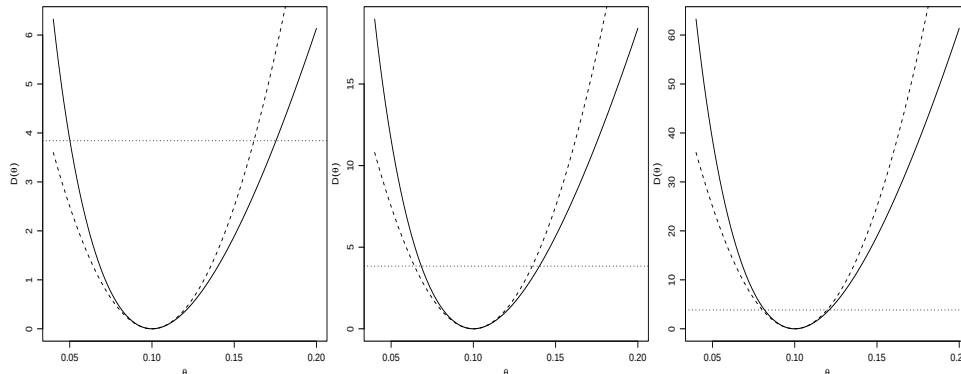


Figura 46: Funções deviance e deviance aproximada para o parâmetro θ da $\text{Exp}(\theta)$ em amostras de média 10 e tamanhos $n = 10$ (esquerda), 30 (centro) e 100 (direita).

18.4 Exercícios

- 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 σ^2 .
 - Obtenha a função deviance para σ^2 e faça o seu gráfico.
 - Obtenha a função deviance para σ e faça o seu gráfico.
 - Obtenha os IC's a 90% de confiança.
- Repita as análises mostradas no exemplo acima da distribuição exponencial mas agora utilizando a seguinte parametrização para a função de densidade:

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

Discuta as diferenças entre os resultados obtidos nas duas parametrizações.

19 Ilustrando propriedades de estimadores

19.1 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 ideia básica é a seguinte:

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.

19.1.1 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 $n = 2, 5, 10, 15, 20, \dots, 1000, 1010, 1020, \dots, 5000$,
4. fazemos os cálculos e produzimos um gráfico como mostrado na Figura 47 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)
```

19.2 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 28](#): se $Y_1, Y_2, \dots, Y_n \sim N(\mu, \sigma^2)$ então $\bar{y} \sim N(\mu, \sigma^2/n)$. Resultados como estes podem ser ilustrados computacionalmente como visto na [Sessão 28](#).

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:

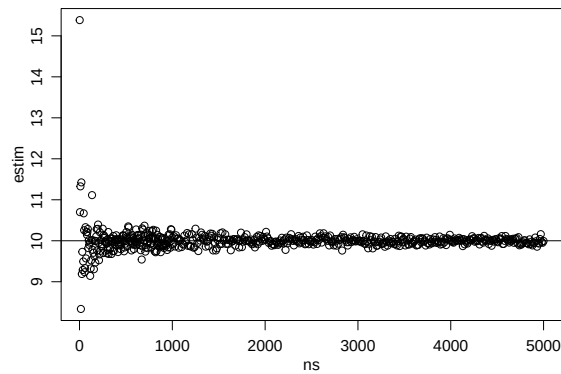


Figura 47: Médias de amostras de diferentes tamanhos.

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] 3.071737
> VarT <- var(Tvals)
> VarT
[1] 1.121181
```

Nestes comandos primeiro obtemos 1000 amostras de tamanho 20 que armazenamos em uma matriz de dimensão 20×1000 , 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 $\hat{E}[T] \approx 3.07$ e $\hat{\text{Var}}[T] \approx 1.12$.

Se você 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 mesmas amostras teríamos que usar a mesma semente para geração de números aleatórios.

19.3 Não-tendenciosidade

Fica como exercício.

19.4 Variância mínima

Fica como exercício.

19.5 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 (19.2) 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 λ 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 $S^2 = \sum_{i=1}^n (X_i - \bar{X})^2 / (n - 1)$ e $\hat{\sigma}^2 = \sum_{i=1}^n (X_i - \bar{X})^2 / n$ do parâmetro de variância σ^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 μ de uma distribuição $N(\mu, \sigma^2)$, onde $X_{[1]}$ e $X_{[n]}$ são os valores mínimo e máximo da amostra, respectivamente.

20 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 ideia 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

20.1 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 duas 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:

$$H_0 : \sigma_A^2 = \sigma_B^2 \quad \text{versus} \quad H_a : \sigma_A^2 \neq \sigma_B^2$$

que é equivalente à

$$H_0 : \frac{\sigma_A^2}{\sigma_B^2} = 1 \quad \text{versus} \quad H_a : \frac{\sigma_A^2}{\sigma_B^2} \neq 1$$

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
```

20.1.1 Fazendo as contas passo a passo

Vamos calcular a estatística de teste. Como temos o computador a disposição não precisamos 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.

20.1.2 Escrevendo uma função

Esta fica por sua conta!

Escreva a sua própria função para testar hipóteses sobre variâncias de duas distribuições normais.

20.1.3 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
```

Nestes argumentos vemos que a função recebe dois vetores 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.

20.2 Exercícios

Os exercícios a seguir foram retirados do livro de Bussab & Morettin (2003).

Note que nos exercícios abaixo nem sempre você poderá usar funções de teste do R porque em alguns casos os dados brutos não estão disponíveis. Nestes casos você deverá fazer os cálculos usando o R como calculadora.

1. Uma máquina automática de encher pacotes de café enche-os segundo uma distribuição normal, com média μ e variância $400g^2$. O valor de μ pode ser fixado num mostrador situado numa posição um pouco inacessível dessa máquina. A máquina foi regulada para $\mu = 500g$. Desejamos, de meia em meia hora, colher uma amostra de 16 pacotes e verificar se a produção está sob controle, isto é, se $\mu = 500g$ ou não. Se uma dessas amostras apresentasse uma média $\bar{x} = 492g$, você 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, e 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?

21 Intervalos de confiança e testes de hipótese

Nesta sessão vamos verificar como utilizar o R para obter intervalos de confiança e testar hipóteses sobre parâmetros de interesse.

21.1 Média de uma distribuição normal com variância desconhecida

Considere resolver o seguinte problema:

Exemplo 1 O tempo de reação de um novo medicamento pode ser considerado como tendo distribuição Normal e deseja-se fazer inferência sobre a média que é desconhecida obtendo um intervalo de confiança. Vinte pacientes foram sorteados e tiveram seu tempo de reação anotado. Os dados foram os seguintes (em minutos):

2.9 3.4 3.5 4.1 4.6 4.7 4.5 3.8 5.3 4.9
4.8 5.7 5.8 5.0 3.4 5.9 6.3 4.6 5.5 6.2

Neste primeiro exemplo, para fins didáticos, vamos mostrar duas possíveis soluções:

1. fazendo as contas passo a passo, utilizando o R como uma calculadora
2. usando uma função já existente no R.

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, 5.0, 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 é dado por:

$$\left(\bar{x} - t_{\alpha/2} \sqrt{\frac{S^2}{n}} \quad , \quad \bar{x} + t_{1-\alpha/2} \sqrt{\frac{S^2}{n}} \right)$$

Vamos agora obter a resposta de duas formas diferentes.

21.1.1 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] 0.9920789
```

Com isto podemos montar o intervalo utilizando os quantis da distribuição t .

```
> 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
```

21.1.2 Usando a função `t.test`

Mostramos a solução 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 já disponíveis para procedimentos estatísticos usuais.

Para este exemplo específico a função `t.test()` pode ser utilizada como vemos no resultado do comando a seguir que coincide com os obtidos anteriormente.

```
> t.test(tempo)

One Sample t-test

data:  tempo
t = 21.305, 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
```

O resultado da função mostra a estimativa obtida da média (4.745), o intervalo de confiança a 95% (4.2788429, 5.2111571) e testa a igualdade de média a zero ($p\text{-value} = 1.0064249 \times 10^{-14}$), em um teste bilateral.

Os valores definidos no IC e teste de hipótese acima são *defaults* que podem ser modificados. Por exemplo, para obter um IC a 99%, em um teste unilateral para $H_1 : \mu > 3$ fazemos:

```
> t.test(tempo, alt = "greater", mu = 3, conf.level = 0.99)

One Sample t-test

data:  tempo
t = 7.835, df = 19, p-value = 1.14e-07
alternative hypothesis: true mean is greater than 3
99 percent confidence interval:
 4.179408      Inf
sample estimates:
mean of x
 4.745
```

□

21.2 Teste χ^2 de independência

Quando estudamos a relação entre duas variáveis qualitativas em geral fazemos uma tabela com o resultado do cruzamento destas variáveis. Em geral existe interesse em verificar se as variáveis estão associadas e para isto calcula-se uma medida de associação tal como o χ^2 , coeficiente de contingência C , ou similar. O passo seguinte é testar se existe evidência que a associação é significativa. Uma possível forma de fazer isto é utilizando o teste χ^2 .

Exemplo 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
```

```
, , Sex = Male
```

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

```
, , Sex = Female
```

	Eye			
Hair	Brown	Blue	Hazel	Green
Black	36	9	5	2
Brown	66	34	29	14
Red	16	7	7	7
Blond	4	64	5	8

Para saber mais sobre estes dados veja `help(HairEyeColor)` Note que estes dados já vem “resumidos” na forma de uma tabela de frequências tri-dimensional, com cada uma das dimensões correspondendo a um dos atributos - cor dos cabelos, olhos e sexo.

Para ilustrar aqui o teste χ^2 vamos verificar se existe associação entre 2 atributos: cor dos olhos e cabelos entre os indivíduos do sexo masculino. Portanto as hipóteses são:

$$H_0 : \text{ não existe associação}$$

$$H_a : \text{ existe associação}$$

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	53	50	25	15
Red	10	10	7	7
Blond	3	30	5	8

```
> chisq.test(HairEyeColor[,1])
```

```
Warning in chisq.test(HairEyeColor[, , 1]): Aproximação do qui-quadrado pode estar incorreta
```

```
Pearson's Chi-squared test
```

```
data: HairEyeColor[, , 1]
```

```
X-squared = 41.28, df = 9, p-value = 4.447e-06
```

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.

Há duas possibilidades de contornar este problema: a primeira é agrupar categorias na tabela. Uma outra (e usualmente melhor) possibilidade 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 = 41.28, df = NA, p-value = 0.0004998
```

Note que agora a mensagem de alerta não é mais emitida e que a significância foi confirmada ($p\text{-valor} < 0.05$). Note que se você rodar este exemplo poderá obter um $p\text{-value}$ 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.

□

21.3 Teste χ^2 para aderência à uma certa distribuição

Exemplo Uma certa hipótese genética, se verdadeira deve produzir indivíduos com 4 fenótipos (A, B, C e D) na população seguindo a relação 9:3:3:1. Para verificar se a hipótese genética é plausível foi coletada uma amostra de indivíduos na população e obteve-se o seguinte número de indivíduos para cada fenótipo:

Fenótipo	A	B	C	D
Nº indivíduos	190	50	63	20

Teste a hipótese de que a hipótese genética é plausível com nível de significância de 5%.

Portanto as hipóteses são:

$$\begin{aligned} H_0 : & \text{ segue a distribuição esperada} \\ H_a : & \text{ não segue a distribuição esperada} \end{aligned}$$

O nível de significância foi definido como $\alpha = 5\%$ e a estatística de teste $\chi_c^2 = \sum_i \frac{(o_i - e_i)^2}{e_i}$ tem distribuição $\chi_{(3)}^2$. Os comandos para efetuar este teste no R são:

```
> o <- c(190, 50, 63, 20)
> e <- c(9, 3, 3, 1)/16
> chisq.test(o, p=e)
```

```
Chi-squared test for given probabilities
```

```
data: o
X-squared = 2.3223, df = 3, p-value = 0.5083
```

Portanto a conclusão é que não se rejeita H_0 ao nível de 5%, ou seja, a hipótese genética é plausível.

□

21.4 Teste para o coeficiente de correlação linear de Pearson

Quando temos duas variáveis quantitativas podemos utilizar o coeficiente de correlação linear de Pearson para medir a associação entre as variáveis, desde que a relação entre elas seja linear.

Exemplo 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` para testar se existe associação entre estas duas variáveis.

As hipóteses são:

$$H_0 : \text{ não existe associação } (\rho = 0)$$

$$H_a : \text{ existe associação } (\rho \neq 0)$$

Vamos ainda fixar o nível de significância em 5%. Os comandos para efetuar o teste e os resultados são mostrados a seguir.

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

Portanto o p-valor acima mostra que a correlação encontrada de -0.8676594 difere significativamente de zero, ou seja existe evidência de associação (negativa) entre estas duas variáveis.

OBS: 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. $\square$

21.5 Comparação de duas médias

Quando temos uma variável qualitativa com dois níveis e outra quantitativa o interesse em geral está em comparar as médias da quantitativa para cada grupo da qualitativa. Para isto podemos utilizar o *teste T*. Há diferentes tipos de teste T: para amostras independentes com variâncias iguais ou desiguais, ou para amostras pareadas.

Exemplo Os dados a seguir correspondem 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 1% 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

As hipóteses são:

$$H_0 : \mu_1 = \mu_2$$

$$H_a : \mu_1 \neq \mu_2$$

Vamos ainda fixar o nível de significância em 5%. Abaixo os comandos para efetuar o teste bilateral com variâncias iguais.

```
> 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=TRUE, conf=0.99)

Two Sample t-test

data:  m1 and m2
t = -0.31724, df = 18, p-value = 0.7547
alternative hypothesis: true difference in means is not equal to 0
99 percent confidence interval:
 -3.324208  2.664208
sample estimates:
mean of x mean of y
   3.63      3.96
```

Os resultados mostram que não há evidências para rejeitar a hipótese de igualdade entre as médias ao nível de 1%.

□

21.6 Exercícios

1. Revisite os dados `milsa` visto na aula de estatística descritiva e selecione pares de variáveis adequadas para efetuar:
 - (a) um teste χ^2
 - (b) um teste para o coeficiente de correlação
 - (c) um teste t
2. 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()`

3. 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.

22 Transformação de dados

Transformação de dados é uma das possíveis formas de contornar 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*.

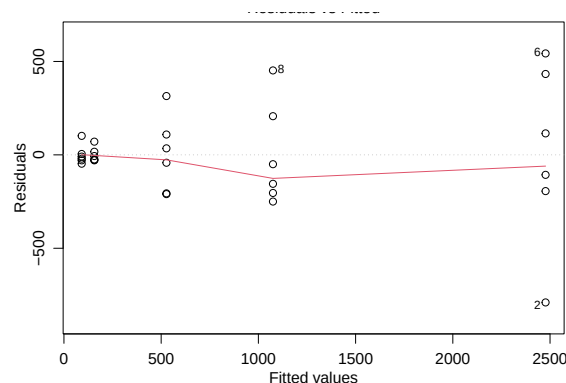
```
> y <- scan()
1: 2370
2: 1687
3: 2592
...
30: 44
31:
Read 30 items

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

'data.frame':      30 obs. of  2 variables:
 $ trat: Factor w/ 5 levels "1","2","3","4",...: 1 1 1 1 1 1 2 2 2 2 ...
 $ resp: num  2370 1687 2592 2283 2910 ...
```

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

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



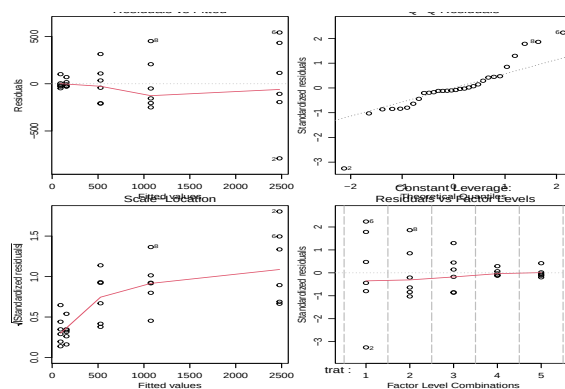
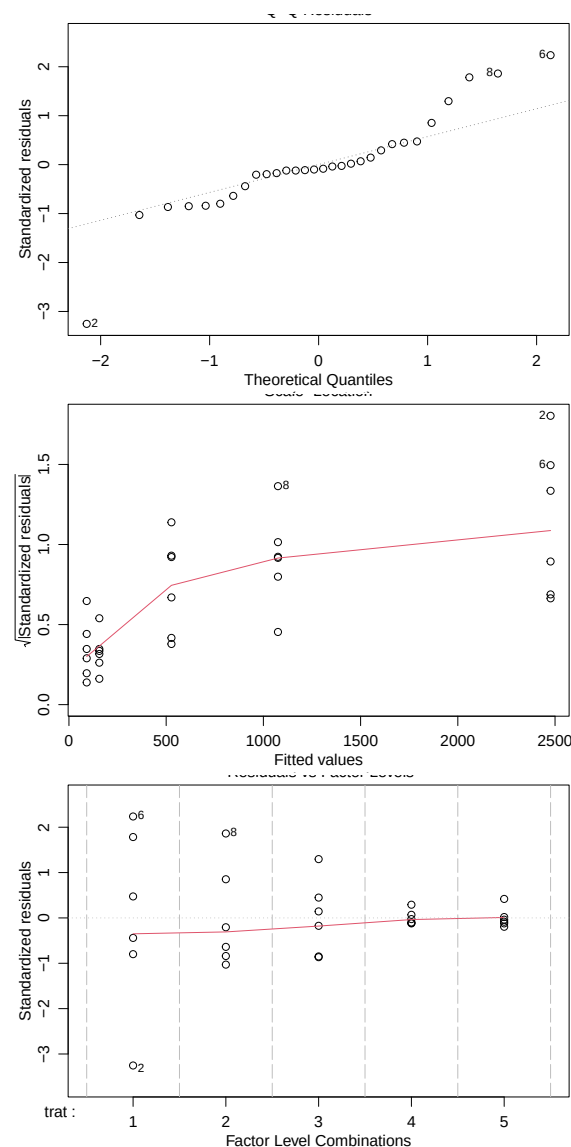


Figura 48: Gráficos de diagnóstico para dados originais



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 mensagem 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

Carregando pacotes exigidos: MASS

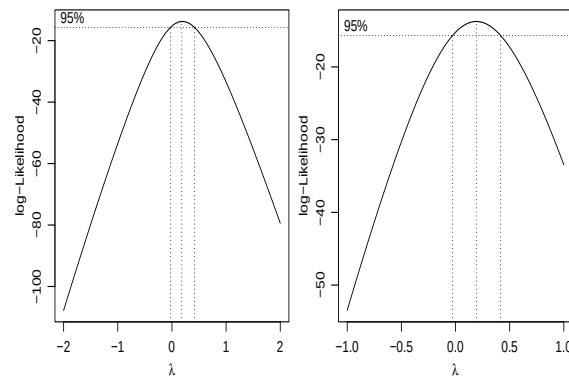


Figura 49: Perfis de verossimilhança para o parâmetro λ da transformação Box-Cox

```
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 = 0.89608, 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 λ é um parâmetro a ser estimado dos dados. Se $\lambda = 0$ a equação acima se reduz a

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

onde $\log$ é o logaritmo neperiano. Uma vez obtido o valor de λ 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 λ . 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 fazemos um novo gráfico com um *zoom* na região de interesse.

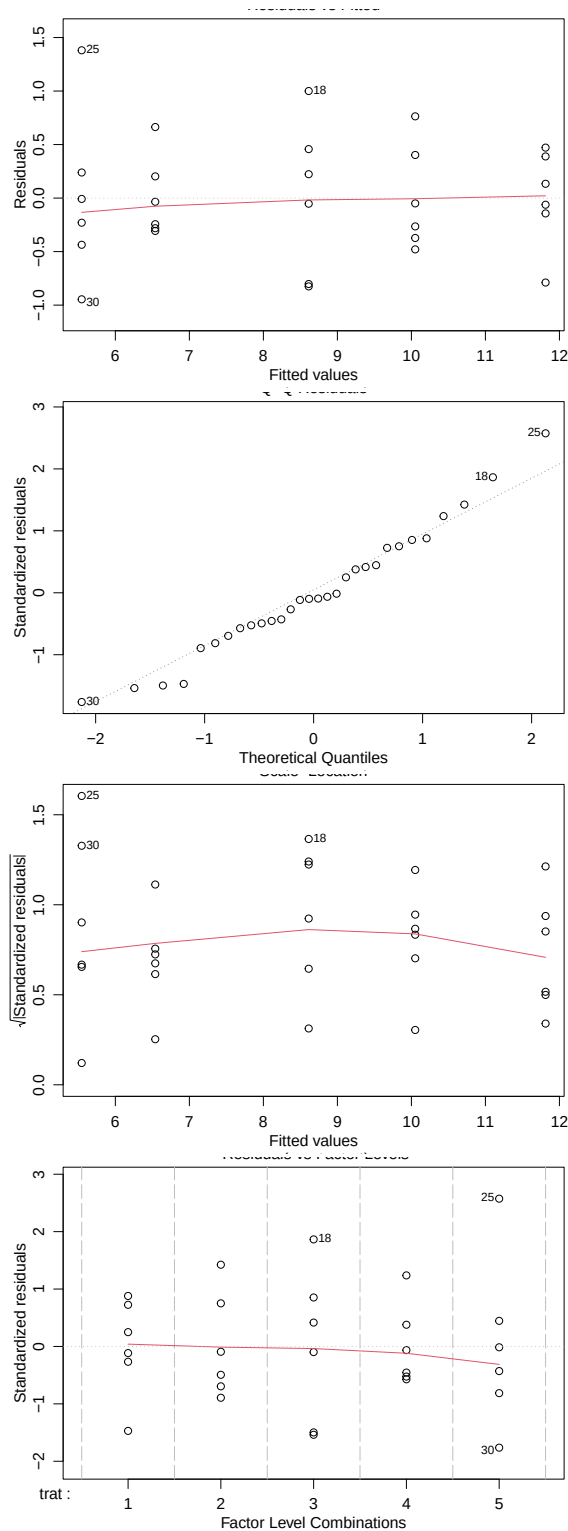
```
> require(MASS)
> boxcox(resp ~ trat, data=tr, plotit=T)
> boxcox(resp ~ trat, data=tr, lam=seq(-1, 1, 1/10))
```

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 têm 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 λ e que o valor 0 está contido neste intervalo. Isto indica que podemos utilizar a transformação logarítmica dos dados e os resultados serão bem próximos dos obtidos com a transformação previamente adotada.

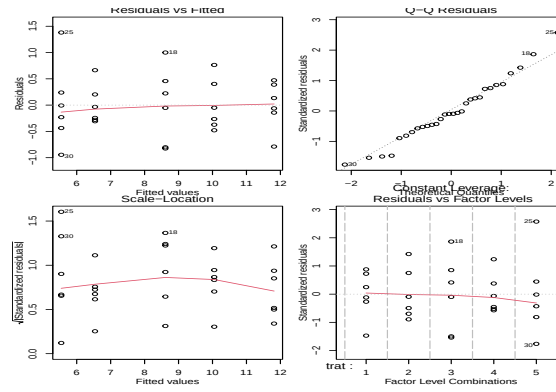
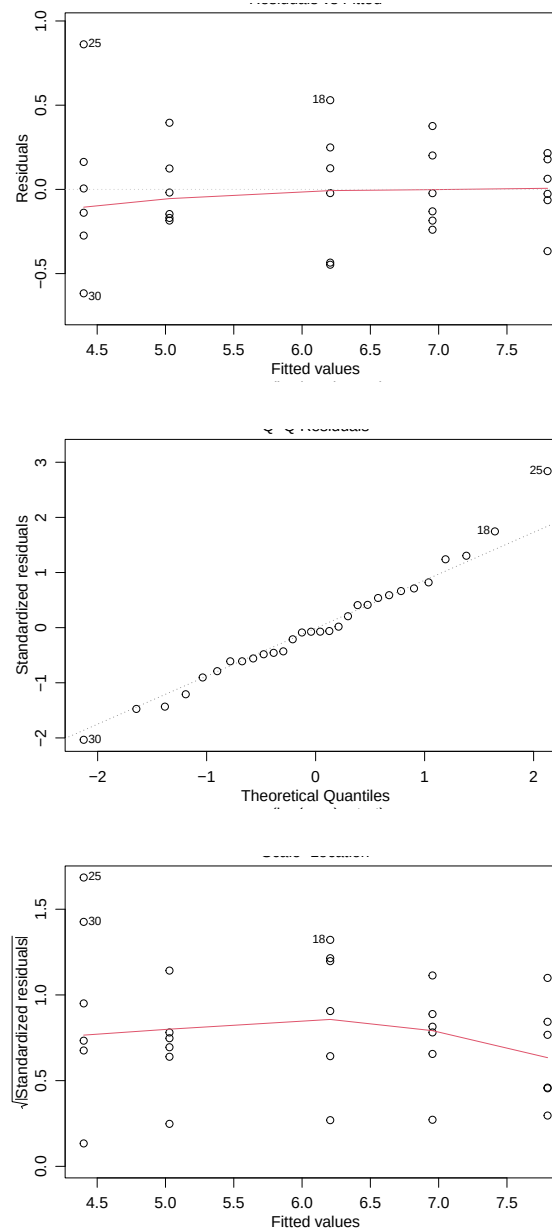


Figura 50: Gráficos de diagnóstico para dados transformados

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



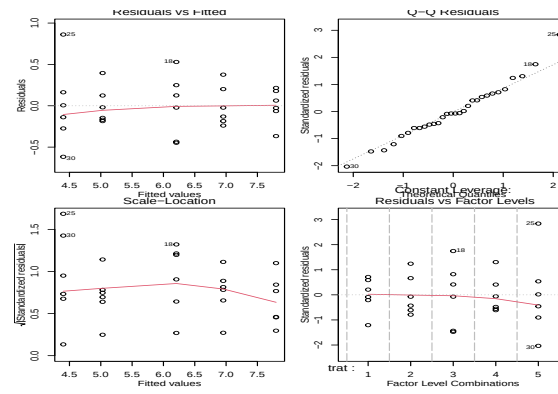
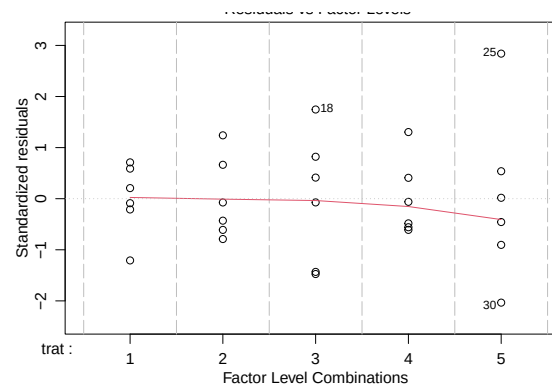


Figura 51: Gráficos de diagnóstico para dados com transformação logarítmica



```
Carregando pacotes exigidos: mgcv
Carregando pacotes exigidos: nlme
This is mgcv 1.9-4. For overview type '?mgcv'.
```

23 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 R que representam simbolicamente relação entre variáveis e/ou objetos de dados. Fórmulas 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 de automóvel. Para carregar os dados 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"
```

23.1 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 *lattice* são disponibilizados pelo pacote **lattice**, no qual as fórmulas são ainda mais importantes e largamente utilizadas.

A Figura 52 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*: 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 53 mostra agora um exemplo onde o gráfico de rendimento explicado pelo peso é feito para cada número de cilindros separadamente. Neste caso a fórmula 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)
```

23.2 Fórmulas em funções

Assim como no caso de gráficos, algumas funções de análise de dados também aceitam fórmulas em seus argumentos. Considere o exemplo do teste-*t* para comparação de duas amostras na comparação do rendimento de veículos com câmbio 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.

```
> 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:
 -10.84837 -3.64151
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 between group 0 and group 1 is not equal to 0
95 percent confidence interval:
 -10.84837 -3.64151
sample estimates:
mean in group 0 mean in group 1
 17.14737      24.39231
```

Portanto em *mpg ~ 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
```

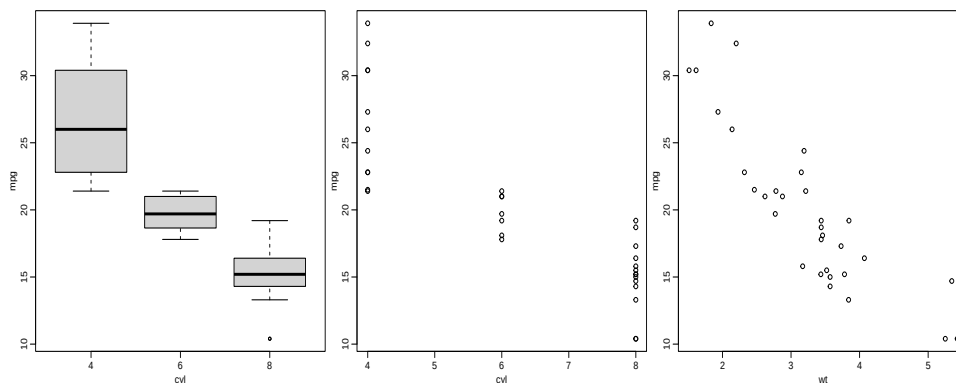


Figura 52: Exemplos de gráficos definidos através de fórmulas.

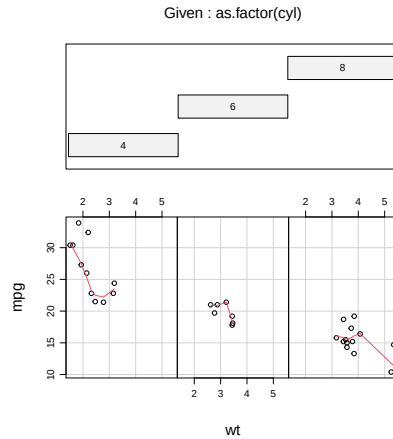


Figura 53: Gráfico obtido através de fórmula com termo condicional.

23.3 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 fórmula 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 ideias e vemos alguns comandos aplicados à manipulação de fórmulas. As funções `all.vars()` e `terms()` são particularmente úteis para manipulação de fórmulas e 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>
```

23.4 Especificação de modelos com uma covariável

Entre os diversos usos de fórmulas, o mais importante deles é sem dúvida o fato que fórmulas são utilizadas na declaração de modelos estatísticos. Um aspecto particularmente importante da linguagem S, e 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 ~ 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 ~ wt` lê-se: *mpg é modelado por wt*, através de um modelo linear `lm()`, o que implica no modelo acima. A Figura 54 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, resultante 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 ajuste é mostrado na linha pontilhada. Outras funções estendem 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.

```
> 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))
```

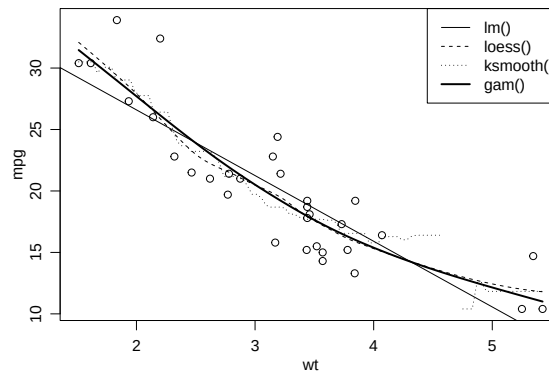


Figura 54: Diferentes modelos ajustados para descrever a relação entre duas variáveis quantitativas.

Nos exemplos acima é interessante notar o uso de `predict()` que é utilizada para prever 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)))
```

23.5 Extensões de modelos com uma covariável

As formulas admitem operadores aritméticos em seus termos. Por exemplo considere a relação entre o rendimento (*mpg*) e a potência (*hp*). A linha sólida no gráfico da esquerda da Figura 55 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 logaritmo 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 55 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.

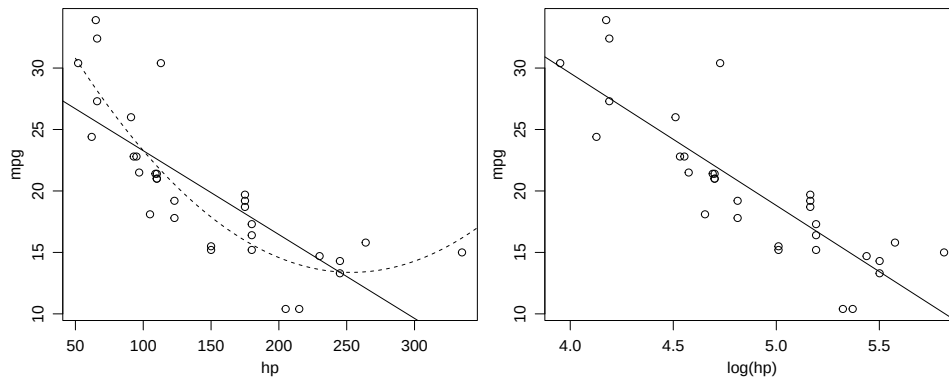


Figura 55: Ilustração do uso de operadores aritméticos e funções polinomiais na especificação de fórmulas.

No exemplo é importante notar que a interpretação dos parâmetros é diferente devido ao fato de 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. Quando se usa as opções *default* a função `poly()` vai sempre contruir polinômios ortogonais. Caso queira-se usar potências usuais deve-se adicionar à chamada desta função o argumento `raw=T`.

```
> 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 56. 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

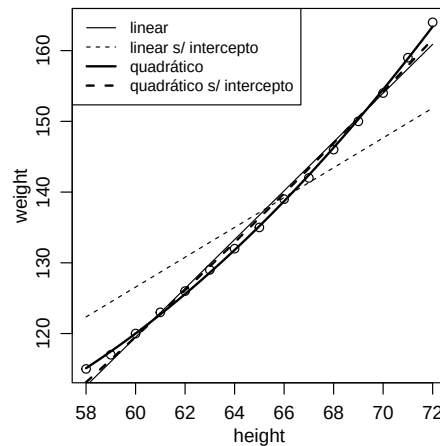


Figura 56: Ajustes de modelos de primeiro e segundo grau, com e sem estimação do intercepto.

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

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)
> 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=
```

23.6 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 5 resume as principais operações possíveis para definir modelos com uma ou duas variáveis e que podem ser estendidas para o caso de mais variáveis.

Esta notação é uma implementação das ideias 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

> 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;
- 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 simbólica* quais e como os termos devem ser incluídos no modelo. Em fórmulas, tais operadores só indicam operações

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 C
$A+B*(C+D)$	$A+B+C+D+B:C+B:D$	todos efeitos principais e interações duplas de B com C e B com D
$A*B*C$	$A+B+C+A:B+A:C+B:C+A:B:C$	todos efeitos principais e interações possíveis
$(A+B+C)^{\sim}3$	$A+B+C+A:B+A:C+B:C+A:B:C$	três covariáveis e interações de ordem 2 e 3 (igual ao anterior)
$(A+B+C)^{\sim}2$	$A+B+C+A:B+A:C+B:C$	três covariáveis e interações de ordem 2
$(A+B+C)^{\sim}3 - A:B:C$	$(A+B+C)^{\sim}2$	três covariáveis e interações de ordem 2
$(A+B+C)^{\sim}2 - A:C$	$A+B+C+A:B+B:C$	três covariáveis e interações de ordem 2, exceto por $A:C$
$A+I(A^{\sim}2)+I(A^{\sim}3)$	$\text{poly}(A,3)$	regressão polinomial cúbica em A (*)
$A+I(A^{\sim}2)+I(A^{\sim}3)$	$\text{poly}(A,3,\text{raw}=\text{TRUE})$	regressão polinomial cúbica em A
$A+I(A^{\sim}2)+B$	$\text{poly}(A,2)+B$	termos lineares em A e B e quadrático em A (*)
$A+I(A^{\sim}2)+B$	$\text{poly}(A,2,\text{raw}=\text{TRUE})+B$	termos lineares em A e B e quadrático em A
$y \sim .$	$A+B+\dots$	inclui como covariáveis todas as variáveis no objeto de dados, exceto a resposta
$y \sim . - A$	$B+\dots$	inclui como covariáveis todas as variáveis no objeto de dados, exceto a resposta e a variável A

aritméticas com os termos envolvidos quando utilizados dentro de $I()$. A função $I()$ garante que a expressão nela contida seja avaliada como uma função aritmética, tal e qual está escrita ("as is").

Na tabela 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 23.5.

23.7 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 (.) 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 mecanismo 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 pela notação de ponto (·).

24 Experimentos com delineamento inteiramente casualizados

Nesta sessão iremos usar o R para analisar um experimento em delineamento inteiramente casualizado com apenas um fator. Tal procedimento é também chamado em alguns textos de "análise da variância de simples entrada" (*one-way anova*). A seguir são apresentados os comandos exemplificando alguns procedimentos usuais para a análise dos dados de um experimento deste tipo que, neste exemplo, envolve um fator com nove níveis (tratamentos). O primeiro passo é ler os dados.

```
> ex01 <- read.table("http://www.leg.ufpr.br/~paulojus/dados/exemplo01.txt", head=T, stringsAsFactors=T)
```

Caso não consiga executar o comando acima diretamente com o endereço *http* utilize um navegador para ir até esta página e copie o arquivo `exemplo01.txt` para o seu diretório de trabalho. Caso o arquivo esteja em outro diretório deve-se colocar o caminho completo deste diretório no argumento de `read.table()` acima. A seguir vamos inspecionar o objeto que armazena os dados e seus componentes. Em particular é importante certificar-se que a variável resposta é do tipo *numeric* e, se os níveis de tratamentos forem qualitativos, a variável indicadora dos tratamentos é do tipo *factor*. Caso isto não ocorra é necessário transformar as variáveis para estes tipos antes de prosseguir com as análises.

```
> head(ex01)
  trat resp
1   t1  385
2   t1  323
3   t1  417
4   t1  370
5   t1  437
6   t1  340

> is.numeric(ex01$resp)
[1] TRUE

> is.factor(ex01$trat)
[1] TRUE
```

Portanto o objeto `ex01` é um `data-frame` com duas variáveis, sendo uma delas um fator (a variável `trat`) e a outra uma variável numérica (`resp`). Vamos iniciar obtendo um rápido resumo dos dados que mostra que este é um experimento "balanceado" com mesmo número de repetições (seis) para cada tratamento. Calculamos também as médias, variâncias e erros padrão das médias para cada tratamento separadamente.

```
> summary(ex01)
      trat      resp
t1      : 6   Min.   :115.0
t2      : 6   1st Qu.:307.5
t3      : 6   Median :377.5
t4      : 6   Mean   :353.5
t5      : 6   3rd Qu.:417.0
t6      : 6   Max.   :474.0
(Other):18

> ex01.nrep <- with(ex01, tapply(resp, trat, length))
> ex01.mds <- with(ex01, tapply(resp, trat, mean))
> ex01.var <- with(ex01, tapply(resp, trat, var))
> ex01.se <- with(ex01, tapply(resp, trat, function(x) sqrt(var(x)/length(x))))
> data.frame(Repet= ex01.nrep, Medias=ex01.mds, Variancias = ex01.var, EP=ex01.se, row.names=paste("tra", 1:nrow(ex01)))
```

Carregando pacotes exigidos: MASS

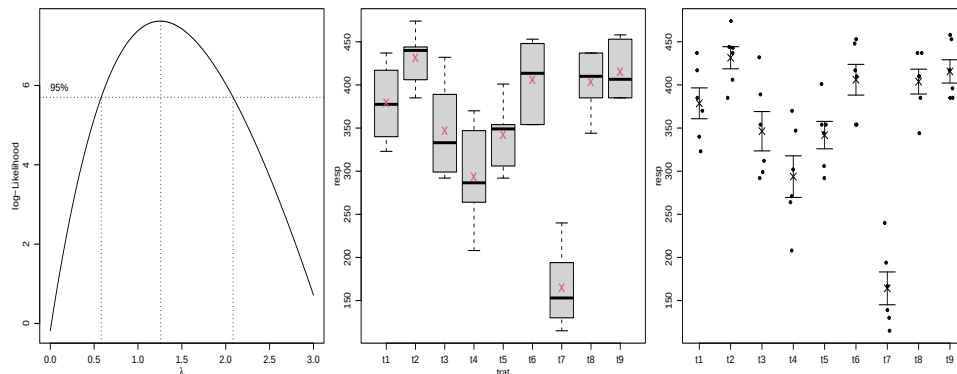


Figura 57: Explorando os dados: perfil de verossimilhança do parâmetro da transformação BoxCox (esquerda), *boxplot* dos dados (centro) e dados com médias e erros padrão para cada tratamento (direita).

	Repet	Medias	Variancias	EP
trat-1	6	378.6667	1916.267	17.87114
trat-2	6	431.5000	987.500	12.82900
trat-3	6	346.3333	3117.867	22.79571
trat-4	6	293.6667	3494.667	24.13389
trat-5	6	341.8333	1513.767	15.88378
trat-6	6	406.0000	1903.600	17.81198
trat-7	6	164.1667	2173.367	19.03228
trat-8	6	403.8333	1242.167	14.38846
trat-9	6	415.6667	1091.067	13.48497

Vamos prosseguir com a análise exploratória com gráficos gerados pelos comandos a seguir e mostrados na Figura 57. O gráfico de esquerda utiliza a função `boxcox()` do pacote **MASS** para verificar a necessidade de transformação dos dados o que neste caso não é necessária visto que o valor um está contido no intervalo definido pelas linhas tracejadas. A transformação Box-Cox é discutida em mais detalhes em uma outra Seção deste material. O gráfico do meio mostra um *boxplot* para os dados de cada tratamento, o que deve ser analisado com cautela lembrando que cada *boxplot* é produzido com apenas seis observações. Optamos aqui por indicar também neste gráfico a média de cada tratamento. O gráfico da direita produzido com `stripchart()` é uma alternativa ao *boxplot* para amostras de tamanho pequeno. Na chamada desta função optamos por alterar valores *default* de alguns argumentos como por exemplo para `method="jitter"` que provoca um pequeno deslocamento horizontal aleatório dos pontos evitando assim sobreposição de pontos com valores coincidentes ou muito próximos. Ainda neste gráfico acrescentamos as médias e barras que somam e subtraem os erros padrões da média para cada tratamento. Na função `arrows()` os quatro argumentos iniciais informam coordenadas para as barras, `code=3` informa que as "setas" devem ser colocadas em ambas extremidades e `angle=90` faz com que a "seta" se torne uma pequena barra horizontal com o tamanho controlado por `length`.

```
> require(MASS); boxcox(resp ~trat, lambda=seq(0,3,l=101), data=ex01)
> plot(ex01)
> points(ex01.mds, pch="x", col=2, cex=1.5)
> with(ex01, stripchart(resp~trat, met="jitter", vert=T, pch=19))
> points(ex01.mds, pch=4, cex=1.5)
> arrows(1:9,ex01.mds+ex01.se,1:9, ex01.mds-ex01.se,angle=90, code=3, length=0.1)
```

É importante notar que as barras simplesmente refletem a variância dos dados dentro de cada tratamento e não são adequadas para detectar diferenças entre tratamentos, o que será discutido mais adiante nesta sessão. Além dos gráficos acima podemos também verificar o pressuposto de homogeneidade de variâncias com o testes de igualdade de variâncias, como por exemplo, o *teste de Bartlett*. Neste caso o teste indica variâncias homogêneas. Caso isto não ocorresse uma possível alternativa seria usar o procedimento descrito na Sessão 24.3.

```
> bartlett.test(resp ~ trat, data=ex01)
```

```
Bartlett test of homogeneity of variances
```

```
data: resp by trat
```

```
Bartlett's K-squared = 3.6738, df = 8, p-value = 0.8853
```

Uma vez concluída a análise exploratória e verificada a adequação de alguns pressupostos o passo seguinte é ajustar o modelo usando `aov()` ou `lm()`. Neste exemplo, por se tratar da análise de um experimento, tipicamente avaliada pelo quadro de análise de variância, optamos por usar `aov()`. Embora `aov()` use `lm()` internamente, os resultados são organizados internamente de forma conveniente para a efetuar a análise de variância.

```
> ex01.mod <- aov(resp ~ trat, data = ex01)
```

```
> ex01.mod
```

```
Call:
```

```
aov(formula = resp ~ trat, data = ex01)
```

```
Terms:
```

	trat	Residuals
Sum of Squares	332918.1	87201.3
Deg. of Freedom	8	45

```
Residual standard error: 44.02053
```

```
Estimated effects may be unbalanced
```

```
> anova(ex01.mod)
```

```
Analysis of Variance Table
```

```
Response: resp
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
trat	8	332918	41615	21.475	5.445e-13
Residuals	45	87201	1938		

Portanto o objeto `ex01.mod` é uma lista que guarda os resultados da análise para o modelo ajustado. Vamos inspecionar este objeto e seus elementos mais detalhadamente ilustrando como usá-lo para obter a análise dos resultados e extrair elementos para a análise de resíduos. A função `names()` mostra os elementos da lista e adicionalmente existem funções que extraem elementos do objeto. Duas tipicamente utilizadas são `coef()` para extrair os coeficientes, `residuals()` para extrair resíduos e `fitted()` para valores ajustados, mas há ainda várias outras como `effects()`, `AIC()`, `logLik()`, `model.tables()`, entre outras.

```
> names(ex01.mod)
```

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

```
> coef(ex01.mod)
```

(Intercept)	tratt2	tratt3	tratt4	tratt5	tratt6	tratt7
378.66667	52.83333	-32.33333	-85.00000	-36.83333	27.33333	-214.50000
	tratt8	tratt9				
	25.16667	37.00000				

```
> model.tables(ex01.mod)
Tables of effects

  trat
trat
  t1    t2    t3    t4    t5    t6    t7    t8    t9
25.15  77.98 -7.19 -59.85 -11.69  52.48 -189.35  50.31  62.15
> model.tables(ex01.mod, type="means")
Tables of means
Grand mean

353.5185

  trat
trat
  t1    t2    t3    t4    t5    t6    t7    t8    t9
378.7 431.5 346.3 293.7 341.8 406.0 164.2 403.8 415.7
```

O resultado de `coef()` vai depender da parametrização adotada e definida pelos contrastes. Os valores *default* e/ou correntes são dados por `options()$contrasts`. Para fatores qualitativos como no caso deste exemplo a parametrização *default* corresponde a "contr.treatment" que assinala o valor da média do primeiro tratamento (primeiro nível do fator) ao primeiro coeficiente. Os demais representam a diferença das médias de cada um dos tratamentos a este tratamento de referência.

Uma outra forma de especificar o modelo para este exemplo é mostrada a seguir com o uso -1 que, para níveis quantitativos corresponde a ajustar um modelo com intercepto igual a zero. No caso de níveis qualitativos como neste exemplo, monta uma matriz do modelo de forma a que cada coeficiente corresponda à média de cada um dos tratamentos. Note que apenas a interpretação dos coeficientes muda e a análise de variância permanece a mesma.

```
> ex01.mod1 <- aov(resp ~ trat - 1, data = ex01)
> coef(ex01.mod1)
  tratt1  tratt2  tratt3  tratt4  tratt5  tratt6  tratt7  tratt8  tratt9
378.6667 431.5000 346.3333 293.6667 341.8333 406.0000 164.1667 403.8333 415.6667
> anova(ex01.mod1)
Analysis of Variance Table

Response: resp
      Df Sum Sq Mean Sq F value    Pr(>F)
trat    9 7081587   786843  406.05 < 2.2e-16
Residuals 45   87201    1938
```

A parametrização para os coeficientes é determinada pela matriz do modelo e é definida pelo argumento `contrasts` de `options()` ou pela função `contrasts()` que mostra ou atribui a matriz de contrastes a ser utilizada. Fatores são definidos como sendo *unordered* (por exemplo níveis qualitativos como no caso da análise vista aqui) ou *ordered*, o que é usado, por exemplo, no caso de níveis quantitativos.

```
> options()$contrasts
      unordered      ordered
"contr.treatment" "contr.poly"
> contrasts(ex01$trat)
  t2 t3 t4 t5 t6 t7 t8 t9
t1  0  0  0  0  0  0  0  0
t2  1  0  0  0  0  0  0  0
t3  0  1  0  0  0  0  0  0
```

```

t4 0 0 1 0 0 0 0 0
t5 0 0 0 1 0 0 0 0
t6 0 0 0 0 1 0 0 0
t7 0 0 0 0 0 1 0 0
t8 0 0 0 0 0 0 1 0
t9 0 0 0 0 0 0 0 1

```

Para definir a parametrização a ser utilizada e definida pelos contrastes, pode-se usar outras opções de contrastes já disponibilizadas pelo R tipicamente usando `options()`. Nos comandos a seguir alteramos a opção para fatores *unordered* para `"contr.sum"`. Os coeficientes obtidos são diferentes dos obtidos anteriormente sendo o primeiro a média geral e os demais uma comparação da média de cada tratamento contra as médias dos demais. Os resultados da análise de variância permanecem inalterado.

```

> options(contrasts=c("contr.sum", "contr.poly"))
> contrasts(ex01$trat)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
t1      1     0     0     0     0     0     0     0
t2      0     1     0     0     0     0     0     0
t3      0     0     1     0     0     0     0     0
t4      0     0     0     1     0     0     0     0
t5      0     0     0     0     1     0     0     0
t6      0     0     0     0     0     1     0     0
t7      0     0     0     0     0     0     1     0
t8      0     0     0     0     0     0     0     1
t9     -1    -1    -1    -1    -1    -1    -1    -1
> coef(lm(resp~trat, data=ex01))
(Intercept)      trat1      trat2      trat3      trat4      trat5      trat6
 353.518519   25.148148   77.981481  -7.185185  -59.851852  -11.685185   52.481481
      trat7      trat8
-189.351852   50.314815

```

Os contrastes já definidos no R são listados e descritos a seguir. Além destes outros pacotes podem ter outras definições de contrastes, como em de `"contr.sdif"` do pacote **MASS**. Estes contrastes terão efeito se o termo `-1` **não** for incluído no modelo pois neste caso os coeficientes são sempre as médias de cada um dos tratamentos, independente da opção de contraste adotada.

- `"contr.treatment"`: já descrito o texto acima, com o primeiro tratamento sendo o de referência.
- `"contr.SAS"`: semelhante ao anterior porém usando o último tratamento como referência.
- `"contr.helmert"`: fornece a média geral como primeiro coeficiente e os demais representam comparações sequenciais dos tratamentos, isto é, segundo contra o primeiro, terceiro contra os dois primeiros, quarto contra os três primeiros e assim por diante.
- `"contr.sum"`: fornece a média geral como primeiro coeficiente e os demais comparam cada um tratamentos com os demais, exceto o último.
- `"contr.poly"`: opção usada para fatores ordenados (*ordered*) como no caso de níveis quantitativos.

Além dos contrastes pré definidos, outros contrastes definidos pelo usuário e atribuídos ao fator em estudo usando a função `contrasts()`. Retornamos a este tópico com um exemplo na Sessão [24.1.3](#).

Retornando à análise do exemplo, vamos ver agora alguns gráficos e ferramentas para avaliar o modelo ajustado. Um método associado a `plot()` produz automaticamente gráficos de

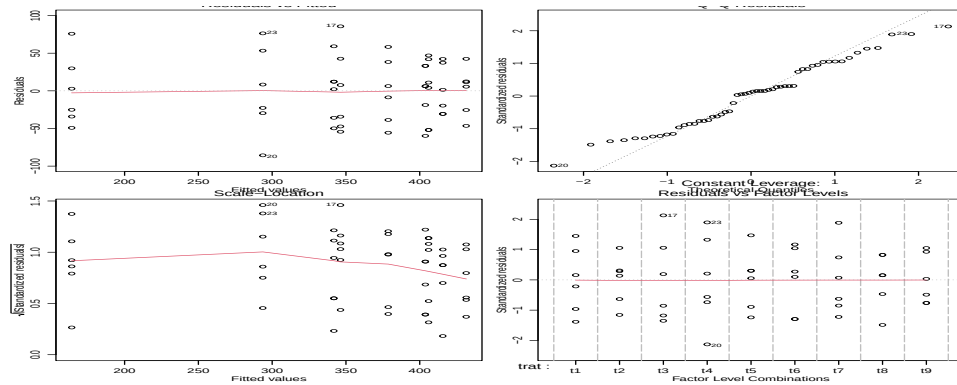


Figura 58: Gráficos para diagnóstico do ajuste do modelo.

resíduos para objetos das classes `lm` e `aov` conforme ilustrado na Figura 58 produzida com o comando `plot(ex01.mod)`.

Além dos gráficos "pré-preparados" pelo R, o usuário pode obter outros que desejar extraindo a informação necessária do objeto que contém o ajuste do modelo. Na Figura 59 mostramos quatro gráficos: resíduos padronizados *versus* valores preditos, boxplot, histograma dos resíduos padronizados e `qqplot()` dos resíduos. Para isto obtemos os resíduos padronizados dividindo os resíduos do modelo pela raiz quadrada da variância do termo de erro.

```
> ex01.res <- resid(ex01.mod)
> respad <- ex01.res/sqrt(sum(ex01.res^2)/ex01.mod$df.res)
> plot(fitted(ex01.mod), respad, xlab="valores ajustados", ylab="resíduos")
> title("Resíduos Padronizados vs \n Valores Preditos")
> boxplot(respad); title("Resíduos Padronizados")
> hist(respad, main="Histograma dos resíduos padronizados")
> qqnorm(ex01.res, ylab="Resíduos", main=NULL); qqline(ex01.res)
> title("Gráfico Normal de \n Probabilidade dos Resíduos")
```

Um teste de normalidade dos resíduos pode ser efetuado como indicado a seguir.

```
> shapiro.test(ex01.res)
```

Shapiro-Wilk normality test

```
data: ex01.res
W = 0.97159, p-value = 0.2263
```

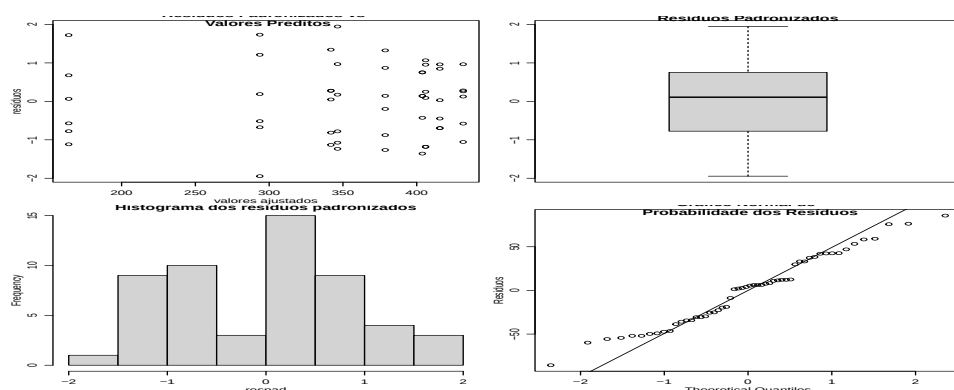


Figura 59: Gráficos de resíduos.

24.1 Comparando tratamentos

Uma das formas possíveis formas de interpretar os resultados no caso de efeito de tratamentos significativos é utilizar algum procedimento de comparações de tratamentos após verificar o resultado da anova, o que justifica o termo às vezes utilizado que descreve tais procedimentos como comparações *post-hoc*.

A questão do uso de comparações de tratamentos é polêmica no meio estatístico e não vamos aqui entrar em tal discussão. Nesta sessão vamos ilustrar três procedimentos deixando a cargo do leitor o julgamento de qual procedimento é mais adequado para o problema em questão. Os procedimentos discutidos a seguir correspondem a três possíveis abordagens ao problema de comparação de mais de duas médias, sendo eles: (i) teste-*t* para comparações duas a duas, (ii) teste de Tukey e (iii) contrastes e contrastes ortogonais. O primeiro caso se desdobra em mais opções uma vez que permite que os valores *p* sejam ou não ajustados, e caso sejam, por diferentes métodos.

Os procedimentos mostrados aqui são implementados em pacotes básicos do R. O pacote **multcomp** disponibiliza uma extensa lista de procedimentos adicionais de comparações múltiplas e alguns procedimentos específicos podem ainda ser encontrados em outros pacotes do R.

24.1.1 Comparações de pares

A função `pairwise.t.test()` calcula todas as possíveis comparações entre dois grupos, podendo ser vista como uma extensão ao teste-*t* para duas amostras, retornando o valor-*p* para cada comparação. A principal diferença é que o nível de significância deve ser corrigido para garantir o nível de significância conjunto para todas as comparações. O argumento `p.adjust.method` da função permite o usuário escolher entre diferentes métodos propostos para ajustar o nível de significância sendo o *default* o procedimento proposto por Holm, que é uma modificação ao ajuste de Bonferroni, que também está disponível através do argumento `p.adj="bonferroni"`. Mais detalhes podem ser encontrados na documentação da função.

```
> with(ex01, pairwise.t.test(resp, trat))
```

Pairwise comparisons using t tests with pooled SD

data: resp and trat

	t1	t2	t3	t4	t5	t6	t7	t8
t2	0.65049	-	-	-	-	-	-	-
t3	1.00000	0.03768	-	-	-	-	-	-
t4	0.03768	6.4e-05	0.65049	-	-	-	-	-
t5	1.00000	0.02345	1.00000	0.83853	-	-	-	-
t6	1.00000	1.00000	0.39692	0.00160	0.28829	-	-	-
t7	2.5e-09	3.8e-12	1.8e-07	0.00019	3.2e-07	8.2e-11	-	-
t8	1.00000	1.00000	0.45676	0.00203	0.33686	1.00000	1.0e-10	-
t9	1.00000	1.00000	0.18109	0.00048	0.11918	1.00000	2.5e-11	1.00000

P value adjustment method: holm

24.1.2 Teste de Tukey

O teste Tukey de comparações múltiplas é implementado na função `TukeyHSD()`. A saída em formato texto do teste de Tukey é mostrada a seguir e `plot(ex01.HSD)` produz o gráfico mostrado na Figura 60. As saídas da função mostram intervalos de confiança para as diferenças entre pares de médias.

```
> ex01.HSD <- TukeyHSD(ex01.mod, ordered=TRUE)
> ex01.HSD
```

```
Tukey multiple comparisons of means
 95% family-wise confidence level
factor levels have been ordered
```

```
Fit: aov(formula = resp ~ trat, data = ex01)
```

```
$trat
      diff      lwr      upr    p adj
t4-t7 129.500000  46.719034 212.28097 0.0002153
t5-t7 177.666667  94.885701 260.44763 0.0000004
t3-t7 182.166667  99.385701 264.94763 0.0000002
t1-t7 214.500000 131.719034 297.28097 0.0000000
t8-t7 239.666667 156.885701 322.44763 0.0000000
t6-t7 241.833333 159.052367 324.61430 0.0000000
t9-t7 251.500000 168.719034 334.28097 0.0000000
t2-t7 267.333333 184.552367 350.11430 0.0000000
t5-t4  48.166667 -34.614299 130.94763 0.6203900
t3-t4  52.666667 -30.114299 135.44763 0.5040619
t1-t4  85.000000   2.219034 167.78097 0.0401018
t8-t4 110.166667  27.385701 192.94763 0.0024139
t6-t4 112.333333  29.552367 195.11430 0.0018566
t9-t4 122.000000  39.219034 204.78097 0.0005599
t2-t4 137.833333  55.052367 220.61430 0.0000730
t3-t5   4.500000 -78.280966  87.28097 1.0000000
t1-t5  36.833333 -45.947633 119.61430 0.8721075
t8-t5  62.000000 -20.780966 144.78097 0.2886707
t6-t5  64.166667 -18.614299 146.94763 0.2479215
t9-t5  73.833333  -8.947633 156.61430 0.1146645
t2-t5  89.666667   6.885701 172.44763 0.0247945
t1-t3  32.333333 -50.447633 115.11430 0.9342210
t8-t3  57.500000 -25.280966 140.28097 0.3855262
t6-t3  59.666667 -23.114299 142.44763 0.3369467
t9-t3  69.333333 -13.447633 152.11430 0.1671352
t2-t3  85.166667   2.385701 167.94763 0.0394343
t8-t1  25.166667 -57.614299 107.94763 0.9849417
t6-t1  27.333333 -55.447633 110.11430 0.9749062
t9-t1  37.000000 -45.780966 119.78097 0.8693183
t2-t1  52.833333 -29.947633 135.61430 0.4998060
t6-t8   2.166667 -80.614299  84.94763 1.0000000
t9-t8 11.833333 -70.947633  94.61430 0.9999286
t2-t8 27.666667 -55.114299 110.44763 0.9730043
t9-t6   9.666667 -73.114299  92.44763 0.9999849
t2-t6 25.500000 -57.280966 108.28097 0.9836416
t2-t9 15.833333 -66.947633  98.61430 0.9993743
```

Visualizações mais convenientes dos resultados podem ser obtidas com operações sobre o objeto resultante, tal como a usualmente adotada de listar as médias em ordem decrescente e indicar com letras as diferenças significativas ou não entre estas médias. Vamos ilustrar aqui uma possível forma de obter tal visualização. Inicialmente vamos obter a DMS (diferença mínima significativa). No caso deste experimento balanceado, isto é, o mesmo número de repetições em cada tratamento, o intervalo de confiança para cada diferença é o mesmo e a DMS é portanto comum e dada por metade da amplitude do intervalo.

```
> dms <- unname(0.5 * diff(ex01.HSD[[1]][1,2:3]))
> dms
[1] 82.78097
```

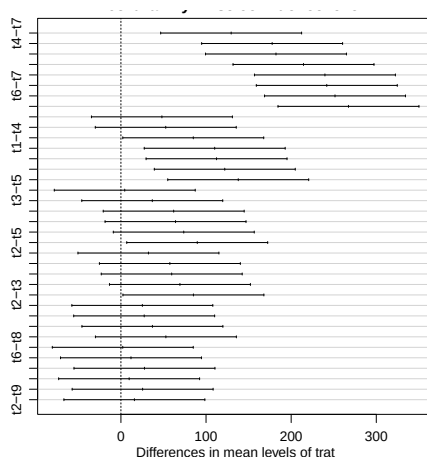


Figura 60: Resultados do teste de Tukey.

O passo seguinte é ordenar as médias de forma decrescente e verificar as diferenças significativas. O código abaixo é uma (mas certamente não a única) maneira de indicar as diferenças significativas usando código de letras usual na literatura.

```
> ex01.mds.ord <- sort(ex01.mds, decreasing = TRUE)
> i <- pos <- letra <- 1
> letras <- character(nlevels(ex01$trat))
> while(i <= nlevels(ex01$trat)){
+   print(letters[letra])
+   ind <- (ex01.mds.ord[i] - (ex01.mds.ord[-(1:i)])) < dms
+   pos.i <- i+sum(ind)
+   if(pos.i > pos){
+     letras.vec <- rep(" ",length(letras))
+     letras.vec[i:pos.i] <- letters[letra]
+     letras <- paste(letras, letras.vec, sep="")
+     pos <- pos.i
+     letra <- letra + 1
+   }
+   i <- i + 1
+ }
```

```
[1] "a"
[1] "b"
[1] "c"
[1] "c"
[1] "c"
[1] "c"
[1] "c"
[1] "d"
[1] "d"
[1] "d"
```

```
> data.frame(medias = ex01.mds.ord, diferencas = letras)
```

	medias	diferencas
t2	431.5000	a
t9	415.6667	ab
t6	406.0000	ab
t8	403.8333	ab
t1	378.6667	ab
t3	346.3333	bc
t5	341.8333	bc
t4	293.6667	c
t7	164.1667	d

Neste caso o procedimento é simples pois para um experimento balanceado e pelo teste de Tukey tem-se apenas um único valor de DMS. O algoritmo deve ser modificado e generalizado para outras situações ou pode-se usar funções de pacotes como `multcompLetters` do pacote `multcompView`.

24.1.3 Contrastes e contrastes ortogonais

Na análise de experimentos pode-se ter interesse em estudar determinadas comparações entre as médias que podem ser especificadas pelo usuário na forma de *contrastes*, que são um caso particular das *funções estimáveis* para o modelo. Vamos iniciar revendo definições.

Seja o modelo linear escrito na forma matricial $Y = X\beta + \epsilon$ onde Y é a variável resposta, X a matriz do modelo, β o vetor de p parâmetros (coeficientes) e ϵ o vetor de erros. Uma combinação linear dos coeficientes da forma $\sum_p \lambda_p \beta_p$ onde $\lambda = [\lambda_1, \dots, \lambda_p]$ é um vetor de constantes é dita uma *função estimável* para o dado modelo se λ pode ser escrita como uma combinação linear das linhas da X . Um *contraste* é um caso especial de função estimável em que a soma das constantes é nula, isto é, pode ser escrito como $\sum_p c_p \beta_p$ onde $\sum_p c_p = 0$.

No que se segue vamos ver como obter estimativas de contrastes de interesse no R, onde *fórmulas* lineares são usadas para definir as matrizes do modelo usadas no ajuste de modelos lineares e lineares generalizados. No caso de fatores (qualitativos) a matriz X do modelo não é definida unicamente para um mesmo experimento, podendo ser escrita de diversas formas alternativas que irão produzir a ajustes equivalentes. Tais formas são definidas pela escolha de contrastes ou funções estimáveis que definirão a interpretação dos coeficientes β do modelo. Portanto, se o interesse é apenas na análise de variância a particular forma adotada é irrelevante. Por outro lado, a escolha deve ser adequada se os coeficientes devem ser interpretados.

Ao ajustar um modelo as estimativas de contrastes podem ser obtidas de duas formas:

- após o ajuste do modelo, a partir de operações matriciais sobre os coeficientes ajustados;
- diretamente no ajuste do modelo, associando ao(s) fatores a estrutura de contrastes desejadas. Desta forma os coeficientes já fornecem estimativas dos contrastes e cálculos adicionais não são necessários.

Vamos discutir aqui algumas ideias iniciais sobre como implementar a segunda forma. Como na análise de contrastes os coeficientes passam a ser diretamente interpretados, passamos a usar `lm()` no ajuste do modelo.

Uma classe especial de contrastes é a de *contrastes ortogonais*. Um conjunto de contrastes ortogonais tem a propriedade de que a soma dos produtos dos coeficientes de qualquer par de contrastes deste conjunto é nula. Contrastes ortogonais são particularmente interessantes pois permitem desdobrar (particionar) a soma de quadrados de tratamentos em parcelas referentes a cada um dos contrastes. Isto permite que cada contraste seja testado diretamente por um teste t (ou o equivalente teste F).

Com nove tratamentos é possível definir oito contrastes ortogonais com cada um deles sendo associado a um dos graus de liberdade dos tratamentos. A definição destes contrastes não é única e deve refletir comparações relevantes para o problema em questão, assegurando-se que a ortogonalidade seja mantida o que garante que a soma das somas de quadrados dos contrastes seja equivalente à soma de quadrados total dos tratamentos. Para obter o desdobramento abordamos a modelagem como um problema de regressão múltipla onde os contrastes definem variáveis quantitativas a serem incluídas no modelo que é ajustado com `lm()`. Neste exemplo vamos considerar o seguinte conjunto de contrastes entre as médias dos tratamentos que são especificados nas linhas de uma matriz como se segue.

C1: t1, t2 e t3 *versus* t4 a t9

C2: t1 versus t2 e t3

C3: t2 versus t3

C4: t4, t5 versus t6, t7, t8, t9

C5: t4 versus t5

C6: t6 e t7 versus t8 e t9

C7: t6 versus t7

C8: t8 versus t9

```
> c1 <- rbind(c(2,2,2,-1,-1,-1,-1,-1,-1),
+             c(2,-1,-1, rep(0,6)),
+             c(0, 1,-1, rep(0,6)),
+             c(rep(0,3), c(2,2,-1,-1,-1,-1)),
+             c(rep(0,3), c(1,-1), rep(0,4)),
+             c(rep(0,5), c(1,1,-1,-1)),
+             c(rep(0,5), c(1,-1, 0,0)),
+             c(rep(0,5), c(0,0, 1,-1)))
> c1
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    2    2    2   -1   -1   -1   -1   -1   -1
[2,]    2   -1   -1    0    0    0    0    0    0
[3,]    0    1   -1    0    0    0    0    0    0
[4,]    0    0    0    2    2   -1   -1   -1   -1
[5,]    0    0    0    1   -1    0    0    0    0
[6,]    0    0    0    0    0    1    1   -1   -1
[7,]    0    0    0    0    0    1   -1    0    0
[8,]    0    0    0    0    0    0    0    1   -1
```

O próximo passo é fazer com que a matriz do modelo seja montada pelo R de forma que os coeficientes reflitam os contrastes desejados. Para isto associamos ao fator que representa os tratamentos (**trat** no exemplo) o *atributo* **contrast** contendo a inversa generalizada obtida por **ginv()** do pacote **MASS**. A análise de variância deste modelo é a mesma obtida anteriormente. entretanto os coeficientes são agora dados pela média geral seguida pelas estimativas de cada um dos oito contrastes definidos que podem ser testadas diretamente pelo teste-*t* usando o comando **summary()**.

```
> c1.ginv <- ginv(c1)
> colnames(c1.ginv) <- paste("contr", 1:8, sep="")
> contrasts(ex01$trat) <- c1.ginv
> mod1 <- lm(resp~trat, data=ex01)
> anova(mod1)
```

Analysis of Variance Table

Response: resp

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
trat	8	332918	41615	21.475	5.445e-13
Residuals	45	87201	1938		

```
> summary(mod1)
```

Call:

```
lm(formula = resp ~ trat, data = ex01)
```

Residuals:

Min	1Q	Median	3Q	Max
-85.67	-33.29	4.75	33.17	85.67

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	353.52	5.99	59.014	< 2e-16
tratcontr1	287.83	76.25	3.775	0.000466
tratcontr2	-20.50	44.02	-0.466	0.643682
tratcontr3	85.17	25.41	3.351	0.001638
tratcontr4	-118.67	62.25	-1.906	0.063029
tratcontr5	-48.17	25.41	-1.895	0.064503
tratcontr6	-249.33	35.94	-6.937	1.26e-08
tratcontr7	241.83	25.41	9.515	2.41e-12
tratcontr8	-11.83	25.41	-0.466	0.643748

Residual standard error: 44.02 on 45 degrees of freedom

Multiple R-squared: 0.7924, Adjusted R-squared: 0.7555

F-statistic: 21.48 on 8 and 45 DF, p-value: 5.445e-13

Nos comandos a seguir visualizamos os mesmos resultados de uma forma alternativa, usando `model.matrix()` para montar a matriz de covariáveis da forma desejada, onde excluimos o intercepto (primeira coluna) e, para visualização adequada dos resultados, trocamos os nomes das colunas. A este data-frame adicionamos os dados e ajustamos o modelo de regressão com `lm()`. A função `anova()` sobre o modelo ajustado exibe a soma de quadrados decomposta entre os contrastes agora testados pelo teste F que é equivalente ao teste- t mostrado acima pois cada contraste possui um grau de liberdade. Note que a soma delas corresponde a soma de quadrados de tratamentos mostrada no ajuste inicial do modelo e os coeficientes são os mesmos.

```
> ex01co <- data.frame(model.matrix(resp~trat, ex01)[,-1])
> names(ex01co) <- paste("Contraste", 1:8)
> ex01co$resp <- ex01$resp
> mod2 <- lm(resp ~ ., data=ex01co)
> av2 <- anova(mod2); av2
```

Analysis of Variance Table

Response: resp

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
`Contraste 1`	1	27616	27616	14.2512	0.000466
`Contraste 2`	1	420	420	0.2169	0.643682
`Contraste 3`	1	21760	21760	11.2292	0.001638
`Contraste 4`	1	7041	7041	3.6334	0.063029
`Contraste 5`	1	6960	6960	3.5917	0.064503
`Contraste 6`	1	93251	93251	48.1217	1.264e-08
`Contraste 7`	1	175450	175450	90.5405	2.409e-12
`Contraste 8`	1	420	420	0.2168	0.643748
Residuals	45	87201	1938		

```
> sum(av2$Sum[1:8])
```

[1] 332918.1

```
> coef(mod2)
```

	`Contraste 1`	`Contraste 2`	`Contraste 3`	`Contraste 4`	`Contraste 5`
(Intercept)	353.51852	287.83333	-20.50000	85.16667	-118.66667
`Contraste 6`	-249.33333	241.83333	-11.83333		

Os coeficientes retornados equivalem a aplicar os contrastes desejados sobre as médias dos tratamentos. Pode-se ainda visualizar os contrastes assinalados ao fator `trat` através da inversa generalizada.

```
> drop(c1 %**% ex01.mds)
[1] 287.83333 -20.50000 85.16667 -118.66667 -48.16667 -249.33333 241.83333
[8] -11.83333
> fractions(contrasts(ex01$trat))
      contr1 contr2 contr3 contr4 contr5 contr6 contr7 contr8
t1    1/9    1/3     0     0     0     0     0     0
t2    1/9   -1/6    1/2     0     0     0     0     0
t3    1/9   -1/6   -1/2     0     0     0     0     0
t4  -1/18     0     0    1/6    1/2     0     0     0
t5  -1/18     0     0    1/6   -1/2     0     0     0
t6  -1/18     0     0  -1/12     0    1/4    1/2     0
t7  -1/18     0     0  -1/12     0    1/4   -1/2     0
t8  -1/18     0     0  -1/12     0   -1/4     0    1/2
t9  -1/18     0     0  -1/12     0   -1/4     0   -1/2
```

Uma forma alternativa talvez mais direta e conveniente de obter a decomposição da soma de quadrados de tratamentos entre os contrastes é com o uso de `aov()` e `summary()` utilizando o argumento `split`

```
> mod3 <- aov(resp ~ trat, data = ex01)
> summary(mod3, split=list(trat=1:8))
      Df Sum Sq Mean Sq F value    Pr(>F)
trat    8 332918   41615   21.475 5.44e-13
  trat: C1  1  27616   27616   14.251 0.000466
  trat: C2  1    420     420    0.217 0.643682
  trat: C3  1  21760   21760   11.229 0.001638
  trat: C4  1   7041    7041    3.633 0.063029
  trat: C5  1   6960    6960    3.592 0.064503
  trat: C6  1  93251   93251   48.122 1.26e-08
  trat: C7  1 175450  175450   90.541 2.41e-12
  trat: C8  1    420     420    0.217 0.643748
Residuals 45  87201    1938
```

Nota: A atribuição do atributo `contrast` ao fator **não** terá efeito sobre a construção da matriz do modelo caso o termo de intercepto esteja retirado na definição do modelo, por exemplo, se o modelo acima fosse definido por `resp trat - 1`.

Para cancelar a atribuição dos contrastes a um fator e retornar à definida por `option()` basta fazer atribuir a valor `NULL`.

```
> contrasts(ex01$trat) <- NULL
```

Interpretando os contrastes: a interpretação dos contrastes vai depender dos valores dos coeficientes definidos na matriz dos contrastes (matriz `c1` no exemplo). Por exemplo suponha que desejamos obter contrastes de forma não só a obter testes de significância mas também obter estimativas dos contrastes com interpretação desejada. No exemplo, para comparar médias μ_i (ou efeitos t_i) dos tratamentos com efeitos estimados dos contrastes na mesma escala das médias dos tratamentos poderíamos definir `c1` da forma a seguir. Todos os resultados e testes de significância (estatísticas de testes e p-valores) permanecem inalterados e apenas os valores dos efeitos estimados são diferentes

```
> c1 <- rbind(c(1/3,1/3,1/3,-1/6,-1/6,-1/6,-1/6,-1/6),
+             c(1,-0.5,-0.5, rep(0,6)),
+             c(0, 1,-1, rep(0,6)),
+             c(rep(0,3), c(0.5,0.5,-0.25,-0.25,-0.25,-0.25)),
+             c(rep(0,3), c(1,-1), rep(0,4)),
+             c(rep(0,5), c(0.5,0.5,-0.5,-0.5)),
```

```

+           c(rep(0,5), c(1,-1, 0,0)),
+           c(rep(0,5), c(0,0, 1,-1)))
> fractions(c1)
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]  1/3  1/3  1/3 -1/6 -1/6 -1/6 -1/6 -1/6 -1/6
[2,]   1 -1/2 -1/2   0   0   0   0   0   0
[3,]   0   1  -1   0   0   0   0   0   0
[4,]   0   0   0  1/2  1/2 -1/4 -1/4 -1/4 -1/4
[5,]   0   0   0   1  -1   0   0   0   0
[6,]   0   0   0   0   0  1/2  1/2 -1/2 -1/2
[7,]   0   0   0   0   0   1  -1   0   0
[8,]   0   0   0   0   0   0   0   1  -1

> c1.ginv <- ginv(c1)
> colnames(c1.ginv) <- rownames(c1) <- paste("contr", 1:8, sep="")
> contrasts(ex01$trat) <- c1.ginv
> mod4 <- lm(resp~trat, data=ex01)
> summary(mod4)

```

Call:

```
lm(formula = resp ~ trat, data = ex01)
```

Residuals:

Min	1Q	Median	3Q	Max
-85.67	-33.29	4.75	33.17	85.67

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	353.52	5.99	59.014	< 2e-16
tratcontr1	47.97	12.71	3.775	0.000466
tratcontr2	-10.25	22.01	-0.466	0.643682
tratcontr3	85.17	25.41	3.351	0.001638
tratcontr4	-29.67	15.56	-1.906	0.063029
tratcontr5	-48.17	25.41	-1.895	0.064503
tratcontr6	-124.67	17.97	-6.937	1.26e-08
tratcontr7	241.83	25.41	9.515	2.41e-12
tratcontr8	-11.83	25.41	-0.466	0.643748

Residual standard error: 44.02 on 45 degrees of freedom

Multiple R-squared: 0.7924, Adjusted R-squared: 0.7555

F-statistic: 21.48 on 8 and 45 DF, p-value: 5.445e-13

```

> coef(mod4)
(Intercept) tratcontr1 tratcontr2 tratcontr3 tratcontr4 tratcontr5 tratcontr6
 353.51852   47.97222  -10.25000   85.16667  -29.66667  -48.16667  -124.66667
tratcontr7 tratcontr8
241.83333  -11.83333

```

Definindo contrastes com funções do pacote gmodels Este pacote oferece várias funções auxiliares que facilitam o uso e a apresentação dos resultados em formatos por vezes convenientes. No exemplo a seguir vamos repetir a obtenção dos resultados para os contrastes definidos na matriz *c1* conforme definida anteriormente.

```

> require(gmodels)
Carregando pacotes exigidos: gmodels
> mod5 <- aov(resp~trat, data=ex01, contrast=list(trat=make.contrasts(c1)))
> summary(mod5, split=list(trat=1:8))

```

```

      Df Sum Sq Mean Sq F value    Pr(>F)
trat    8 332918   41615   21.475 5.44e-13
  trat: C1  1  27616   27616   14.251 0.000466
  trat: C2  1    420    420    0.217 0.643682
  trat: C3  1  21760   21760   11.229 0.001638
  trat: C4  1   7041   7041    3.633 0.063029
  trat: C5  1   6960   6960    3.592 0.064503
  trat: C6  1  93251   93251  48.122 1.26e-08
  trat: C7  1 175450  175450  90.541 2.41e-12
  trat: C8  1    420    420    0.217 0.643748
Residuals 45  87201   1938
> fit.contrast(mod5, "trat", c1)
      Estimate Std. Error    t value    Pr(>|t|)
tratcontr1  47.97222   12.70763   3.7750713 4.660336e-04
tratcontr2 -10.25000   22.01027  -0.4656918 6.436823e-01
tratcontr3  85.16667   25.41527   3.3510042 1.638352e-03
tratcontr4 -29.66667   15.56361  -1.9061560 6.302888e-02
tratcontr5 -48.16667   25.41527  -1.8951863 6.450255e-02
tratcontr6 -124.66667  17.97131  -6.9369836 1.263698e-08
tratcontr7  241.83333   25.41527   9.5152781 2.408674e-12
tratcontr8 -11.83333   25.41527  -0.4655994 6.437479e-01

```

Experimentos desbalanceados: finalmente vale ressaltar que o exemplo acima tratou de um experimento *balanceado*, isto é, com o mesmo número de repetições para cada tratamento e no caso de desbalanceamento ajustes são necessários na definição dos contrastes.

24.2 Recursos adicionais para comparações múltiplas

Na sessão anterior discutimos a comparação *post-hoc* de tratamentos utilizando funções como *pairwise.t.test()* e *TukeyHSD* implementadas no conjunto de pacotes básicos do R.

Outros procedimentos são implementados em pacotes contribuídos do R. Entre estes encontra-se os pacotes **multcomp** e **multcompView** que implementam diversos outros procedimentos e gráficos para visualizações dos resultados. Vale notar que estes pacotes devem ser instalados com a opção *dependencies=TRUE* para garantir plena funcionalidade pois suas funções dependem de diversos outros pacotes.

```

> install.packages("multcompView", dep=TRUE)
> require(multcomp)
> require(multcompView)

```

```

Carregando pacotes exigidos: multcomp
Carregando pacotes exigidos: mvtnorm
Carregando pacotes exigidos: survival
Carregando pacotes exigidos: TH.data

```

```
Anexando pacote: 'TH.data'
```

```
O seguinte objeto é mascarado por 'package:MASS':
```

```
geyser
```

```
Carregando pacotes exigidos: multcompView
```

Para ilustrar o uso desta pacote vamos efetuar novamente o teste de Tukey visto acima porém agora utilizando cálculos e gráficos gerados por funções destes pacotes, cujos resultados, embora iguais, são apresentados em forma diferente do visto anteriormente. A indicação de

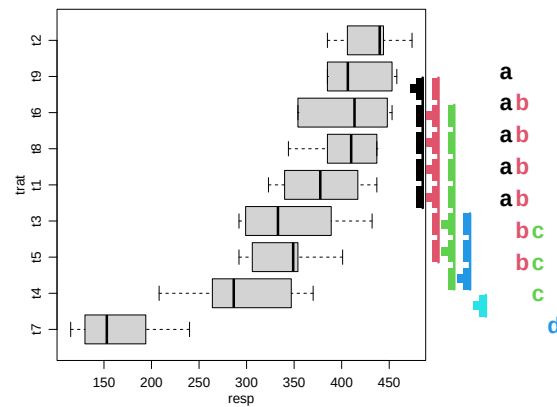


Figura 61: Visualização dos resultados do teste de Tukey com função do pacote `multicompView`

letras para diferenças entre pares de tratamentos mostrada a seguir requer que `TukeyHSD` seja invocada sem a ordenação dos tratamentos e uma representação visual é dada na Figura 61.

```
> multcompLetters(TukeyHSD(ex01.mod)$trat[,4])
  t2  t3  t4  t5  t6  t7  t8  t9  t1
"a" "bc" "b" "bc" "ac" "d" "ac" "ac" "ac"
> multcompBoxplot(resp~trat,data=ex01,compFn="TukeyHSD",decreasing=FALSE)
```

24.3 Análise para variâncias não homogêneas

No caso de variâncias não homogêneas em experimentos inteiramente casualizados a função `oneway.test()` pode ser utilizada nas análises. Uma outra alternativa é a análise não paramétrica da Kruskal-Wallis implementada por `kruskal.test()`.

25 Análise de experimentos em esquema fatorial

O experimento fatorial descrito em Banzato & Kronka (1989) comparou o crescimento de mudas de eucalipto considerando como fatores diferentes tipos de recipientes e espécies.

25.1 Lendo os dados

Vamos considerar agora que os dados já estejam digitados em um arquivo texto. [Clique aqui](#) para ver e/ou copiar o arquivo com conjunto de dados para o seu diretório de trabalho. A seguir deve-se ler ("importar") os dados para R com o comando `read.table()`: Se você não tiver restrições de acesso (firewall, etc) pode importar o arquivo diretamente fornecendo a URL (endereço *web*) do arquivo.

```
> ex04 <- read.table("http://www.leg.ufpr.br/~paulojus/dados/exemplo04.txt", head=T, stringsAsFactors=T)
> # ex04 <- read.table("DEST/aulasR/dados/exemplo04.txt", head=T)
```

Antes de começar as análise vamos usar alguns comandos para 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 e verificar se cada variável possui o "tipo" correto.

```
> head(ex04)
  rec esp resp
1  r1  e1 26.2
2  r1  e1 26.0
3  r1  e1 25.0
4  r1  e1 25.4
5  r1  e2 24.8
6  r1  e2 24.6
> dim(ex04)
[1] 24  3
> names(ex04)
[1] "rec" "esp" "resp"
> is.factor(ex04$rec)
[1] TRUE
> is.factor(ex04$esp)
[1] TRUE
> is.factor(ex04$resp)
[1] FALSE
> is.numeric(ex04$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 no caso deste experimento é a variável resposta.

25.2 Análise exploratória

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

```
> 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

```

Vamos explorar um pouco mais os dados calculando as médias para cada nível de cada fator e também para as combinações dos níveis dos fatores.

```

> ex04.mr <- with(ex04,tapply(resp, rec, mean))
> ex04.mr
      r1      r2      r3
25.4875 22.7250 20.6875
> ex04.me <- with(ex04,tapply(resp, esp, mean))
> ex04.me
      e1      e2
23.85833 22.07500
> ex04.m <- with(ex04,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

```

As combinações dos níveis dos fatores podem ainda ser obtidas com `interaction()` que produz uma saída na forma de um vetor com nomes que combinam os níveis dos fatores envolvidos.

```

> with(ex04, tapply(resp, interaction(rec,esp), mean))
r1.e1 r2.e1 r3.e1 r1.e2 r2.e2 r3.e2
25.650 25.875 20.050 25.325 19.575 21.325

```

Nos comandos mostrados anteriormente a função `mean()` pode ser substituída por qualquer outra função de interesse seja pré definida ou definida pelo usuário. Nos exemplos a seguir ilustramos ambas situações onde são obtidas as medianas com a função pré-definida `median` e o número de observações acima de 22 para cada combinação dos fatores, com uma função definida por nós.

```

> with(ex04, tapply(resp, interaction(rec,esp), median))
r1.e1 r2.e1 r3.e1 r1.e2 r2.e2 r3.e2
25.70 26.00 19.30 25.00 19.30 21.35
> with(ex04, tapply(resp, interaction(rec,esp), function(x) sum(x > 22)))
r1.e1 r2.e1 r3.e1 r1.e2 r2.e2 r3.e2
      4      4      1      4      0      1

```

As médias para os fatores e suas combinações também poderiam ser obtidas com o comando `model.tables()` o que será mostrado mais adiante. Entretanto neste estágio de análise descritiva, preferimos o mecanismo mais geral de `tapply()` que permite o cálculo de 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 62.

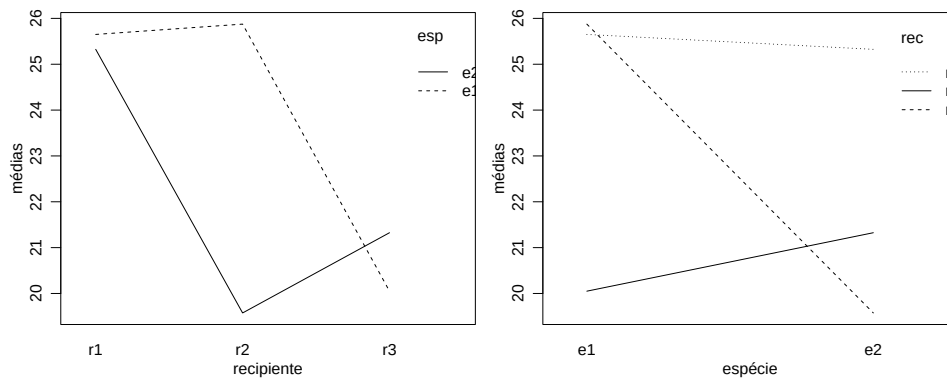


Figura 62: Gráficos de interação entre os fatores.

```
> with(ex04, interaction.plot(rec, esp, resp, ylab="médias", xlab="recipiente", xpd=F))
> with(ex04, interaction.plot(esp, rec, resp, ylab="médias", xlab="espécie", xpd=F))
```

Pode-se usar o R para obter outros tipos de gráficos de acordo com o interesse de quem está analisando os dados. Os comandos a seguir ilustram alguns outros tipos de gráficos que podemos produzir. Na figura 63 são mostrados gráficos semelhantes aos mostrados anteriormente, porém com pontos referentes às observações o que permite visualizar a variabilidade em cada grupo definido pelas combinações dos níveis dos fatores.

```
> with(ex04, plot.default(rec, resp, ty="n", ylab="Resposta", xlab="recipiente", ))
> with(ex04, points(rec[esp=="e1"], resp[esp=="e1"], col=1))
> points(ex04.m[,1], pch="x", col=1, cex=1.5)
> with(ex04, points(rec[esp=="e2"], resp[esp=="e2"], col=2))
> points(ex04.m[,2], pch="x", col=2, cex=1.5)
> with(ex04, interaction.plot(rec, esp, resp, xpd=F, lty=1, col=1:2))
>
> with(ex04, plot.default(esp, resp, ty="n", ylab="Resposta", xlab="espécie"))
> with(ex04, points(esp[rec=="r1"], resp[rec=="r1"], col=1))
> points(ex04.m[,1], pch="x", col=1, cex=1.5)
> with(ex04, points(esp[rec=="r2"], resp[rec=="r2"], col=2))
> points(ex04.m[,2], pch="x", col=2, cex=1.5)
> with(ex04, points(esp[rec=="r3"], resp[rec=="r3"], col=3))
> points(ex04.m[,3], pch="x", col=3, cex=1.5)
> with(ex04, interaction.plot(esp, rec, resp, xpd=F, lty=1, col=1:3))
```

Além destes gráficos produzidos pelo *sistema básico de gráficos do R* pode-se usar comandos fornecidos pelo pacote **lattice** que implementam um poderoso conjunto alternativo de gráficos mas não serão abordados aqui.

25.3 Análise de variância

Seguindo o modelo adequado, a análise de variância para este experimento inteiramente casualizado em esquema fatorial pode ser obtida com as funções `aov()` ("analysis of variance") ou `lm()` ("linear model"). A primeira usa a segunda internamente visto que o modelo é linear, porém ajusta os resultados em um formato em geral mais adequado para análise de experimentos. Nestas funções os modelos são declarados por "fórmulas". A seguir vemos duas fórmulas que especificam o mesmo modelo.

```
> ex04.av <- aov(resp ~ rec + esp + rec:esp, data=ex04)
> ex04.av <- aov(resp ~ rec * esp, data=ex04)
> summary(ex04.av)
```

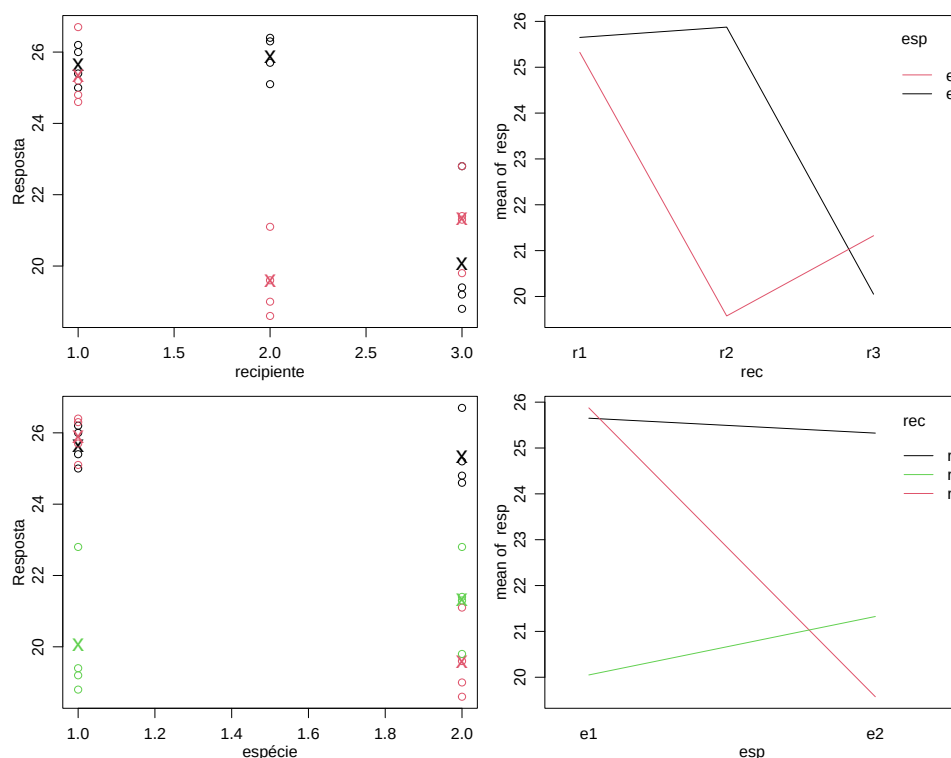


Figura 63: Gráficos de pontos examinando a interação entre os fatores.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
rec	2	92.86	46.43	36.20	4.92e-07 ***
esp	1	19.08	19.08	14.88	0.00116 **
rec:esp	2	63.76	31.88	24.85	6.64e-06 ***
Residuals	18	23.09	1.28		

 Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Isto significa que dentro de uma fórmula no R, o símbolo ":" define o termo de interação e "*" indica a inclusão dos efeitos principais e interações. A análise acima mostra que neste caso o efeito de interação é significativo, confirmando o que foi indicado nos gráficos exploratórios do efeito 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 da classe `aov` produz tabelas dos efeitos (se `type="effects"`) ou das médias (se `type="means"`) definidas pelo modelo. O resultado mostra a média geral, médias de cada nível dos fatores e das combinações dos níveis dos fatores. No resultado está incluído também o número de dados que gerou cada média.

```
> model.tables(ex04.av, type="means")
```

```
Tables of means
Grand mean
```

```
22.96667
```

```
rec
```

```
rec
```

```
  r1    r2    r3
```

```
25.487 22.725 20.687
```

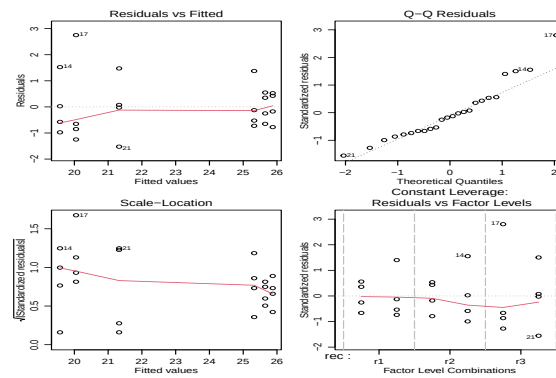


Figura 64: Gráficos de resíduos produzidos para objetos da classe `lm`.

```
esp
esp
  e1    e2
23.858 22.075

rec:esp
  esp
rec e1    e2
  r1 25.650 25.325
  r2 25.875 19.575
  r3 20.050 21.325
```

Mas isto ainda não é tudo que se pode extrair da análise! O objeto `ex04.av` possui vários elementos que guardam diversas outras informações sobre o ajuste do modelo e que podem ser exploradas subsequentemente por métodos de funções para as classes `aov` e `lm` ou por requisições definidas pelo usuário. A seguir veremos alguns exemplos.

```
> 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"
```

A chamada `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, a análise de resíduos.

25.4 Análise de resíduos

A análise de resíduos é útil para verificar os pressupostos do modelo. Usando o mecanismos de classes, o comando `plot(ex04.av)` aplicado sobre o objeto que contém o ajuste do modelo produz uma figura com quatro gráficos básicos para análise dos resíduos conforme mostrado na Figura 64.

Os gráficos permitem uma análise dos resíduos que auxilia no julgamento da adequação do modelo. Evidentemente não é necessário limitar-se aos gráficos produzidos automaticamente pelo R – você pode criar os seus próprios gráficos. Nestes gráficos pode-se usar outras variáveis,

tipos de gráficos, mudar texto de eixos e títulos, etc, etc, etc. Os comandos a seguir mostram como obter os gráficos *boxplot* dos resíduos para os níveis de cada um dos fatores como mostrado na Figura 65.

```
> residuos <- resid(ex04.av)
> plot(ex04$rec, residuos)
> title("Resíduos vs Recipientes")
> plot(ex04$esp, residuos)
> title("Resíduos vs Espécies")
```

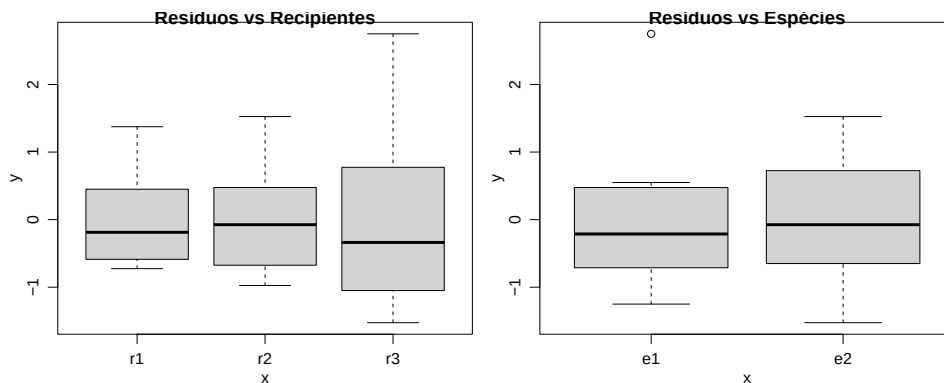


Figura 65: Gráficos de resíduos para cada um dos fatores.

A Figura 66 mostra outros gráficos definidos pelo usuário: resíduos *versus* valores preditos, um *boxplot* dos resíduos padronizados, e um *qqplot* dos resíduos do modelo. Note que o objeto que contém o ajuste foi utilizado para extrair resíduos, valores preditos e a estimativa *s2* da variância dos resíduos.

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

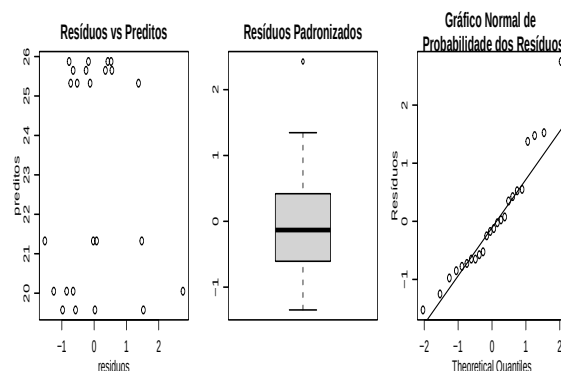


Figura 66: Alguns gráficos de resíduos definidos pelo usuário.

Além da análise gráfica de resíduos há alguns testes já programados em funções. Como exemplo vejamos o teste de Shapiro-Wilks para testar a normalidade dos resíduos.

```
> shapiro.test(residuos)
```

```
Shapiro-Wilk normality test
```

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

25.5 Desdobrando interações

Quando a interação entre os fatores é significativa pode-se adotar como estratégia de análise o desdobramento dos graus de liberdade de um fator dentro de cada nível do outro fator. Uma forma de obter tal desdobramento 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, data=ex04)
> summary(ex04.avr, split=list("rec:esp"=list(r1=1, r2=2, r3=3)))
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
rec	2	92.86	46.43	36.195	4.92e-07 ***
rec:esp	3	82.84	27.61	21.527	3.51e-06 ***
rec:esp: r1	1	0.21	0.21	0.165	0.690
rec:esp: r2	1	79.38	79.38	61.881	3.11e-07 ***
rec:esp: r3	1	3.25	3.25	2.535	0.129
Residuals	18	23.09	1.28		

```
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
> ex04.ave <- aov(resp ~ esp/rec, data=ex04)
> 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	19.08	19.08	14.88	0.00116 **
esp:rec	4	156.62	39.16	30.52	8.44e-08 ***
esp:rec: e1	2	87.12	43.56	33.96	7.78e-07 ***
esp:rec: e2	2	69.50	34.75	27.09	3.73e-06 ***
Residuals	18	23.09	1.28		

```
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Neste ponto vale uma explicação mais detalhada sobre como obter os desdobramentos da interação usando o argumento `split`, em particular como definir os elementos da lista que, no caso de `ex04.ave` foram `e1=c(1,3)` e `e2=c(2,4)`. Inicialmente vamos extrair usando `effects()` os efeitos ajustados pelo modelo.

```
> effects(ex04.ave)[1:6]
```

(Intercept)	espe2	espe1:recr2	espe2:recr2	espe1:recr3	espe2:recr3
-112.513229	-4.368257	4.939804	6.123724	-7.919596	-5.656854

Os efeitos que temos interesse no desdobramento são os da interação, que são: `espe1:recr2`, `espe2:recr2`, `espe1:recr3` e `espe2:recr3`. Portanto temos que localizar no vetor de efeitos as posições desses efeitos de interação que são: 1º : `espe1:recr2`, 2º : `espe2:recr2`, 3º : `espe1:recr3` e 4º : `espe2:recr3`. Isto mostra que a posição dos efeitos que contém a espécie1 (`e1`) são 1 e 3, e espécie2 (`e2`) são 2 e 4 o que define os valores nos vetores indicados no argumento `split`.

25.6 Teste de Tukey para comparações múltiplas

Há vários testes de comparações múltiplas disponíveis na literatura, e muitos deles são implementados nos pacotes básicos do R e/ou em pacotes contribuídos. Por exemplo, o pacote **multcomp** é inteiramente dedicado à implementação de diversos procedimentos de comparações múltiplas no R. Além disto, procedimentos que não estejam implementados podem ser calculados utilizando os recursos usuais do R utilizando os objetos com o ajuste dos modelos. Como ilustração mostramos a seguir duas formas de obter resultados para o *Teste de Tukey*, a primeira usando uma implementação já disponível com a função `TukeyHSD()` e uma segunda fazendo os cálculos necessários passo a passo com operações básicas do R. Para a função já disponível simplesmente digitamos os comandos a seguir e os resultados podem ser mostrados na forma texto ou gráfica como na Figura 67 que é produzida com o comando `plot(ex04.tk1)`.

```
> ex04.tk1 <- TukeyHSD(ex04.av)
> ex04.tk1
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = resp ~ rec * esp, data = ex04)

$rec
      diff      lwr      upr    p adj
r2-r1 -2.7625 -4.207787 -1.3172128 0.0003395
r3-r1 -4.8000 -6.245287 -3.3547128 0.0000003
r3-r2 -2.0375 -3.482787 -0.5922128 0.0055472

$esp
      diff      lwr      upr    p adj
e2-e1 -1.783333 -2.75476 -0.8119067 0.0011553

$`rec:esp`
      diff      lwr      upr    p adj
r2:e1-r1:e1  0.225 -2.3201851  2.770185 0.9997185
r3:e1-r1:e1 -5.600 -8.1451851 -3.054815 0.0000204
r1:e2-r1:e1 -0.325 -2.8701851  2.220185 0.9983324
r2:e2-r1:e1 -6.075 -8.6201851 -3.529815 0.0000068
r3:e2-r1:e1 -4.325 -6.8701851 -1.779815 0.0004825
r3:e1-r2:e1 -5.825 -8.3701851 -3.279815 0.0000120
r1:e2-r2:e1 -0.550 -3.0951851  1.995185 0.9811892
r2:e2-r2:e1 -6.300 -8.8451851 -3.754815 0.0000041
r3:e2-r2:e1 -4.550 -7.0951851 -2.004815 0.0002705
r1:e2-r3:e1  5.275  2.7298149  7.820185 0.0000444
r2:e2-r3:e1 -0.475 -3.0201851  2.070185 0.9902110
r3:e2-r3:e1  1.275 -1.2701851  3.820185 0.6135909
r2:e2-r1:e2 -5.750 -8.2951851 -3.204815 0.0000143
r3:e2-r1:e2 -4.000 -6.5451851 -1.454815 0.0011258
r3:e2-r2:e2  1.750 -0.7951851  4.295185 0.2914242
```

Esta saída fornece resultados detalhados de várias comparações possíveis entre os níveis dos fatores e suas combinações. Entretanto, neste caso, nem todos os resultados mostrados 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 níveis dos fatores e o gráfico com tais resultados pode ser obtido com `plot(ex04.tk2)`.

```
> ex04.tk2 <- TukeyHSD(ex04.ave, "esp:rec")
> ex04.tk2
```

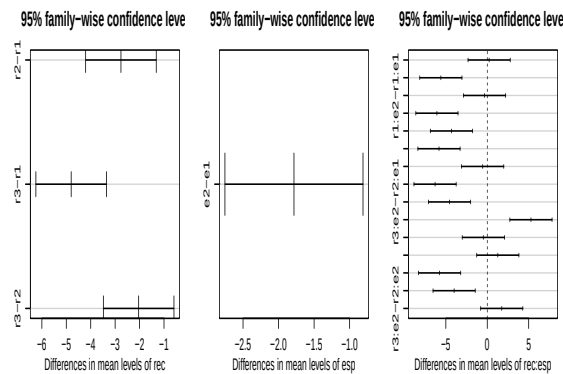


Figura 67: Visualização dos resultados do teste de Tukey de comparações múltiplas.

Tukey multiple comparisons of means
95% family-wise confidence level

```
Fit: aov(formula = resp ~ esp/rec, data = ex04)
```

```
$`esp:rec`
      diff      lwr      upr    p adj
e2:r1-e1:r1 -0.325 -2.8701851  2.220185 0.9983324
e1:r2-e1:r1  0.225 -2.3201851  2.770185 0.9997185
e2:r2-e1:r1 -6.075 -8.6201851 -3.529815 0.0000068
e1:r3-e1:r1 -5.600 -8.1451851 -3.054815 0.0000204
e2:r3-e1:r1 -4.325 -6.8701851 -1.779815 0.0004825
e1:r2-e2:r1  0.550 -1.9951851  3.095185 0.9811892
e2:r2-e2:r1 -5.750 -8.2951851 -3.204815 0.0000143
e1:r3-e2:r1 -5.275 -7.8201851 -2.729815 0.0000444
e2:r3-e2:r1 -4.000 -6.5451851 -1.454815 0.0011258
e2:r2-e1:r2 -6.300 -8.8451851 -3.754815 0.0000041
e1:r3-e1:r2 -5.825 -8.3701851 -3.279815 0.0000120
e2:r3-e1:r2 -4.550 -7.0951851 -2.004815 0.0002705
e1:r3-e2:r2  0.475 -2.0701851  3.020185 0.9902110
e2:r3-e2:r2  1.750 -0.7951851  4.295185 0.2914242
e2:r3-e1:r3  1.275 -1.2701851  3.820185 0.6135909
```

Mas ainda assim temos resultados que podem não interessar. Mais especificamente, considere que estamos interessados nas comparações dos níveis de um fator dentro de cada um dos níveis do outro fator. Neste ponto, vamos fazer as comparações dos recipientes para cada uma das espécies, fazendo os cálculos passo a passo. Primeiro vamos obter a estimativa da variância dos resíduos, que é usada junto com o valor da *amplitude estudantizada* fornecida por `qtukey()` para obter o valor da *diferença mínima significativa* que no código a seguir armazenamos no objeto `dt`.

```
> s2 <- sum(resid(ex04.av)^2)/ex04.av$df.res
> dt <- qtukey(0.95, 3, 18) * sqrt(s2/4)
> dt
[1] 2.043945
```

Este valor é então usado para comparar as médias de interesse. Anteriormente armazenamos as médias para as combinações de todos os níveis dos fatores no objeto `ex04.m` onde as linhas se referem aos recipientes e colunas às espécies. No objeto `m1` armazenamos as médias para `espécie1` e na sequência são feitos cálculos para verificar a significância da diferença entre as médias dos recipientes para esta espécie.

```

> # comparação de médias de recipientes para espécie 1 :
> 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
[1]  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 *
> # comparação de médias de recipientes para espécie 2 :
> 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

```

No código mostrado anteriormente fazemos alguma manipulação dos objetos para formatar a saída. Esta sequência pode ser usada para definir uma função o que evitaria a digitação de todos estes comandos a cada comparação de médias desejada. Procedimento análogo pode ser adotado para fazer outras comparações de interesse.

26 Análise de covariância

26.1 Exemplo 1

Os dados disponíveis em [neste link](http://www.leg.ufpr.br/~paulojus/dados/exemplo12.txt) são um exemplo onde a *análise de covariância* pode ser utilizada. Neste caso temos uma variável resposta e duas variáveis explicativas sendo uma delas qualitativa (um fator) e outra quantitativa (numérica). Copie o arquivo com o conjunto de dados para sua área de trabalho e depois importe para o R com `read.table()`. Alternativamente, se o computador estiver conectado a internet `read.table()` pode ler diretamente o arquivo como mostrado a seguir.

O primeiro passo é a leitura e organização dos dados, em particular assegurando que a variável qualitativa indicadora dos tratamentos seja indicada como sendo um **fator** evitando que seus valores sejam interpretados como quantidades numéricas. Note que neste caso temos 2 variáveis numéricas, a resposta (**resp**) e a covariável (**cov**).

```
> ex12 <- read.table("http://www.leg.ufpr.br/~paulojus/dados/exemplo12.txt", header=T)
> ex12
```

	maq	cov	resp
1	1	20	36
2	1	25	41
3	1	24	39
4	1	25	42
5	1	32	49
6	2	22	40
7	2	28	48
8	2	22	39
9	2	30	45
10	2	28	44
11	3	21	35
12	3	23	37
13	3	26	42
14	3	21	34
15	3	15	32

```
> ex12$maq <- as.factor(ex12$maq)
> str(ex12)
```

```
'data.frame':      15 obs. of  3 variables:
 $ maq : Factor w/ 3 levels "1","2","3": 1 1 1 1 1 2 2 2 2 2 ...
 $ cov : int  20 25 24 25 32 22 28 22 30 28 ...
 $ resp: int  36 41 39 42 49 40 48 39 45 44 ...
```

```
> summary(ex12)
```

maq	cov	resp
1:5	Min. :15.00	Min. :32.0
2:5	1st Qu.:21.50	1st Qu.:36.5
3:5	Median :24.00	Median :40.0
	Mean :24.13	Mean :40.2
	3rd Qu.:27.00	3rd Qu.:43.0
	Max. :32.00	Max. :49.0

Os dados podem ser visualizados na Figura 68 produzida com o comando a seguir. Note-se que os dados de cada um dos tratamentos são indicados por cores (argumento `col`) e padrões de pontos (argumento `pch`) indexados pela variável `maq` que define os níveis dos tratamentos

```
> with(ex12, plot(resp~cov, col=c(1,2,4)[maq], pch=(1:3)[maq]))
```

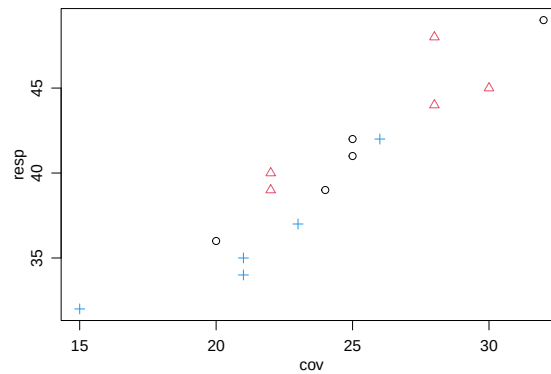


Figura 68: Dados para análise de covariância.

Na análise de covariância não temos ortogonalidade entre os fatores. Desta forma os testes de significância têm que ser obtidos em ajustes separados: (i) para o efeito de covariáveis, corrigido pelo efeito dos tratamentos qualitativos e (ii) para o efeito dos tratamentos qualitativos, corrigidos pelo efeito da covariável.

Primeiro vamos testar a inclinação (coeficiente β_1) da reta de regressão. Na análise de variância abaixo devemos considerar apenas o teste referente à variável `cov` que neste caso está corrigida para o efeito de `maq`. Note que para isto a variável `cov` tem que ser a última na especificação do modelo.

```
> ex12.cov <- aov(resp ~ maq + cov, data=ex12)
> summary(ex12.cov)
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
maq	2	140.40	70.20	27.59	5.17e-05 ***
cov	1	178.01	178.01	69.97	4.26e-06 ***
Residuals	11	27.99	2.54		

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

A seguir testamos o efeito do fator `maq` corrigindo para o efeito da covariável. Para isto basta inverter a ordem dos termos na especificação do modelo.

```
> ex12.trat <- aov(resp ~ cov + maq, data=ex12)
> summary(ex12.trat)
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
cov	1	305.13	305.13	119.933	2.96e-07 ***
maq	2	13.28	6.64	2.611	0.118
Residuals	11	27.99	2.54		

```
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Portanto, olhamos o primeiro quadro da ANOVA para verificar o efeito da covariável, e no segundo para verificar o efeito do tratamento. Se desejável poderia-se tomar os resultados de cada um deles para compor um quadro de análise, porém com a ressalva que, devido a não ortogonalidade, a soma das somas de quadrados não corresponde à soma de quadrados total. Entretanto, há uma função `Anova()` no pacote `car` do R que já monta tal quadro automaticamente conforme ilustrado a seguir.

```
> require(car)
Carregando pacotes exigidos: car
Carregando pacotes exigidos: carData
> Anova(ex12.cov, type="III")
```

Anova Table (Type III tests)

Response: resp

	Sum Sq	Df	F value	Pr(>F)	
(Intercept)	87.434	1	34.3664	0.0001089	***
maq	13.284	2	2.6106	0.1180839	
cov	178.014	1	69.9694	4.264e-06	***
Residuals	27.986	11			

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Note que esta função irá retornar o mesmo resultado para qualquer ordem dos termos no modelo, ou seja, no exemplo acima `Anova(ex12.cov, type="III")` e `Anova(ex12.trat, type="III")` retornam os mesmos resultados.

O argumento `type="III"` refere-se a um jargão consagrado pelo software SAS que corresponde a *soma de quadrados do tipo III*. Em geral nas funções básicas do R evita-se tal jargão e procura-se usar só conceitos ligados à parametrização do modelo através da definição dos **contrastes** e por isto tal terminologia está apenas em um pacote contribuído.

Neste caso a função `Anova` faz o mesmo que mostrado nas duas análises de variâncias iniciais, obtendo para cada termo a soma de quadrados quando este é corrigido para os demais, ou seja, colocado na última posição na especificação do modelo.

27 Efeitos aleatórios

27.1 Componentes de variância

27.1.1 Introdução

O problema que ilustra este tópico consiste em encontrar valores "padrão"(ou de referência) para os teores de elementos químicos. Para isto, amostras de referência supostamente de teores iguais foram enviadas a diferentes laboratórios nos quais determinações de teores foram feitas com replicações.

Como exemplo, considere os dados dos teores medidos de um único elemento mostrados a seguir e que podem ser obtidos em <http://www.leg.ufpr.br/~paulojus/dados/MgO.xls>

Lab	MgO
A	1.86
A	1.88
A	1.86
B	2.00
B	2.00
B	1.99
B	2.02
B	2.01
C	1.84
C	1.83
C	1.83
D	1.64
D	1.73
D	1.68
E	0.28
E	0.31
E	0.68
F	1.88
F	1.87
F	1.86
G	1.87
G	1.87
G	1.86
H	1.85
H	1.86
H	1.85

```
> require(readxl)
Carregando pacotes exigidos: readxl
> mgo <- as.data.frame(read_excel("MgO.xls"))
> head(mgo)
> str(mgo)
> summary(mgo)
```

Pode-se identificar duas fontes de variação nos valores medidos, uma devido à variabilidade entre laboratórios e outra devida à variabilidade das replicações feitas nos laboratórios. O objetivo é encontrar um valor "característico" para as amostras, que seria dado por alguma

"média" adequada, associada a uma medida de variabilidade desta média, por exemplo dada por um intervalo de confiança. Além disto deseja-se estimar os "componentes de variância", isto é, medidas da variabilidade entre e dentro de laboratórios.

Nos resultados a seguir ajustamos um modelo ajustado com a função `lme()` do pacote **nlme**.

```
> require(nlme)
```

```
Carregando pacotes exigidos: nlme
```

O `summary()` mostra um resumo dos resultados mais importantes do ajuste do modelo incluindo as estimativas da média (**Fixed Effects: Intercept**) e dos desvios padrão entre laboratórios (**Random Effects: Intercept**) e das replicatas (**Random Effects: Residual**).

```
> mgo.lme <- lme(MgO ~ 1, random= ~1|Lab, mgo)
> summary(mgo.lme)
```

```
Linear mixed-effects model fit by REML
```

```
Data: mgo
```

```
      AIC      BIC    logLik
-13.63106 -9.974431  9.815529
```

```
Random effects:
```

```
Formula: ~1 | Lab
```

```
(Intercept) Residual
```

```
StdDev:    0.5112631 0.07633555
```

```
Fixed effects: MgO ~ 1
```

```
      Value Std.Error DF   t-value p-value
```

```
(Intercept) 1.675227 0.1813956 18  9.235211      0
```

```
Standardized Within-Group Residuals:
```

```
      Min      Q1      Med      Q3      Max
-1.99864232 -0.06883532 -0.02227551  0.10285379  3.24138039
```

```
Number of Observations: 26
```

```
Number of Groups: 8
```

O intervalo de confiança para média e as estimativas das variâncias e desvios padrões podem ser obtidos como mostrado a seguir.

```
> intervals(mgo.lme, which="fixed")
```

```
Approximate 95% confidence intervals
```

```
Fixed effects:
```

```
      lower      est.      upper
(Intercept) 1.294129 1.675227 2.056325
```

```
> VarCorr(mgo.lme)
```

```
Lab = pdLogChol(1)
```

```
      Variance StdDev
(Intercept) 0.261389947 0.51126309
Residual    0.005827116 0.07633555
```

27.1.2 Avaliando o ajuste e qualidade dos dados

Os resultados mostrados anteriormente devem ser vistos apenas como uma ilustração dos comandos básicos para obtenção dos resultados. Entretanto, não se deve tomar os resultados obtidos como corretos ou definitivos pois uma análise criteriosa deve verificar anomalias dos dados e adequação a pressupostos do modelo.

```
> print(plot(mgo.lme))
```

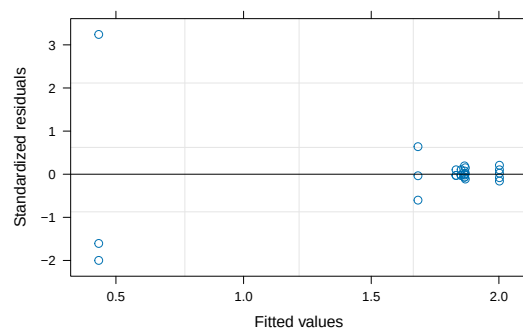


Figura 69: Gráfico de resíduos do modelo ajustado

Os gráficos de resíduos das figuras 69 e 70 mostram observações discrepantes e pode-se detectar que ocorrem nos dados do Laboratório E. No primeiro desses todos os resíduos são visualizados conjuntamente, enquanto no segundo usa-se gráficos condicionais do sistema gráfico fornecido pelo pacote **lattice** para separar os resíduos de cada laboratório.

A observação de valor 0.68 do laboratório E é bastante diferente das demais replicatas deste laboratório (0.28 e 0.31), sendo que este dado também foi considerado suspeito pela fonte dos dados. Uma possível alternativa é, em acordo com o responsável pelos dados, optar por remover este dado da análise o que pode ser feito com o comando a seguir.

```
> mgo1 <- subset(mgo, !(Lab == "E" & MgO > 0.6))
> dim(mgo1)
[1] 25 2
```

O modelo ajustado assume que os dados possuem distribuição normal e os gráficos de perfil de verossimilhança do parâmetro da transformação Box-Cox na figura 71 mostram que, excluindo-se o dado atípico, a transformação não é necessária.

```
> require(MASS)
> with(mgo, boxcox(MgO~Lab, lam=seq(1.5,5.5, len=200)))
> with(mgo1, boxcox(MgO~Lab, lam=seq(0,3, len=200)))
> #with(mgo2, boxcox(MgO~Lab, lam=seq(1.5,10, len=200)))

> print(plot(mgo.lme, resid(.) ~ fitted(.) | Lab, abline = 0))
```

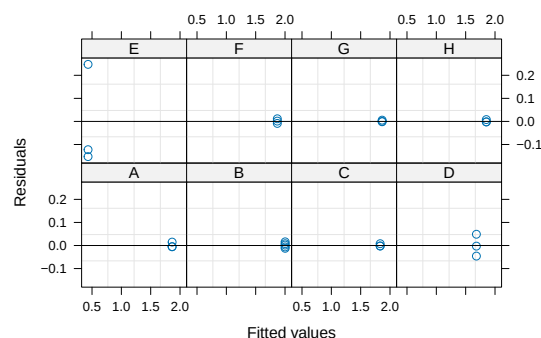


Figura 70: Gráfico de resíduos para cada laboratório do modelo ajustado.

Carregando pacotes exigidos: MASS

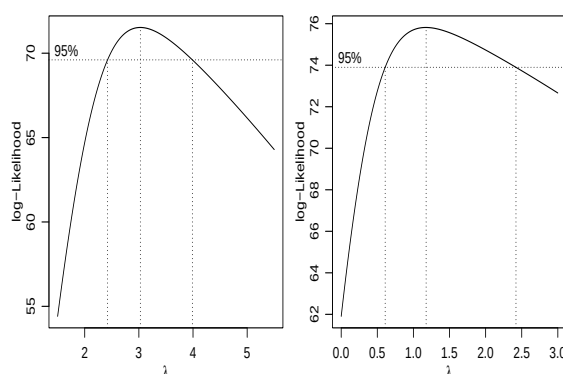


Figura 71: Perfis de verossimilhança do parâmetro da transformação Box-Cox na presença e ausência do ponto atípico do Laboratório E.

O modelo ajustado com o novo conjunto de dados apresenta resultados diferentes do anterior, reduzindo a estimativa de variância entre as replicatas.

```
> mgo1.lme <- lme(Mg0 ~ 1, random = ~1|Lab, mgo1)
> summary(mgo1.lme)
```

Linear mixed-effects model fit by REML

Data: mgo1

	AIC	BIC	logLik
	-58.04521	-54.51105	32.02261

Random effects:

Formula: ~1 | Lab

	(Intercept)	Residual
StdDev:	0.5577469	0.01888633

Fixed effects: Mg0 ~ 1

	Value	Std.Error	DF	t-value	p-value
(Intercept)	1.659103	0.1972315	17	8.411958	0

Standardized Within-Group Residuals:

	Min	Q1	Med	Q3	Max
	-2.4333720	-0.3487900	-0.1439774	0.3565135	2.5808360

Number of Observations: 25

Number of Groups: 8

```
> intervals(mgo1.lme, which="fixed")
```

Approximate 95% confidence intervals

Fixed effects:

	lower	est.	upper
(Intercept)	1.242981	1.659103	2.075225

```
> VarCorr(mgo1.lme)
```

Lab = pdLogChol(1)

	Variance	StdDev
(Intercept)	0.3110815938	0.55774689
Residual	0.0003566936	0.01888633

Além disto, nota-se que na verdade todas as observações do Laboratório E parecem atípicas com valores inferiores aos obtidos nos demais laboratórios. Poderia-se então considerar ainda remover todas as observações deste laboratório.

```
> print(plot(mgo1.lme, resid(., type = "p") ~ fitted(.) | Lab, abline = 0))
```

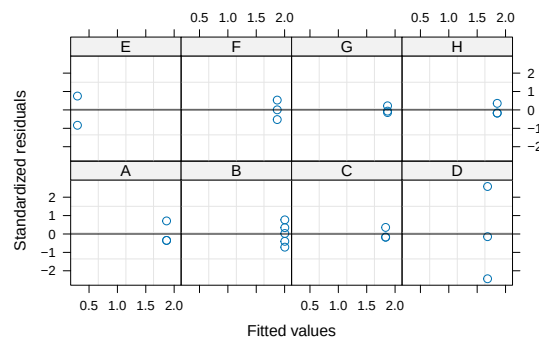


Figura 72: Gráfico de resíduos para cada laboratório do modelo ajustado.

```
> mgo2 <- subset(mgo, Lab != "E")
> dim(mgo2)
[1] 23 2
> mgo2.lme <- lme(MgO ~ 1, random= ~1|Lab, mgo2)
> summary(mgo2.lme)
Linear mixed-effects model fit by REML
Data: mgo2
      AIC      BIC    logLik
-77.11573 -73.8426 41.55786

Random effects:
Formula: ~1 | Lab
      (Intercept)  Residual
StdDev:  0.09317201 0.01872537

Fixed effects:  MgO ~ 1
              Value Std.Error DF  t-value p-value
(Intercept) 1.854044 0.03543846 16 52.31728      0

Standardized Within-Group Residuals:
      Min      Q1      Med      Q3      Max
-2.5760112 -0.3054222 -0.1399831  0.3484244  2.4812983

Number of Observations: 23
Number of Groups: 7
> intervals(mgo2.lme, which="fixed")
Approximate 95% confidence intervals

Fixed effects:
      lower      est.      upper
(Intercept) 1.778918 1.854044 1.92917
> VarCorr(mgo2.lme)
Lab = pdLogChol(1)
      Variance StdDev
(Intercept) 0.0086810227 0.09317201
Residual    0.0003506395 0.01872537
```

Os resultados são substancialmente diferentes e a decisão de exclusão ou não dos dados deste Laboratório deve ser cuidadosamente investigada dentro do contexto destes dados e em conjunto com especialista da área.

27.1.3 Fundamentos

Assumindo que efeitos aleatórios podem ser usados para descrever o efeito de laboratórios, podemos descrever os teores por um modelo de efeitos aleatórios:

$$Y_{ij} = \mu + \varepsilon_i + \epsilon_{ij},$$

em que y_{ij} são valores observados na j -ésima medida feita no i -ésimo laboratório, μ é o valor real do elemento na amostra padrão, $\varepsilon_i \sim N(0, \sigma_\varepsilon^2)$ é o efeito aleatório do i -ésimo laboratório e σ_ε^2 que representa a variabilidade de medidas fornecidas por diferentes laboratórios (entre laboratórios) e $\epsilon_{ij} \sim N(0, \sigma_\epsilon^2)$ é o termo associado à j -ésima medida feita no i -ésimo laboratório e σ_ϵ^2 é a variabilidade das medidas de replicatas dentro dos laboratórios.

O problema então consiste em estimar μ e a variância associada à esta estimativa, que por sua vez está associada aos valores dos parâmetros de variância do modelo σ_ε^2 e σ_ϵ^2 . Esses últimos parâmetros são chamados de *componentes de variância*. Diferentes métodos de estimação são propostos na literatura tais como estimadores de momentos baseados na análise de variância, estimadores *minque* (estimadores de norma quadrática mínima), estimadores de máxima verossimilhança e máxima verossimilhança restrita.

Sob o modelo assumido as observações têm distribuição normal

$$Y \sim N(\mathbb{1}\mu, V),$$

em que $\mathbb{1}$ é um vetor unitário de dimensão igual ao número de observações n e V é a matriz de variâncias e covariâncias das observações com elementos dados por: $\text{Var}(Y_{i,j}) = \sigma_\varepsilon^2 + \sigma_\epsilon^2$, a variância de cada observação individual; $\text{Cov}(Y_{i,j}, Y_{i,j'}) = \sigma_\varepsilon^2$ a covariância entre observações diferentes do mesmo laboratório, e os demais elementos são nulos. No caso balanceado, isto é, igual número de replicatas nos diferentes laboratórios, a matriz V pode ser obtida por um produto de Kronecker simples entre matrizes diagonais e unitárias multiplicadas pelos componentes de variância.

Considerando os recursos computacionais atualmente disponíveis e as propriedades dos diferentes estimadores, nossa preferência é pelo uso de estimadores de máxima verossimilhança restrita. Estes estimadores são obtidos maximizando-se a função de verossimilhança de uma projeção do vetor dos dados no espaço complementar ao definido pela parte fixa do modelo. Tipicamente, os estimadores de σ_ε^2 e σ_ϵ^2 são obtidos por maximização numérica de tal função e o estimador do parâmetro de interesse e sua variância são então obtidos por:

$$\hat{\mu} = (\mathbb{1}'\hat{V}^{-1}\mathbb{1})^{-1}\mathbb{1}'\hat{V}^{-1}y \quad (6)$$

$$\hat{\text{Var}}(\hat{\mu}) = (\mathbb{1}'\hat{V}^{-1}\mathbb{1})^{-1} \quad (7)$$

em que $\hat{V}$ é a matriz de variâncias e covariâncias estimada das observações obtida a partir das estimativas $\hat{\sigma}_\varepsilon^2$ e $\hat{\sigma}_\epsilon^2$.

No exemplo em questão são estes os estimadores utilizados para obter as estimativas mostradas na seção anterior (ver o resultado de `summary(mgo1.lme)`), com valores mostrados novamente a seguir.

```
> names(mgo1.lme)
[1] "modelStruct"  "dims"          "contrasts"      "coefficients"  "varFix"
[6] "sigma"        "apVar"         "logLik"         "numIter"       "groups"
[11] "call"        "terms"        "method"        "fitted"        "residuals"
[16] "fixDF"       "na.action"    "data"

> mgo1.lme$coeff$fixed
(Intercept)
1.659103
```

```
> VarCorr(mgo1.lme)[,1]
      (Intercept)      Residual
"0.3110815938" "0.0003566936"
```

O intervalo de confiança para média pode então ser obtido por:

$$\hat{\mu} \pm t_{1-\alpha/2, n-1} \sqrt{\hat{\text{Var}}(\hat{\mu})},$$

Nos comandos a seguir mostramos a obtenção do intervalo segundo cálculos dessa expressão e a equivalência com o informado pela função `intervals.lme()`.

```
> mgo1.lme$varFix
      (Intercept)
(Intercept) 0.03890025
> with(mgo1.lme, coefficients$fixed + qt(c(0.025, 0.975), df=fixDF$X) * sqrt(varFix))
Warning in qt(c(0.025, 0.975), df = fixDF$X) * sqrt(varFix): Reciclar uma array de comprimento 1 na aritmética
Em vez disso, use c() ou as.vector().
[1] 1.242981 2.075225
> intervals(mgo1.lme, which="fixed")
Approximate 95% confidence intervals

Fixed effects:
      lower      est.      upper
(Intercept) 1.242981 1.659103 2.075225
```

Para uma observação individual o intervalo é dado por

$$y \pm t_{1-\alpha/2, n-1} \sqrt{\sigma_\epsilon^2 + \sigma_\epsilon^2};$$

e as estimativas $\hat{\sigma}_\epsilon^2$ e $\hat{\sigma}_\epsilon^2$ podem ser obtidas da seguinte forma.

```
> vcomp <- as.numeric(VarCorr(mgo1.lme)[,1])
> vcomp
[1] 0.3110815938 0.0003566936
```

O coeficiente de correlação intraclasse reflete a relação entre a variabilidade das observações dentro dos laboratórios em relação a variabilidade total. É definido pela expressão a seguir e calculado como mostrado nas linhas de comando.

$$\rho = \frac{\sigma_\epsilon^2}{\sigma_\epsilon^2 + \sigma_\epsilon^2}.$$

```
> vcomp[1]/sum(vcomp)
[1] 0.9988547
```

27.1.4 Alternativas de código

O pacote **lme4** reimplementa algumas funcionalidades do **nlme** onde o modelo é definido indicando os termos aleatórios entre parênteses na fórmula e eliminando o uso do argumento `random`.

```
> require(lme4)
Carregando pacotes exigidos: lme4
Carregando pacotes exigidos: Matrix
```

Anexando pacote: 'lme4'

O seguinte objeto é mascarado por 'package:nlme':

```
lmList
```

O comando para se obter uma análise equivalente à anterior é mostrado a seguir. Os resultados são apresentados de forma diferente, porém os elementos são equivalentes.

```
> mgo1.lmer <- lmer(MgO ~ 1 + (1|Lab), mgo1)
```

```
> summary(mgo1.lmer)
```

Linear mixed model fit by REML ['lmerMod']

Formula: MgO ~ 1 + (1 | Lab)

Data: mgo1

REML criterion at convergence: -64

Scaled residuals:

Min	1Q	Median	3Q	Max
-2.4334	-0.3488	-0.1440	0.3565	2.5808

Random effects:

Groups	Name	Variance	Std.Dev.
Lab	(Intercept)	0.3110816	0.55775
	Residual	0.0003567	0.01889

Number of obs: 25, groups: Lab, 8

Fixed effects:

	Estimate	Std. Error	t value
(Intercept)	1.6591	0.1972	8.412

A opção padrão é o ajuste por *máxima verossimilhança restrita*. Estimativas de máxima verossimilhança podem ser obtidas usando o argumento REML=FALSE.

```
> mgo1.lmer.ml <- lmer(MgO ~ 1 + (1|Lab), mgo1, REML=FALSE)
```

```
> summary(mgo1.lmer.ml)
```

Linear mixed model fit by maximum likelihood ['lmerMod']

Formula: MgO ~ 1 + (1 | Lab)

Data: mgo1

AIC	BIC	logLik	-2*log(L)	df.resid
-59.5	-55.9	32.8	-65.5	22

Scaled residuals:

Min	1Q	Median	3Q	Max
-2.4333	-0.3482	-0.1434	0.3570	2.5809

Random effects:

Groups	Name	Variance	Std.Dev.
Lab	(Intercept)	0.2721697	0.52170
	Residual	0.0003567	0.01889

Number of obs: 25, groups: Lab, 8

Fixed effects:

	Estimate	Std. Error	t value
(Intercept)	1.6591	0.1845	8.993

28 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.

28.1 Relações entre a distribuição normal e a χ^2

Resultado 1: Se $Z \sim N(0, 1)$ então $Z^2 \sim \chi^2_{(1)}$.

Veamos 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 comandos abaixo e o resultado está no gráfico da esquerda da [Figura 73](#).

```
> set.seed(17)
> 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 73](#) mostra o histograma dos quadrados dos 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 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:

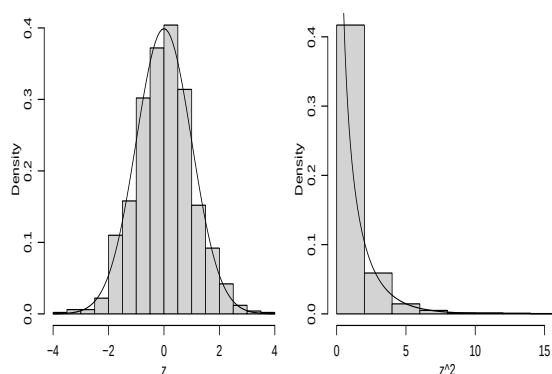


Figura 73: Histograma da amostra e a curva teórica da distribuição normal padrão (esquerda) e histograma dos valores ao quadrado com a curva teórica da distribuição $\chi^2_{(1)}$ (direita).

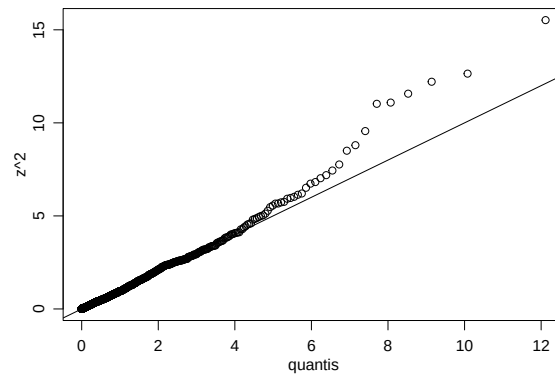


Figura 74: Comparando dados e quantis da χ^2 utilizando o *qq-plot*

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 distribuição χ^2 usando a função `qchisq` em um conjunto de probabilidades geradas pela função `ppoints`. Por fim usamos a função `qqplot` para obter o gráfico mostrado na Figura 74, adicionando neste gráfico a bissetriz do primeiro quadrante para facilitar a avaliação do ajuste.

```
> quantis <- qchisq(ppoints(length(z)), df=1)
> qqplot(quantis, z^2)
> abline(0,1)
```

Note que o comando `qchisq(ppoints(length(z)), df=1)` acima está concatenando 3 comandos e calcula os quantis da χ^2 a partir de uma sequência de valores de probabilidade gerada por `ppoints`. O número de elementos desta sequência deve igual ao número de dados e por isto usamos `length(z)`.

Resultado 2: Se $Z_1, Z_2, \dots, Z_n \sim N(0, 1)$ então $\sum_1^n Z_i^2 \sim \chi_{(n)}^2$.

Para ilustrar este resultado vamos gerar 10.000 amostras de 3 elementos cada da distribuição normal padrão, elevar os valores ao quadrado e, para cada amostra, somar os quadrados dos três números. Na Figura 75 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 $\chi_{(3)}^2$.

```
> 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)
```

28.2 Distribuição amostral da média de amostras da distribuição normal

Resultado 3: Se $Y_1, Y_2, \dots, Y_n \sim N(\mu, \sigma^2)$ então $\bar{y} \sim N(\mu, \sigma^2/n)$.

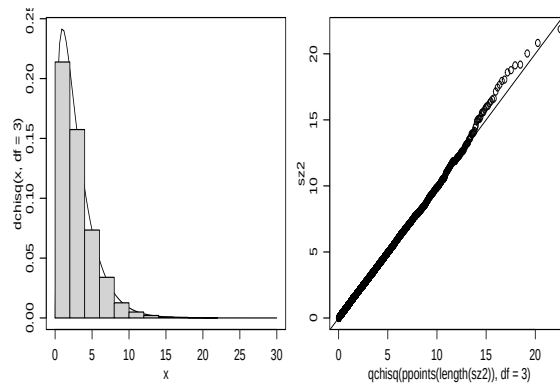


Figura 75: Histograma da uma amostra da soma dos quadrados de três valores da normal padrão e a curva teórica da distribuição de $\chi^2_{(3)}$ (esquerda) e o respectivo *qq-plot*.

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] 99.97721
> 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 por 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 76](#) 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.

28.3 Exercícios

1. Ilustrar usando simulação o resultado que afirma que para o estimador $S^2 = \sum \frac{(Y_i - \bar{Y})^2}{n-1}$ da variância de uma distribuição normal, a variável $V = (n-1)S^2/\sigma^2$ tem distribuição χ^2_{n-1} .

DICA: Você 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 $V = (n-1)S^2/\sigma^2$,

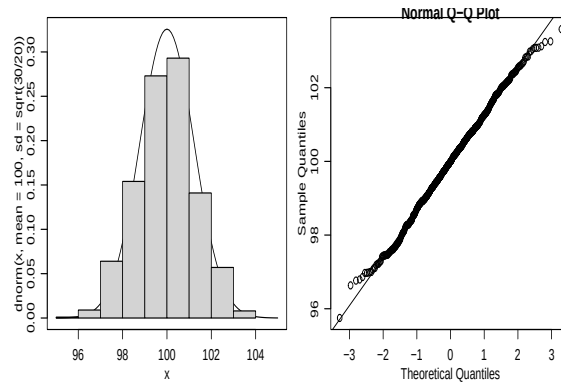


Figura 76: Histograma de uma amostra da distribuição amostral da média e a curva teórica da distribuição e o respectivo *qq-plot*.

- faça um histograma com os valores V e compare com a curva de uma distribuição χ^2_{n-1} .
2. No exercício anterior compare os valores teóricos $E[S^2] = \sigma^2$ e $Var[S^2] = \frac{2\sigma^4}{n-1}$ 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: $T_1 = \bar{x}$, 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 $Y_1, \dots, Y_n$ a.a. de uma distribuição $N(\mu, \sigma^2)$. 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: comece verificando passo a passo, como no exercício anterior, o que é necessário para ilustrar este resultado.

5. Ilustrar o resultado que diz que o quociente de duas variáveis independentes com distribuição χ^2 tem distribuição F .

29 Agrupando comandos, execução condicional, controle de fluxo, "loops" e a "família" *apply

29.1 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 ver 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 pode 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; y; z
[1] 1 2 3
[1] 5 6 7
[1] 5.000000 3.000000 2.333333
```

29.2 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, entretanto, 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 separar dados por grupos. Por exemplo considere o conjunto de dados `mtcars`, 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
[16] 15.5 15.2 13.3 19.2

$`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 `aggregate()`.

29.3 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 amostras de tamanho $1, 2, \dots, n$ e calcular 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 no R. Na verdade, para este tipo de cálculo recomenda-se o uso de funções do tipo `*apply` que veremos no restante desta sessão.

29.4 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] 1.00687

$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 gerávamos amostras com tamanhos sequenciais 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
```

29.5 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 alguns 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=0.10), 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 armazenado 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]
modeloA	0.02725767	-0.01017973	0.0958355	0.02058979	0.04582751	0.07898205
modeloB	0.01706928	-0.02781770	0.1023454	0.02210935	0.06210404	0.05914628

```
[,7]      [,8]      [,9]      [,10]
modeloA 0.06122656 -0.05981805 0.006781871 -0.02798788
modeloB 0.04085053 -0.05680834 -0.020411456 -0.02029610
```

Os comandos acima podem ser reescritos em versões simplificadas:

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

	[1]	[6]
0.027257675	-0.010179733	0.095835502
0.020589788	0.045827513	0.078982050
0.061226561	-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]
A	0.02725767	-0.01017973	0.0958355	0.02058979	0.04582751	0.07898205	0.06122656
B	0.01706928	-0.02781770	0.1023454	0.02210935	0.06210404	0.05914628	0.04085053

```
[,8]      [,9]      [,10]
A -0.05981805 0.006781871 -0.02798788
B -0.05680834 -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 coeficiente de variação do consumo para cada tipo de câmbio.

```
> 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
[9] 56.41214 65.46560 35.99390 42.67566 40.26776 47.61359 57.92209 60.69673
[17] 48.80234 44.29422 44.48886 39.02277 60.93054 32.73959 39.37867 56.89312
[25] 37.06637 61.45986 44.66166 50.60778 37.78913 39.13208 48.53931 43.29661
[33] 66.03602 65.55652 58.05864 55.21829 45.90542 45.01864 37.73984 38.00313
[41] 34.85114 34.24760 65.07629 49.01286 62.37572 38.36997 57.93003 39.72861
[49] 66.62899 45.37572

> y

[1] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A" "B" "B" "B" "B" "B" "B" "B" "B"
[20] "B" "C" "C" "C" "C" "C" "C" "C" "C" "C" "C" "D" "D" "D" "D" "D" "D" "D"
[39] "D" "D" "E" "E" "E" "E" "E" "E" "E" "E" "E" "E" "E" "E" "E" "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 matriz 1000×300 representando 1000 amostras de tamanho 300. O que desejamos é calcular a média de cada uma das amostras. Os códigos a seguir mostram 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)}
+ )
```

```

usuário  sistema decorrido
0.006    0.000    0.006

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

usuário  sistema decorrido
0.005    0.000    0.005

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

usuário  sistema decorrido
0.001    0.000    0.001

```

A função `rowMeans()` é substancialmente mais eficiente (menos tempo de execução. Outras funções similares 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 matriz indicando, 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
+   }
+ })

usuário  sistema decorrido
0.048    0.017    0.066

```

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)
+ })

```

```
usuário   sistema decorrido
0.003     0.000     0.002
```

```
> 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 matriz 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
+ }
+ })
```

```
usuário   sistema decorrido
0.055     0.001     0.057
```

```
> 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)
+ })
```

```
usuário   sistema decorrido
0.002     0.000     0.003
```

```
> identical(A,B)
```

```
[1] TRUE
```

Uso da família `*apply` – outros exemplos Os exemplos a seguir foram retirados de mensagens enviadas à lista `R_STAT`.

1. adaptado de mensagem enviada por Silvano C Costa

Tenho uma pergunta onde pode haver mais de uma resposta, por exemplo:

Q1 - Qual Esporte você pratica:

1.()Futebol 2.()Volei 3.() Natação 4.()Atletismo

Q2 - Sexo:

1.Masculino() 2.Feminino()

Então teria os dados dessa forma:

Q1.1 Q1.2 Q1.3 Q1.4 Q2

```
1      0      1      0      1 => Homem Praticante de Futebol,Natacao
0      1      1      0      2 => Mulher praticante de Volei e Natação
0      0      0      1      2 => Mulher praticante de Atletismo
```

Gostaria de criar uma tabela cruzada entre essas variáveis:

	M	F
Futebol	21	10
Natação	13	20
Volei	5	2
Atletismo	10	10

Para mostrar como obter a solução, como não temos o questionário aqui vamos primeiro simular dados deste tipo como se tivéssemos 75 questionários.

```
> esportes <- as.data.frame(matrix(sample(c(0,1), 300, rep=TRUE), nc=4))
> names(esportes) <- c("Futebol", "Natacao", "Volei", "Atletismo")
> esportes$S <- sample(c("M", "F"), 75, rep=TRUE)
> dim(esportes)
```

```
[1] 75  5
```

```
> head(esportes)
```

	Futebol	Natacao	Volei	Atletismo	S
1	1	1	0	0	F
2	1	0	0	0	F
3	0	1	0	0	M
4	1	0	0	0	M
5	1	1	1	1	M
6	0	1	1	1	M

Solução 1: Para cada esporte podemos contar os praticantes de cada sexo em cada esporte, separadamente utilizando `table()` e verificando a segunda linha da tablea a seguir.

```
> with(esportes, table(Futebol, S))
```

	S
Futebol	F M
0	16 24
1	21 14

Desta forma, podemos obter a tabela desejada combinando os resultados de tabelas para cada esporte.

```
> with(esportes, rbind(table(Futebol, S)[2,],
+      table(Natacao, S)[2,], table(Volei, S)[2,], table(Aletismo, S)[2,]))
      F  M
[1,] 21 14
[2,] 20 17
[3,] 15 18
[4,] 19 21
```

Solução 2: alternativamente, podemos usar `sapply()` para tomar cada esporte e, como os dados são codificados em 0/1, usar `tapply()` para somar os praticantes (1) de cada sexo.

```
> sapply(esportes[,1:4], function(x) tapply(x, esportes$S, sum))
      Futebol Natacao Volei Aletismo
F           21         20        15         19
M           14         17        18         21
```

2. Adaptado de mensagem enviada por André

Queria fazer uma amostragem e tirar as informações. Tenho duas amostras. Aplico o teste t e tenho um P-valor.

```
a <-c(1,2,4,5,3,4,5,6,6,7,2)
b <-c(5,3,4,5,3,4,5,3,4,3,5)
```

Eu gostaria de juntar estes dois vetores num mesmo vetor e fazer 1000 reamostragens neste vetor de tamanho do vetor a e com reposição.

```
e <- c(5,3,4,5,3,4,5,3,4,3,5,1,2,4,5,3,4,5,6,6,7,2)
```

Depois eu queria ver aplicar o teste t(nas amostras) e ver como se comportam estes p-valores.

Inicialmente vamos entrar com os dados e obter o teste-t.

```
> a <-c(1,2,4,5,3,4,5,6,6,7,2)
> b <-c(5,3,4,5,3,4,5,3,4,3,5)
> tt.ab <- t.test(a, b)
> tt.ab
```

Welch Two Sample t-test

```
data: a and b
t = 0.14228, df = 14.14, p-value = 0.8889
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.278239  1.460057
sample estimates:
mean of x mean of y
 4.090909  4.000000
```

Agora obtemos as 1000 reamostras deste vetor cada uma com $11 \times 2 = 22$ valores utilizando `sample()`. As reamostras serão arranjadas num array de dimensão $11 \times 2 \times 1000$.

```
> e <- c(5,3,4,5,3,4,5,3,4,3,5,1,2,4,5,3,4,5,6,6,7,2)
> reamostras <- array(sample(e, length(e)*1000, rep=T),
+                      dim=c(length(e)/2,2,1000))
```

Portanto cada elemento da terceira dimensão corresponde a uma reamostra. Para fazer os testes-t nas 1000 reamostras utilizamos `apply()` que vai gerar uma lista de 1000 elementos com resultados dos testes.

```
> TT <- apply(reamostras, 3, function(x) t.test(x[,1], x[,2]))
```

Para ver o resultado do teste em uma das amostras selecionamos um elemento da lista.

```
> TT[[1]]
```

```
Welch Two Sample t-test
```

```
data: x[, 1] and x[, 2]
t = 1.3948, df = 16.273, p-value = 0.1818
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.4707033  2.2888851
sample estimates:
mean of x mean of y
 4.272727  3.363636
```

Finalmente pode-se extrair uma quantidade de interesse dos resultados, como no exemplo a seguir, extraímos os p-valores.

```
> names(TT[[1]])

[1] "statistic" "parameter" "p.value"    "conf.int"  "estimate"
[6] "null.value" "stderr"    "alternative" "method"    "data.name"

> pvals <- sapply(TT, function(x) x$p.value)
```

Os gráficos na Figura 77 mostram valores obtidos na amostra, comparados aos valores obtidos na reamostragem. À esquerda é mostrado a estimativa da diferença de médias, e à direita os p-valores.

```
> hist(sapply(TT, function(x) diff(x$est)),main="Diferença de médias")
> abline(v=diff(tt.ab$est), col=2, lwd=3)
> hist(sapply(TT, function(x) x$p.val), main="P-valor")
> abline(v=tt.ab$p.val, col=2)
```

OBS: note que estão sendo usadas as opções *default* do teste-t para comparação de duas amostras dadas em `t.test()` (bilateral, não pareado, variâncias diferentes, etc). Para alterar algum argumento basta acrescentar o argumento desejados na chamada de `apply()`.

```
> TT <- apply(reamostras, 3,
+           function(x, ...) t.test(x[,1], x[,2], ...),
+           var.equal=TRUE)
```

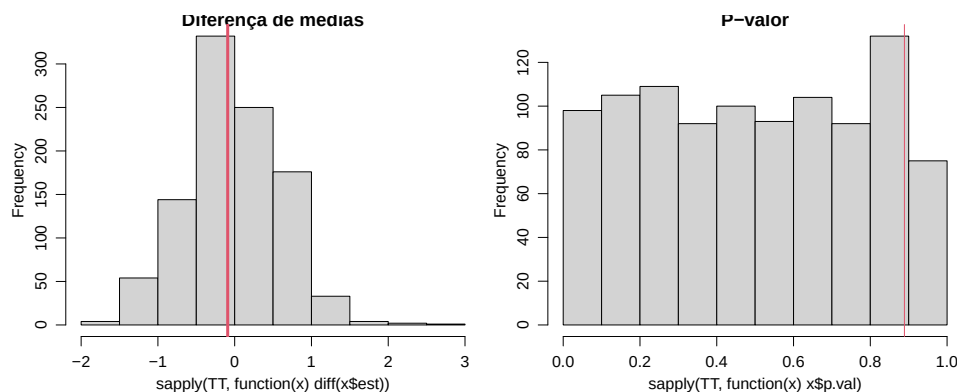


Figura 77: Valores da diferença de médias (esquerda) e p-valor (direita) da amostra (linhas verticais) comparados aos histogramas dos valores obtidos nas reamostras.

29.6 Extensões da família `*apply`

As funções da família `*apply` têm o propósito geral de aplicar uma função repetidamente a diversas partes de um conjunto de dados. Além das mencionadas acima várias outras têm sido implementadas. Notadamente se destacam as do pacote **plyr** (`a_ply`, `d_ply`, `l_ply`, `r_ply`, dentre várias outras).

Por exemplo considere a função `l_ply` que aplica uma função em elementos uma lista porém em situações onde estamos interessados apenas em um *efeito colateral* (gráfico, arquivo gravado em disco, etc) de função e não no resultado que ela retorna. Como exemplo considere o conjunto de dados `trees`, uma data-frame que possui três colunas com diâmetros, alturas e volumes de árvores. Suponha que desejamos fazer o histograma das três variáveis, e então uma possível forma seria utilizar o comando `lapply()`.

```
> data(trees)
> par(mfrow=c(1,3))
> lapply(trees, hist)
```

Neste caso, além de produzir os gráficos os resultados dos cálculos do histograma são retornados na tela, o que pode ser indesejável e/ou desnecessário. A função `l_ply()` produz o mesmo resultado gráfico porém sem que os resultados dos cálculos do histograma sejam retornados.

```
> require(plyr)
> l_ply(trees, hist)
```

Para mais informações sobre o pacote **plyr** consulte as informações e documentações em <http://had.co.nz/plyr/>.

30 Ajuste de modelos não lineares

Modelos não lineares permitem o ajuste de relações entre quantidades de interesse mais complexas que relações lineares ou linearizáveis. Em muitos casos tais modelos têm a sua forma funcional específica para o problema sendo tratado, relacionada a algum mecanismo (biológico, físico, etc) inerente ao processo em questão.

Nesta seção vamos ilustrar o ajuste de modelos não lineares com dados da área de física de solos. Utilizaremos a função `nls()`, cujo nome é um acrônimo para *non-linear least squares*. Esta função é bastante flexível e incorpora diversas opções para fazer ajustes incluindo características do modelo, tipo de ajuste e controle dos algoritmos disponíveis.

Diferentemente dos modelos lineares, o ajuste de modelos não lineares não permite que as expressões dos estimadores dos parâmetros desconhecidos do modelo sejam obtidas analiticamente, sendo portanto necessário o uso de métodos numéricos. Inicialmente mostramos um ajuste feito de forma ingênua (*naïve*), declarando apenas a função e valores iniciais. Tal procedimento, embora simples, pode ser ineficiente para o uso de métodos numéricos. Entretanto, o ajuste com `nls()` pode incorporar procedimentos que tendem a aprimorar o comportamento dos métodos numéricos tais como o fornecimento de funções que informem sobre a derivada do modelo sendo ajustado, inicialização automática com valores iniciais obtidos automaticamente, linearização parcial do modelo, além da escolha e calibragem dos algoritmos. O objetivo aqui não é o de investigar todas estas opções, mas apenas fornecer os elementos iniciais para ilustrar a possibilidade de se obter tais resultados usando o R.

30.1 Exemplo: o modelo de van Genuchten

Inicialmente discutiremos como efetuar um único ajuste para um conjunto de dados e algumas sugestões para examinar resultados. Mais adiante 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 uma questão levantada pelo Prof. Álvaro Pires da Silva do Departamento de Ciência do Solo da ESALQ/USP e refere-se ao ajuste da equação de van Genuchten para a *curva de retenção de água no solo* (ou *curva característica de água no solo*).

Informalmente falando, a equação de van Genuchten é um dos modelos matemáticos utilizados para descrever a curva característica de água no solo. O modelo caracteriza a armazenagem de água através de relação entre a umidade e o potencial matricial. Para determinação da curva característica de água o procedimento usual é o de se tomar uma amostra que é submetida a diferentes tensões em condições de laboratório. Para cada tensão aplicada, a amostra perde parte do conteúdo de água e mede-se a umidade residual na amostra. A partir dos pares de pontos com valores medidos de tensão e umidade, obtém-se a curva de retenção de água no solo que descreve a variação da umidade em função dos valores de tensão. O modelo de van Genuchten é dado pela seguinte equação:

$$\theta = \theta_R + (\theta_S - \theta_R) \left[\frac{1}{1 + (\alpha \Psi_m)^n} \right]^{1-(1/n)} \quad (8)$$

em que Ψ_m é o potencial matricial aplicado à amostra e θ é a umidade volumétrica medida na amostra. Os parâmetros desconhecidos do modelo são θ_S e θ_R que correspondem à umidade volumétrica na saturação e residual, respectivamente, α e n que definem o formato da curva sendo que o primeiro representa o inverso do potencial de entrada de ar e o segundo é um índice da distribuição dos tamanhos de poros. Portanto são obtidos dados para os pares de pontos (Ψ_m, θ) e $(\theta_S, \theta_R, \alpha, n)$ são parâmetros desconhecidos a serem estimados e que caracterizam a curva de retenção.

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 do conjunto de dados original que contém diversas amostras. O objetivo é determinar a curva de retenção de água no solo estimada segundo modelo de van Genuchten para cada uma das amostras. No objeto `cra` a primeira coluna (`am`) é a identificação da amostra, a segunda (`pot`) é o potencial aplicado e a terceira (`u`) é 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/dados/cra.csv", head=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
```

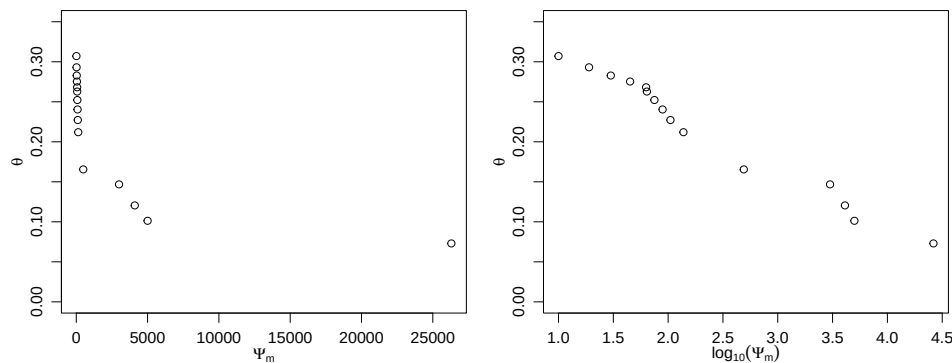
Inicialmente vamos nos concentrar na discussão do ajuste do modelo e para isto, vamos selecionar os dados referentes a uma única amostra.

```
> 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
```

No gráfico à esquerda da Figura 30.1 visualizamos os dados de umidade *versus* pressão aplicada na amostra.

```
> with(cra30, plot(u ~ pot, xlab=expression(Psi[m]), ylab=expression(theta), ylim=c(0,0.35)))
```

Uma melhor visualização é obtida utilizando-se no eixo horizontal o logaritmo (base 10) dos valores das pressões aplicadas conforme mostrado no gráfico à direita.



```
> with(cra30, plot(u ~ log10(pot), xlab=expression(log[10](Psi[m])), ylab=expression(theta), ylim=c(0,0
```

Portanto, os dados nas colunas `u` e `pot` do objeto de dados correspondem, respectivamente, à θ e Ψ_m na equação 8. As demais quantidades ($\theta_S, \theta_R, \alpha, n$) são parâmetros (coeficientes) a serem estimados a partir do ajuste do modelo teórico aos dados. Este é um modelo não linear que pode ser ajustado utilizando o método de mínimos quadrados conforme implementado em `nls()`. 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 cujos 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*, uma lista com nomes dos elementos que devem coincidir com os utilizados na declaração do modelo. Há argumentos adicionais para controlar o comportamento do algoritmo, tal como critério de convergência. A documentação de `nls()` fornece mais detalhes.

A escolha dos valores iniciais é crucial e pode influenciar os resultados do ajuste utilizando métodos numéricos, especialmente em exemplos como este com um pequeno número de dados. Os valores iniciais para θ_S e θ_R foram escolhidos inspecionando-se o gráfico e considerando a interpretação destes como valores de saturação e residual de umidade, portanto considerando-se máximos e mínimos assintóticos para a função. A escolha de valores iniciais para os demais parâmetros é menos óbvia. Uma das formas de se obter tais valores é efetuar um ajuste aproximado, visual por tentativa e erro, traçando-se curvas sobre o gráfico dos dados. O comando a seguir ilustra como fazer tal procedimento a partir do gráfico dos dados originais mostrado anteriormente definindo uma expressão para `curve()` com o modelo de van Genuchten. Os valores foram escolhidos após uma série de tentativas.

```
> curve(0.05 + (0.35-0.05)/((1+(0.1*x)^1.3)^(1-1/1.3)), from=0, to=27000, add=T, lty=2)
```

Definidos os valores iniciais prossegue-se com o ajuste do modelo conforme os comandos a seguir.

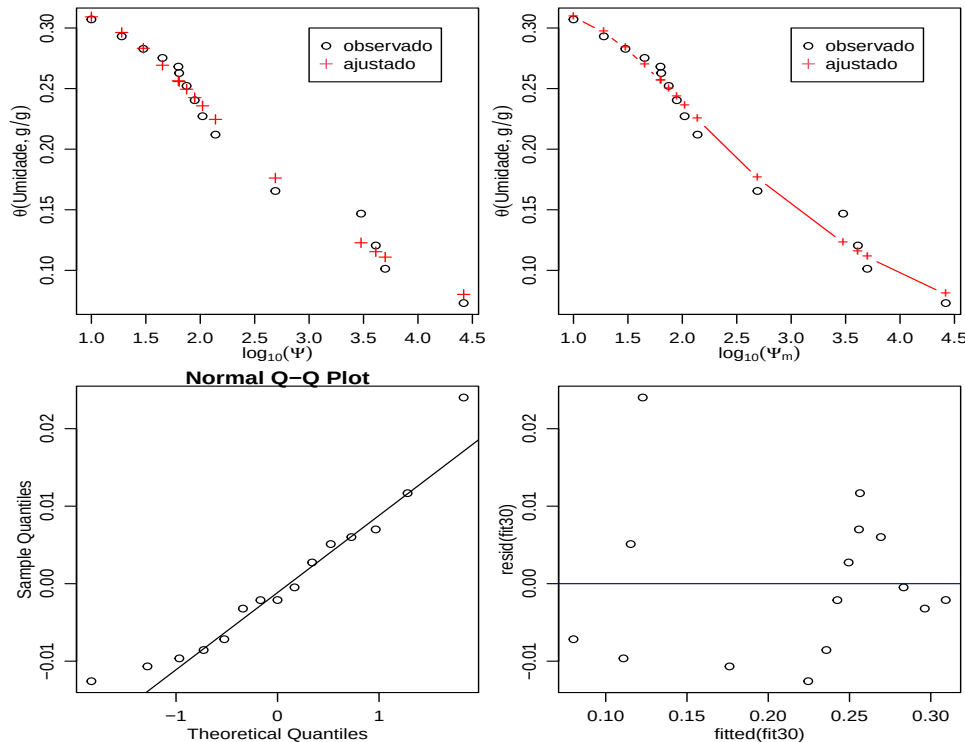
```
> fit30=nls(u~ur+(us-ur)/((1+(alpha*pot)^n)^(1-1/n)), data=cra30,
+   start=list(us=0.35,ur=0.05,alpha=0.1, n=1.3))
> summary(fit30)
```

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

```
Parameters:
```

	Estimate	Std. Error	t value
us	0.324121	0.017744	18.27
ur	0.007082	0.071084	0.10
alpha	0.038780	0.026202	1.48
n	1.211816	0.105207	11.52

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



Number of iterations to convergence: 8
 Achieved convergence tolerance: 7.748e-06

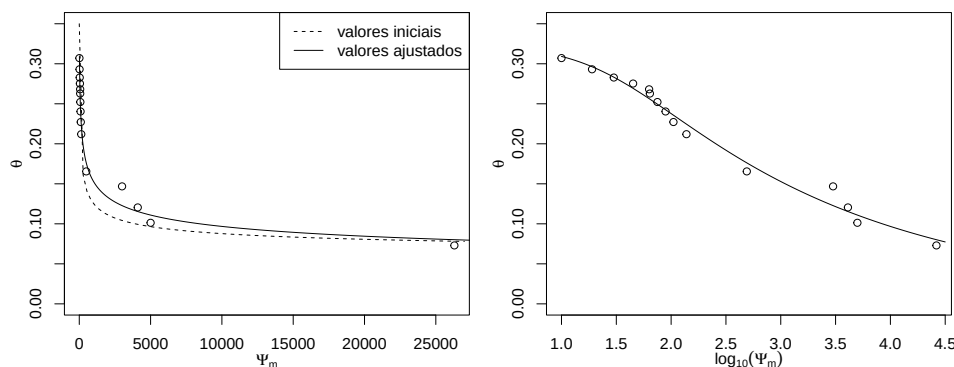
A partir do modelo ajustado pode-se calcular quantidades de interesse. Neste particular exemplo calculamos uma quantidade de interesse prático denotada por S que é um indicador da qualidade física do solo. Quanto maior o valor de S , melhor a sua qualidade física.

```
> S = with(as.list(coef(fit30)),abs((-n*(us-ur)*(((2*n-1)/(n-1))^(1/n-2))))
> S
[1] 0.04097127
```

Os valores preditos são obtidos de forma direta com `fitted(fit30)` ou `predict(fit30)`. Para visualização e avaliação do modelo ajustado podemos fazer diferentes gráficos. A Figura 30.1 mostra os pontos ajustados no gráfico da esquerda, e a união destes pontos no gráfico da direita. Gráficos de resíduos semelhantes aos obtidos para avaliar ajuste de modelos lineares podem e devem também ser investigados em uma análise. Neste exemplo mostramos o *qq-plot* dos resíduos e o gráfico dos resíduos *versus* valores preditos.

```
> 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, 0.30, 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, 0.30, c("observado","ajustado"),pch=c(1,3), col=c(1,2))
> rs <- resid(fit30); qqnorm(rs); qqline(rs)
> plot(fitted(fit30),resid(fit30)); abline(h=0, col="navy")
```

Para obter uma melhor visualização do modelo ajustado pode-se obter valores na curva ajustada não apenas nos pontos observados, mas em uma sequência de valores ao longo do gráfico como ilustrado a seguir. A Figura 30.1 mostra à esquerda o modelo definido pelos



valores iniciais e o modelo ajustado, na escala original. Note que neste exemplo em geral prefere-se a visualização na escala logarítmica do potencial conforme gráfico da direita. As curvas com os modelos ajustados a serem desenhadas sobre o gráfico dos dados são obtidas com os comandos a seguir.

```
> pp <- 10~seq(1, 4.5, l=201)
> lines(pp, predict(fit30, list(pot=pp)))
> legend("topright", c("valores iniciais", "valores ajustados"), lty=2:1)
```

Comentários: 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. Para isto é necessário redefinir a especificação do modelo e utilizar o argumento `method="plinear"` em `nls()`. Neste exemplo em particular pode-se considerar 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. Por outro lado, em geral repara-me-trizações podem mudar a interpretação de alguns parâmetros de modelo. Finalmente cuidados usuais com ajuste de modelos utilizando métodos iterativos devem ser observados, tais como sensibilidade a valores iniciais e verificação de convergência do algoritmo numérico.

30.2 Ajustando modelo a vários conjuntos de dados

Vamos considerar uma situação comum na prática em que tem-se várias amostras para as quais deseja-se fazer ajustes individuais como o ilustrado anteriormente. É portanto conveniente que isto seja feito de forma automática, sem a necessidade de repetir os passos acima a cada ajuste. Neste exemplo vamos considerar duas amostras, mas o procedimento demonstrado a seguir é geral e funcionará igualmente para um maior número de amostras.

Serão mostradas duas soluções. Nesta seção o ajuste é feito para cada amostra individualmente automatizando várias chamadas à função `nls()` através de `lapply()` emulando o comportamento das várias chamadas em um *loop*. Na próxima seção será mostrado como obter todos os ajustes com uma única chamada à `nls()`. Ilustramos ambos os casos porque a forma mais adequada vai depender da situação em questão e dos objetivos da análise.

Começamos definindo uma função que contém uma chamada à `nls()` como acima. Estamos incluindo na função um argumento `ini` para passar valores iniciais. Caso não fornecidos a função 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=0.30,ur=0.02,alpha=0.05,n=1.3))
+   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 a segunda simplifica o objeto resultante se possível. Por exemplo, quando extraíndo coeficientes a função retorna uma matriz 4×2 , já que para cada uma das duas amostras são extraídos quatro coeficientes.

```
> lapply(allfits, summary)
```

```
$`30`
```

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

```
Parameters:
```

	Estimate	Std. Error	t value
us	0.324120	0.017744	18.27
ur	0.007082	0.071084	0.10
alpha	0.038780	0.026202	1.48
n	1.211816	0.105207	11.52

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

```
Number of iterations to convergence: 6
```

```
Achieved convergence tolerance: 8.286e-06
```

```
$`41`
```

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

```
Parameters:
```

	Estimate	Std. Error	t value
us	0.243148	0.009446	25.741
ur	-0.122402	0.171615	-0.713
alpha	0.035928	0.022324	1.609
n	1.113320	0.079473	14.009

```
Residual standard error: 0.006207 on 9 degrees of freedom
```

```
Number of iterations to convergence: 7
```

```
Achieved convergence tolerance: 9.37e-06
```

```
> lapply(allfits, coef)
```

```
$`30`
```

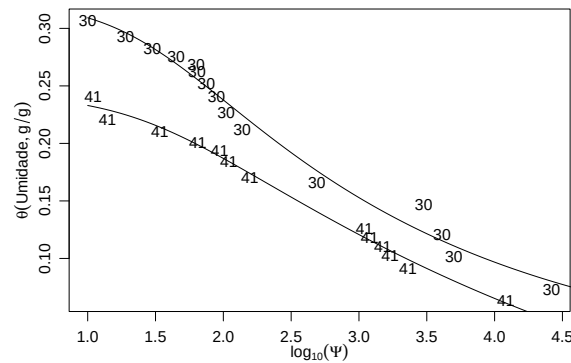
	us	ur	alpha	n
	0.324120351	0.007081962	0.038779951	1.211815750

```
$`41`
```

	us	ur	alpha	n
	0.24314783	-0.12240199	0.03592827	1.11332043

```
> sapply(allfits, coef)
```

	30	41
us	0.324120351	0.24314783
ur	0.007081962	-0.12240199
alpha	0.038779951	0.03592827
n	1.211815750	1.11332043



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 amostras. Para isto simplesmente definimos uma função que recebe o modelo ajustado e usa os coeficientes 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
0.04097126 0.02950320
```

Finalmente, para encerrar este exemplo, vamos mostrar uma possível forma de combinar a visualização dos ajustes 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))
```

30.3 Combinando ajustes

Na seção anterior obtivemos o ajuste para cada amostra separadamente fazendo várias chamadas à função `nls()`. Isto pode ser adequado quando deseja-se de fato ajustes individuais e se, por um lado são efetuadas várias chamadas à função, por outro o número de dados em cada uma delas é pequeno. Uma forma alternativa de obter parâmetros para cada amostra, possivelmente mais eficiente que a mostrada anteriormente, é discutida a seguir.

Nesta seção vamos considerar fazer todos os ajustes de uma só vez, isto é em uma única chamada à `nls()` que portanto vai utilizar todos os dados de todas as amostras. Além do aspecto computacional, isto pode ser interessante pois permite comparar e testar hipóteses para escolha entre diferentes modelos alternativos para explicar os dados. A seguir exemplificamos tal procedimento iniciando com um modelo para cada amostra e comparando com um modelo que assume que os parâmetros (α , n) são comuns entre as amostras. Neste caso interpreta-se que

cada amostra informa sobre os respectivos valores para (θ_S, θ_R) enquanto todas as amostras conjuntamente informam sobre (α, n) . Após ajustar os modelos *candidatos* podemos fazer uma comparação formal dos ajustes através de `anova()`, o que não seria possível ajustando os modelos separadamente como mostrado na seção anterior.

Os dois ajustes são mostrados a seguir. O seletor `[]` é usado para indicar que os dados são tratados em grupos definidos por `am`. No caso do modelo com parâmetros distintos informamos oito valores iniciais para os parâmetros.

```
> form0 <- formula(u~ur[am]+(us[am]-ur[am])*(1/(1+(alpha[am]*pot)^n[am]))^(1-1/n[am]))
> mod0 <- nls(form0, cra,
+             start = list(us = c(0.3,0.3), ur = c(0,0),
+             alpha = c(0.04, 0.04), n = c(1.25, 1.25)))
> mod0
Nonlinear regression model
model: u ~ ur[am] + (us[am] - ur[am]) * (1/(1 + (alpha[am] * pot)^n[am]))^(1 - 1/n[am])
data: cra
      us1      us2      ur1      ur2    alpha1    alpha2      n1      n2
0.324120  0.243148  0.007085 -0.122402  0.038779  0.035928  1.211819  1.113321
residual sum-of-squares: 0.001688

Number of iterations to convergence: 6
Achieved convergence tolerance: 4.574e-06
```

Para ajuste assumindo valores comuns para os parâmetros α e n não utilizamos o indicador de grupos para estes parâmetros e informamos apenas um valor inicial para cada um deles.

```
> form1 <- formula(u ~ ur[am] + (us[am] - ur[am])*(1/(1+(alpha*pot)^n))^(1-1/n))
> mod1 <- nls(form1, cra,
+             start = list(us=c(0.3,0.3), ur = c(0,0), alpha=0.04, n = 1.25))
> mod1
Nonlinear regression model
model: u ~ ur[am] + (us[am] - ur[am]) * (1/(1 + (alpha * pot)^n))^(1 - 1/n)
data: cra
      us1      us2      ur1      ur2    alpha      n
0.32106  0.24870 -0.03056 -0.02759  0.03994  1.17195
residual sum-of-squares: 0.001846

Number of iterations to convergence: 5
Achieved convergence tolerance: 1.976e-06
```

Neste exemplo temos então um modelo inicial com oito e outro mais parcimonioso com apenas seis parâmetros e utilizamos um teste formal para orientar a escolha de modelo, que neste caso indica que o modelo mais parcimonioso com parâmetros comuns explica os dados satisfatoriamente.

```
> anova(mod1,mod0)
Analysis of Variance Table

Model 1: u ~ ur[am] + (us[am] - ur[am]) * (1/(1 + (alpha * pot)^n))^(1 - 1/n)
Model 2: u ~ ur[am] + (us[am] - ur[am]) * (1/(1 + (alpha[am] * pot)^n[am]))^(1 - 1/n[am])
  Res.Df Res.Sum Sq Df    Sum Sq F value Pr(>F)
1      22  0.0018462
2      20  0.0016884  2 0.00015786  0.935 0.4091
```

Carregando pacotes exigidos: sp

31 Classes para dados espaciais: o pacote sp

Pacotes e funções para análise de dados espaciais começaram a surgir desde o início do projeto R. Isto em parte se deve ao R fornecer um ambiente adequado para disponibilização de novas propostas de análise e formas de implementação de métodos, combinado ao fato de que a área de estatística espacial estava na época (e ainda está!) em franco desenvolvimento. As implementações procuravam tanto dotar o ambiente do R de funcionalidades usuais de estatística espacial, como implementar novas propostas metodológicas e ainda fornecer interface com outros sistemas tais como os SIG's (Sistemas de Informação Geográfica) e bancos de dados, especialmente os estruturados espacialmente.

A característica central de dados espaciais é o fato das informações possuírem duas estruturas básicas e interrelacionadas: *geometrias* e *atributos*. Tais estruturas são distintas não somente por se referirem a elementos conceituais diferentes, localização e características da localização, mas também, e talvez principalmente, por nem sempre poderem ser representadas de forma simples, como uma única tabela de dados.

Vejamos dois exemplos. Num primeiro caso vamos imaginar que dois atributos (variáveis) sejam medidos em n pontos, sendo cada um destes identificado por um par de coordenadas. Por exemplo, poderíamos ter os atributos *precipitação* e *temperatura* máxima diária registrada em n estações meteorológicas. Neste caso, poderíamos facilmente estruturar os dados em uma matriz de dimensão $n \times 4$, onde as quatro colunas seriam referentes ao par de coordenadas (geometria) e às duas variáveis (atributos). Num segundo caso vamos imaginar agora tais atributos medidos em cada município de um estado, onde os municípios são identificados por polígonos que definem suas fronteiras. Neste caso, diferentemente do anterior, não temos como combinar a geometria (polígonos) e os atributos (variáveis temperatura e umidade) em uma estrutura simples de dados como a matriz do exemplo anterior.

Tais exemplos reforçam a ideia que dados espaciais (e espaço-temporais) precisam ter representações que acomodem o tratamento de geometrias e atributos. A esta discussão soma-se o fato que a área de dados espaciais é tipicamente dividida em subáreas que dependem do formato específico dos dados e modelos a serem considerados, sejam de variação espacial discreta (dados de área), contínua (geoestatística) ou processos pontuais. Outras divisões e sub-divisões são ainda possíveis mas vamos nos ater nesta discussão a estas três.

Desta forma, na implementação dos pacotes de estatística espacial no R, os autores seguiram diferentes estratégias dependendo do tipo de dado contemplado pelo pacote bem como de suas preferências pessoais. Com o crescimento do número de pacotes e formas alternativas de representar os dados espaciais, em particular suas geometrias, acabou-se por criar uma verdadeira *torre de Babel* das representações de dados espaciais.

Neste contexto, começou-se a discutir a possibilidade de criação de uma estrutura comum e geral, que servisse para os diferentes formatos de dados espaciais. Tal ideia acabou se materializando no pacote **sp** de Roger Bivand e Edzer Pebesma, uma excelente, criativa e bem estruturada proposta baseada em objetos e classes do tipo **S4** do R. A implementação foi inicialmente descrita em um artigo dos autores na R News (2005, vol.5(2), p.9-13) e também na seção de estatística espacial do R (CRAN Spatial Task View) e, mais recentemente, no livro *Applied Spatial Data Analysis with R* que conta ainda com a co-autoria de Virgilio Gómez-Rubio. Detalhes podem ainda ser encontrados no *vignette* que acompanha o pacote **sp**.

Embora não adotado universalmente, vários pacotes de estatística espacial agora aderem a este formato de dados e/ou possuem formas de converter as estruturas de dados de suas

representações específicas para o formato definido pelo pacote **sp**. A tendência é que a estrutura definida pelo **sp** seja largamente adotada, especialmente por novos pacotes, por toda a flexibilidade que traz no tratamento e representação de dados espaciais.

Nesta seção vamos apresentar de maneira informal e através de exemplos simples ideias introdutórias sobre a representação e estrutura de dados espaciais definidas pelo pacote **sp**. Entretanto, note-se que os recursos de tais classes vão muito além dos exemplos apresentados aqui. As referências mencionadas anteriormente são o guia definitivo para informações mais precisas e detalhadas sobre a potencialidade de tal representação e o texto apresentado a seguir visa somente facilitar o estudo destes materiais e não substituí-los!

O passo inicial para seguir estas notas é instalar e carregar o pacote **sp**. Este pacote tem dependências de vários outros e portanto, usa-se o argumento `dep=TRUE` que faz com que todos os demais pacotes necessários sejam também instalados.

```
> install.packages("sp", dep=TRUE)
> require(sp)
> #vignette("sp", package="sp")
```

31.1 Conceitos introdutórios e classes para pontos esparsos

Vamos iniciar considerando um exemplo simples onde os dados consistem de cinco localizações nas quais foram medidos os valores de duas variáveis conforme ilustrado na Figura 78. Os dados serão inicialmente armazenados na forma de uma matriz de coordenadas e um *data-frame* com as variáveis, uma das estruturas básicas de dados no R, comumente utilizada para armazenar estruturas de dados na forma de linhas (indivíduos) e colunas (variáveis). Por motivos de apresentação neste material vamos considerar inicialmente as geometrias e atributos separadamente. Neste caso poderíamos ter ainda optado por incluir as coordenadas no *data-frame*, adicionando então duas colunas.

```
> cord <- cbind(cx = c(1, 3, 6, 2, 5), cy = c(4, 2, 5, 6, 1))
> DF <- data.frame(var1 = c(23, 26, 18, 25, 30), var2 = c(63, 76, 81, 59, 80))
> DF
  var1 var2
1   23   63
2   26   76
3   18   81
4   25   59
5   30   80
> dim(DF)
[1] 5 2
> SPDF <- cbind(cord, DF)
> dim(SPDF)
[1] 5 4
```

A estrutura do *data-frame*, embora suficiente para anotar todas as informações, não distingue explicitamente as geometrias e atributos, cabendo ao usuário saber a que se refere cada uma das colunas. Na definição de classes do **sp**, este tipo de dado, com geometria dada por pontos esparsos na região, é representada por um objeto do tipo **SpatialPointsDataFrame**. O nome é praticamente autoexplicativo indicando que o dado é um *data-frame* de atributos ligados a pontos. Existem várias formas de se construir um objeto do tipo **SpatialPointsDataFrame** e vamos ver três delas aqui, a começar pela mais simples dada a seguir utilizando a função `coordinates()`. Esta função converte um *data-frame* em um **SpatialPointsDataFrame** simplesmente indicando quais os nomes das colunas em que estão armazenadas as coordenadas. Note nos comandos como fazer tal conversão e como os objetos desta classe diferem entre si.

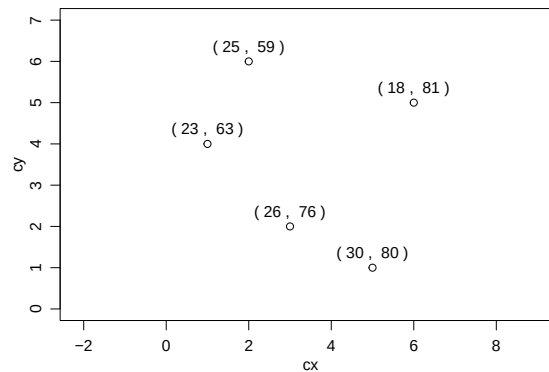


Figura 78: Exemplo hipotético com dois atributos medidos em cinco localizações identificadas por pares de coordenadas. Os valores dos atributos são indicados dentro dos parêntesis.

Após a conversão as coordenadas deixam de fazer parte do data-frame, que armazena agora somente as variáveis, e passam a ser somente uma informação a ele associada.

```
> class(SPDF)
[1] "data.frame"
> dim(SPDF)
[1] 5 4
> coordinates(SPDF) <- c("cx", "cy")
> class(SPDF)
[1] "SpatialPointsDataFrame"
attr(,"package")
[1] "sp"
> dim(SPDF)
[1] 5 2
> SPDF
  coordinates var1 var2
1      (1, 4)  23  63
2      (3, 2)  26  76
3      (6, 5)  18  81
4      (2, 6)  25  59
5      (5, 1)  30  80
```

Todo objeto desta classe possui *atributos* específicos que são criados automaticamente podendo ainda ser modificados pelo usuário. Vamos descrevê-los por grupos cuja divisão ficará mais clara posteriormente. Os dois primeiros são as coordenadas do menor retângulo que envolve as localizações dos dados (**bbox**), e uma *string* que define a projeção dos dados (**proj4string**), ou seja, como o dado está georreferenciado em termos do tipo de coordenadas, podendo esta ser NA. A informação de como esta *string* é escrita segue um padrão definido pelo projeto **proj4** e pode ser alterada usando a função **CRS()**. Estes dois argumentos formam um primeiro grupo devido ao fato de que todo objeto **Spatial*** definido pelo pacote **sp** possui tais atributos. Os demais atributos são específicos da classe **SpatialPointsDataFrame**. Como estes objetos são construídos segundo o padrão **S4** da linguagem R, os atributos são usualmente chamados de *slots*. A listagem de *slots* de uma classe e a forma de extrair cada particular *slot* são ilustradas nos comandos a seguir.

```
> class(SPDF)
[1] "SpatialPointsDataFrame"
attr(,"package")
[1] "sp"
```

```
> getSlots("SpatialPointsDataFrame")
      data  coords.nrs      coords      bbox  proj4string
"data.frame"  "numeric"    "matrix"    "matrix"    "CRS"
> slot(SPDF, "bbox")
      min max
cx   1   6
cy   1   6
> slot(SPDF, "data")
      var1 var2
1     23   63
2     26   76
3     18   81
4     25   59
5     30   80
```

Uma outra forma de criar um objeto que represente dados com geometria dada por pontos esparsos é usar a função `SpatialPointsDataFrame()` que recebe coordenadas e atributos separadamente em dois argumentos obrigatórios. O resultado apenas difere do retornado pela função `coordinates()` no `slot coords.nrs`, que para esta última registra as colunas do data-frame original que foram indicadas como coordenadas.

```
> SPDF1 <- SpatialPointsDataFrame(coords=cord, data=DF)
> all.equal(SPDF, SPDF1)
[1] "Attributes: < Component \"coords.nrs\": Numeric: lengths (2, 0) differ >"
> slot(SPDF, "coords.nrs")
[1] 1 2
> slot(SPDF1, "coords.nrs")
numeric(0)
```

Os demais atributos/slots do objeto são definidos automaticamente mas podem ser modificados.

```
> slot(SPDF1, "bbox") <- cbind(min=c(0,0), max=c(7,7))
> bbox(SPDF1)
      min max
[1,]   0   7
[2,]   0   7
```

A função possui ainda outros argumentos opcionais já comentados anteriormente, exceto por `match.ID` que permite que os dois objetos sejam pareados pelos nomes das suas linhas (`rownames`), permitindo portanto, coordenadas e atributos em ordens diferentes, desde que identificadas pelo mesmo nome da linha. No exemplo a seguir alteramos a ordem dos elementos do data-frame de atributos para ilustrar o pareamento.

```
> args(SpatialPointsDataFrame)
function (coords, data, coords.nrs = numeric(0), proj4string = CRS(as.character(NA)),
      match.ID, bbox = NULL)
NULL
> #function (coords, data, coords.nrs = numeric(0), proj4string = CRS(as.character(NA)),
> #      match.ID = TRUE, bbox = NULL)
> DF1 <- DF[c(3,1,5,2,4),]
> DF1
      var1 var2
3     18   81
1     23   63
5     30   80
2     26   76
4     25   59
```

```
> SPDF2 <- SpatialPointsDataFrame(coords=cord, data=DF, bbox=cbind(min=c(0,0), max=c(7,7)))
> all.equal(SPDF1, SPDF2)
[1] TRUE
```

Até aqui vimos que `coordinates()` e `SpatialPointsDataFrame()` criam diretamente objetos da referida classe. Vamos agora examinar com mais detalhe toda a concepção de classes definida no **sp** que tem uma estrutura hierárquica, começando por classes e construtores de objetos mais gerais que se tornam mais detalhados a cada nível e onde cada classe herda os atributos da classe superior mais geral.

A classe mais geral é **Spatial** e um objeto desta classe tem apenas duas informações (*slots*): o retângulo envolvente (*bbox* - *bounding box*) e a informação do tipo de projeção (**proj4string**). Este objeto então simplesmente define em que região estarão os dados e por qual sistema de coordenadas estarão referenciados, sendo que esta última pode ser um "NA" que em SIG's corresponde a dados sem projeção (*null projection*). O retângulo envolvente pode ser calculado a partir das coordenadas dos dados (como feito automaticamente pelas funções já vistas), ou definido arbitrariamente pelo usuário, podendo ainda ser alterado depois de definido. Deve conter nas linhas a dimensão das coordenadas, e nas colunas os valores mínimos e máximos para cada uma. A *string* que informa sobre a projeção deve ser definida usando a função `CRS()` pois esta irá validar a definição e sintaxe. A função ainda verifica se os valores passados em *bbox* são compatíveis com a projeção informada, retornando erro caso sejam incompatíveis. Métodos são implementados para extrair os elementos do objeto.

```
> getSlots("Spatial")
      bbox proj4string
"matrix"      "CRS"

> bb <- t(apply(cord, 2, range))
> colnames(bb) <- c("min", "max")
> S <- Spatial(bbox = bb, proj4string=CRS(projargs=as.character(NA)))
> S

An object of class "Spatial"
Slot "bbox":
      min max
cx    1    6
cy    1    6

Slot "proj4string":
Coordinate Reference System:
Deprecated Proj.4 representation: NA
> bbox(S)
      min max
cx    1    6
cy    1    6

> #slot(S, "bbox") <- cbind(c(0,0), c(7,7))
> slot(S, "bbox") <- cbind(min=c(0,0), max=c(7,7))
> bbox(S)
      min max
[1,]    0    7
[2,]    0    7

> proj4string(S)
[1] NA
```

A classe **Spatial** possui três subclasses: **SpatialPoints**, **SpatialLines** e **SpatialPolygons**. Estas subclasses definem, como os nomes sugerem, o tipo de geometria dos dados. Seguindo nosso exemplo vamos criar um objeto da classe **SpatialPoints**

que estende a classe `Spatial` adicionando um *slot* `coords` que armazena as coordenadas de um conjunto de pontos. Assim como no exemplo anterior vamos examinar os *slots* e tipos de objetos que estes recebem com a função `getSlots()` aplicada ao nome da classe. Na saída desta função fica claro que a classe `SpatialPoints` herda os atributos de `Spatial`. A função construtora tem `coords` como argumento obrigatório e as herdadas da classe `Spatial`, `bbox` e `proj4string` como opcionais. É ainda importante notar que vários *métodos* usuais já são definidos para este nível de classes tais como `summary()`, `plot()`, entre outros. É ainda possível fazer seleção de elementos.

```
> getSlots("SpatialPoints")
      coords      bbox proj4string
"matrix"      "matrix"      "CRS"

> args(SpatialPoints)
function (coords, proj4string = CRS(as.character(NA)), bbox = NULL)
NULL

> row.names(cord) <- 1:nrow(cord)
> SP <- SpatialPoints(coords = cord)
> # o seguinte nao funciona se nao tem colnames, entao nao deveria aceitar acima sem colnames!
> SP <- SpatialPoints(coords = cord, bbox=bbox(S))
> SP

SpatialPoints:
  cx cy
1  1  4
2  3  2
3  6  5
4  2  6
5  5  1
Coordinate Reference System (CRS) arguments: NA

> summary(SP)

Object of class SpatialPoints
Coordinates:
      min max
[1,]    0   7
[2,]    0   7
Is projected: NA
proj4string : [NA]
Number of points: 5

> ind <- coordinates(SP)[,2] < 3
> SP[ind,]

SpatialPoints:
  cx cy
2  3  2
5  5  1
Coordinate Reference System (CRS) arguments: NA
```

Seguindo esta estrutura de classe e subclasses podemos revisitar a classe `SpatialPointsDataFrame` como sendo uma sub-sub-classe de `Spatial` que estende `SpatialPoints` acomodando uma matriz de atributos, que por default é pareada com as coordenadas pelos nomes das linhas (*rownames*), daí porque na criação do objeto `SpatialPoints` asseguramos a definição dos nomes de linhas. Já vimos acima como um objeto desta classe é criado mas note-se que ele poderia ser criado ainda a partir de um objeto `SpatialPoints` com a simples adição do data-frame dos atributos. Os extratores, métodos e seletores continuam válidos.

```
> getSlots("SpatialPointsDataFrame")
```

```

      data   coords.nrs      coords      bbox   proj4string
"data.frame"  "numeric"    "matrix"    "matrix"    "CRS"
> SPDF3 <- SpatialPointsDataFrame(SP, DF)
> all.equal(SPDF1, SPDF3)
[1] "Attributes: < Component \"coords\": Attributes: < Component \"dimnames\": Component 1: alvo é NULL"
> summary(SPDF3)
Object of class SpatialPointsDataFrame
Coordinates:
      min max
[1,]   0   7
[2,]   0   7
Is projected: NA
proj4string : [NA]
Number of points: 5
Data attributes:
      var1      var2
Min.   :18.0  Min.   :59.0
1st Qu.:23.0  1st Qu.:63.0
Median :25.0  Median :76.0
Mean   :24.4  Mean   :71.8
3rd Qu.:26.0  3rd Qu.:80.0
Max.   :30.0  Max.   :81.0
> bbox(SPDF3)
      min max
[1,]   0   7
[2,]   0   7
> SPDF3[ind,]
      coordinates var1 var2
2      (3, 2)    26   76
5      (5, 1)    30   80

```

Neste ponto podemos revisar a estrutura das classes verificando a saída de `getClass()` que informa os *slots* de cada classe ou subclasse, qual(is) a(s) class(es) por ela estendida – se alguma, bem como quais as subclasses definidas para esta classe em vários níveis.

```

> getClass("Spatial")
Class "Spatial" [package "sp"]

Slots:

Name:      bbox proj4string
Class:     matrix      CRS

Known Subclasses:
Class "SpatialPoints", directly
Class "SpatialMultiPoints", directly
Class "SpatialGrid", directly
Class "SpatialLines", directly
Class "SpatialPolygons", directly
Class "SpatialPointsDataFrame", by class "SpatialPoints", distance 2
Class "SpatialPixels", by class "SpatialPoints", distance 2
Class "SpatialMultiPointsDataFrame", by class "SpatialMultiPoints", distance 2
Class "SpatialGridDataFrame", by class "SpatialGrid", distance 2
Class "SpatialLinesDataFrame", by class "SpatialLines", distance 2
Class "SpatialPixelsDataFrame", by class "SpatialPoints", distance 3
Class "SpatialPolygonsDataFrame", by class "SpatialPolygons", distance 2
> getClass("SpatialPoints")

```

```

Class "SpatialPoints" [package "sp"]

Slots:

Name:      coords      bbox proj4string
Class:     matrix      matrix      CRS

Extends: "Spatial"

Known Subclasses:
Class "SpatialPointsDataFrame", directly
Class "SpatialPixels", directly
Class "SpatialPixelsDataFrame", by class "SpatialPixels", distance 2
> getClass("SpatialPointsDataFrame")
Class "SpatialPointsDataFrame" [package "sp"]

Slots:

Name:      data coords.nrs      coords      bbox proj4string
Class:     data.frame      numeric      matrix      matrix      CRS

Extends:
Class "SpatialPoints", directly
Class "Spatial", by class "SpatialPoints", distance 2

Known Subclasses:
Class "SpatialPixelsDataFrame", directly, with explicit coerce

```

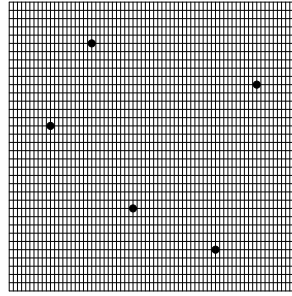
31.2 Pontos em malha regular: grid e pixel

`SpatialGrid` e `SpatialPixels` são representações de pontos arranjados de forma regular numa região, tais como modelos de elevação digital, imagens (por ex. de satélite), malhas de interpolação de pontos, entre outras. Tais representações são comuns em sensoriamento remoto e representações do tipo *raster* em SIG's. Todo o conjunto de pontos fica definido a partir de apenas algumas informações básicas como origem e espaçamento, o que permite que os pontos de toda a malha sejam tratados de uma só vez ao invés de cada ponto individualmente. Estas classes estendem `SpatialPoints` de forma a registrar e utilizar as informações sobre o arranjo regular das localizações, o que é feito com `GridTopology` que define as células da malha de pontos. Como exemplo vamos criar uma malha retangular com espaçamento de $0,1 \times 0,2$ sobre a área do exemplo anterior. As informações necessárias são o centro da primeira célula da malha, o tamanho e número de células em cada dimensão. A combinação da classe `GridTopology` com os elementos de `Spatial` gera a subclasse `SpatialGrid`. A Figura 31.2 mostra a sobreposição das localizações dos dados originais e a malha de pontos que cobre a área no espaçamento especificado.

```

> bbox(SPDF3)
      min max
[1,]   0   7
[2,]   0   7
> espac <- c(0.1, 0.2)
> centro1 <- bbox(SPDF3)[,1] + espac/2
> centro1
[1] 0.05 0.10

```



```
> nums <- ceiling(diff(t(bbox(SPDF3)))/espac)
> GT <- GridTopology(cellcentre.offset = centro1, cellsize = espac, cells.dim=nums)
> SG <- SpatialGrid(GT)
> getClass("SpatialGrid")
Class "SpatialGrid" [package "sp"]
```

Slots:

```
Name:      grid      bbox proj4string
Class: GridTopology matrix      CRS
```

Extends: "Spatial"

Known Subclasses: "SpatialGridDataFrame"

```
> plot(SPDF3, pch=19)
> plot(SG, cex=0.4, add=T)
```

Neste exemplo definimos a malha a partir da dimensão da área. Entretanto, isto não é compulsório, podendo a malha ser criada de outras formas e/ou importada de algum outro objeto ou formato e neste caso o retângulo envolvente (*bounding box*) é criado automaticamente. Note ainda que no exemplo continuamos usando o dado sem projeção ou qualquer tipo de sistema de coordenadas.

A extensão natural é dada pela classe `SpatialGridDataFrame` que associa à malha um data-frame com atributos associados a cada um dos pontos.

```
> getClass("SpatialGridDataFrame")
Class "SpatialGridDataFrame" [package "sp"]
```

Slots:

```
Name:      data      grid      bbox proj4string
Class: data.frame GridTopology matrix      CRS
```

Extends:

```
Class "SpatialGrid", directly
Class "Spatial", by class "SpatialGrid", distance 2
```

O formato de grid (raster) apresenta algumas limitações em certas circunstâncias. Por exemplo se a área tem muitos recortes e/ou várias células não possuem atributos, os valores correspondentes devem ser indicados como NA. Por exemplo, em um modelo de elevação digital de terreno poderiam haver pontos abaixo da superfície da água, ou ainda a área de estudo pode ser bem recortada, levando a um formato bastante irregular com muitas partes dentro de bbox porém fora da área. Como os grids tipicamente tem alta resolução, isto faz com que um grande volume de memória seja utilizado sem necessidade. Além disto, em certas

situações pode-se desejar exportar dados para aplicativos externos em forma de coordenadas de pontos. Nestes casos pode-se usar a representação alternativa de `SpatialPixels` que guarda informações apenas dos pontos de interesse, assim como em `SpatialPoints`, porém guardando também a informação de o que se tem é um subconjunto de uma malha, como definida em `SpatialGrid`.

31.3 Classe para linhas e polígonos

Vimos até aqui que a subclasse `SpatialPoints` e as subclasses derivadas dela estendem a classe `Spatial` para acomodar coordenadas de uma geometria de pontos. As outras geometrias espaciais são *linhas* e *polígonos* que são tratadas pelas classes `SpatialLines` e `SpatialPolygons`, respectivamente. A representação destas geometrias no `sp` é feita através de um conjunto sequencial de pontos, sendo que outras representações possíveis existem em ambientes de SIG. Estas duas geometrias são semelhantes em sua forma, sendo o polígono representado por uma linha fechada, ou seja, uma linha onde o primeiro e último pontos são iguais. Assim como em `SpatialPoints`, ambas subclasses são estendidas pela adição de atributos em subsubclasses `SpatialLinesDataFrame` e `SpatialPolygonsDataFrame`.

```
> getClass("SpatialLines")
Class "SpatialLines" [package "sp"]

Slots:

Name:      lines      bbox proj4string
Class:     list       matrix      CRS

Extends: "Spatial"

Known Subclasses: "SpatialLinesDataFrame"
> getClass("SpatialPolygons")
Class "SpatialPolygons" [package "sp"]

Slots:

Name:      polygons  plotOrder      bbox proj4string
Class:     list      integer      matrix      CRS

Extends: "Spatial"

Known Subclasses:
Class "SpatialPolygonsDataFrame", directly, with explicit coerce
```

Assim como um objeto em `SpatialPoints` é definido por um conjunto de pontos, analogamente em `SpatialLines` é definido por um conjunto de linhas e em `SpatialPolygons` é definido por um conjunto de polígonos. Pontos são definidos simplesmente por um par de coordenadas, enquanto que linhas e polígonos são definidos por um conjunto de pontos com uma certa estrutura. Desta forma, criaram-se as funções `Line` e `Polygon` para se especificar estes elementos. A seguir vamos utilizar um exemplo envolvendo três polígonos disponíveis em um conjunto de dados do pacote **geoR**. Para nos concentrar apenas nos polígonos vamos extraí-los do objeto de dados original e armazená-los em uma lista onde cada elemento é uma matriz.

```
> require(geoR)
Carregando pacotes exigidos: geoR
```

```
-----
Analysis of Geostatistical Data
For an Introduction to geoR go to http://www.leg.ufpr.br/geoR
geoR version 1.9-6 (built on 2025-08-29) is now loaded
-----
```

```
> data(ca20)
> areas <- ca20[c("reg1","reg2","reg3")]
> areas
```

```
$reg1
  east north
1 5590 5690
2 5340 5800
3 5220 5700
4 5250 5370
5 5350 5370
6 5450 5500
7 5510 5600
8 5590 5690
```

```
$reg2
  east north
1 5990 5100
2 5590 5300
3 5350 5370
4 5450 5500
5 5510 5600
6 5590 5690
7 5800 5690
8 5990 5690
```

```
$reg3
  east north
1 5990 5100
2 5590 5300
3 5350 5370
4 5150 5370
5 4920 5000
6 4920 4900
7 5150 4920
8 5350 4900
9 5590 4800
10 5780 4800
```

Para construir um `SpatialPolygon`, o primeiro passo é transformar cada matriz que define um polígono em um objeto da classe `Polygon`, que verifica se o polígono está bem definido, por exemplo, se a coordenada do último ponto coincide com a do primeiro. A estrutura de um objeto da classe `Polygon` inclui ainda o rótulo atribuído ao polígono (`labpt`) que é dado pelo seu centroide, a sua área, e informações sobre se ele é interno ou externo (`hole` e `ringDir`).

```
> getClass("Polygon")
Class "Polygon" [package "sp"]

Slots:

Name:   labpt   area   hole ringDir   coords
Class: numeric numeric logical integer matrix

Extends: "Line"
> Polygon(areas$reg1)
```

An object of class "Polygon"

Slot "labpt":

[1] 5364.879 5596.530

Slot "area":

[1] 95100

Slot "hole":

[1] TRUE

Slot "ringDir":

[1] -1

Slot "coords":

```
  east north
1 5590  5690
2 5340  5800
3 5220  5700
4 5250  5370
5 5350  5370
6 5450  5500
7 5510  5600
8 5590  5690
```

```
> #sapply(areas, function(x) Polygon(x))
```

```
> sapply(areas, function(x) identical(x[1,,drop=T], x[nrow(x),,drop=T]))
```

```
reg1 reg2 reg3
```

```
TRUE FALSE FALSE
```

```
> areas <- lapply(areas, function(x) {if(identical(x[1,,drop=T], x[nrow(x),,drop=T])) x
+                                     else rbind(x, x[1,])})
```

Normalmente, como neste exemplo, se tem mais de um polígono. Um conjunto de polígonos é agrupado na classe `Polygons` que contém uma lista de objetos válidos da classe `Polygon`. É necessário atribuir identificadores para cada polígono que poderão ser posteriormente utilizados para associá-los com atributos. Finalmente, um ou mais objetos `Polygons` são combinados na forma de lista para compor um objeto da classe `SpatialPolygons`. Neste objeto as informações dos polígonos são não apenas armazenadas mas também combinadas para gerar informações da área como um todo.

```
> getClass("Polygons")
```

```
Class "Polygons" [package "sp"]
```

Slots:

```
Name: Polygons plotOrder labpt ID area
```

```
Class: list integer numeric character numeric
```

```
> POLS <- lapply(1:length(areas), function(x) Polygons(list(Polygon(areas[[x]])), ID=paste("reg", x, sep="_")))
> class(POLS)
```

```
[1] "list"
```

```
> SPol <- SpatialPolygons(POLS)
```

```
> class(SPol)
```

```
[1] "SpatialPolygons"
```

```
attr(,"package")
```

```
[1] "sp"
```

```
> getClass("SpatialPolygons")
```

```
Class "SpatialPolygons" [package "sp"]
```

Slots:

```
Name:      polygons      plotOrder      bbox proj4string
Class:      list         integer      matrix      CRS
```

```
Extends: "Spatial"
```

```
Known Subclasses:
```

```
Class "SpatialPolygonsDataFrame", directly, with explicit coerce
```

```
> bbox(SPol)
```

```
      min  max
x 4920 5990
y 4800 5800
```

Métodos tais como `plot`, `spplot`, `summary` entre outros são implementados de forma usual como os demais objetos da família `Spatial`. Atributos ligados aos polígonos são acoplados para criação de `SpatialPolygonsDataFrame`. O data-frame de atributos deve ter nomes de linhas que permitam o pareamento com os nomes atribuídos aos polígonos.

```
> dadosPol <- data.frame(var1 = c(23, 34, 12), var2 = c("a", "b", "a"))
> row.names(dadosPol) <- paste("reg", 1:3, sep="")
> SPolDF <- SpatialPolygonsDataFrame(SPol, dadosPol)
```

32 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 *Writing R Extensions*.

As instruções 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 produzir ambos: `test.o` e `test.so`

```
$ R CMD SHLIB test.c
$ R
```

Uma vez criado o objeto compartilhado `test.so` (seria um `test.dll` no Windows) vamos usar uma função do R para acessar funções disponibilizadas 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 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('test.so')  
> matern(0.1, 1)  
> matern(seq(0,1,l=11), 1)
```

33 (Re)-direcionando saídas de texto e gráficas

Por "default" o R em uma sessão interativa produz saídas de texto na janela do programa e saídas gráficas em 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.

33.1 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 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!) exemplos são citados a seguir.

- `xtable()` do pacote **xtable** prepara tabelas em $\text{\LaTeX}$
- `HTML()` do pacote **R2HTML** e diversas outras funções deste pacote preparam saídas em HTML
- `latex()` e `html()` do pacote **Hmisc** preparam, respectivamente, saídas em $\text{\LaTeX}$ e HTML.

33.2 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
- `windows()` no Windows
- `quartz()` no Macintosh

Para fechar a janela gráfica usamos:

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

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

- `postscript()`

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

Existem ainda outros dispositivos que podem ser específicos de uma determinada plataforma (sistema operacional). Cada uma destas 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 várias 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 você 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()
```

34 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. Entretanto 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.

34.1 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      _
arch           x86_64-pc-linux-gnu
os            linux-gnu
system        x86_64, linux-gnu
status
major         4
minor         6.1
year          2026
month         06
day           24
svn rev       90187
language      R
version.string R version 4.6.1 (2026-06-24)
nickname      Happy Hop
```

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

- `getRversion()` retorna string com a versão do R.
- `.Platform` retorna lista 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 sobre o sistema e usuário.
- `.Machine` detalhes sobre aritmética usada, tal como menor 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 também os recursos (pacotes) disponíveis. As saídas das funções mencionadas podem ser usadas quando informando/reportando problemas encontrados em aplicações e/ou quando escrevendo funções que possuam funcionalidades e opções que dependam destas informações.

```
> sessionInfo()
```

```

R version 4.6.1 (2026-06-24)
Platform: x86_64-pc-linux-gnu
Running under: Ubuntu 24.04.4 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.12.0
LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.12.0  LAPACK version 3.12.0

locale:
 [1] LC_CTYPE=pt_BR.UTF-8      LC_NUMERIC=C               LC_TIME=pt_BR.UTF-8
 [4] LC_COLLATE=pt_BR.UTF-8    LC_MONETARY=pt_BR.UTF-8   LC_MESSAGES=pt_BR.UTF-8
 [7] LC_PAPER=pt_BR.UTF-8      LC_NAME=C                  LC_ADDRESS=C
[10] LC_TELEPHONE=C            LC_MEASUREMENT=pt_BR.UTF-8 LC_IDENTIFICATION=C

time zone: America/Sao_Paulo
tzcode source: system (glibc)

attached base packages:
[1] stats      graphics  grDevices  utils      datasets  methods    base

other attached packages:
[1] knitr_1.51

loaded via a namespace (and not attached):
[1] compiler_4.6.1 cli_3.6.6      tools_4.6.1    ote1_0.2.0     xfun_0.57
[6] rlang_1.2.0    evaluate_1.0.5

```

34.2 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/mateRial/aulasR/Rnw"
> wdir <- getwd()
> wdir
[1] "/home/paulojus/mateRial/aulasR/Rnw"
> R.home()
[1] "/usr/lib/R"
> setwd(R.home())
> getwd()
[1] "/usr/lib/R"
> setwd(path.expand("~/"))
> 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 e arquivo temporários.

```
> tempdir()
[1] "/tmp/RtmpHPhk0i"
> tempfile()
[1] "/tmp/RtmpHPhk0i/file74aef236f8e4c"
```

34.3 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/mateRial/aulasR/Rnw"
> dir("../")
[1] "aulasR.aux"          "aulasR.bbl"          "aulasR.blg"
[4] "aulasR.log"          "aulasR.out"          "aulasR.pdf"
[7] "aulasR.tex"          "aulasR.tex~"         "codigoR"
[10] "compila_rnw.sh"      "dados"               "ExSweave"
[13] "figuras"             "Rembrapa.pdf"        "Rnw"
[16] "Rplots.pdf"          "setup_knitr.R"       "tex"
[19] "verifica_pacotes.R"
> setwd(R.home())
> dir()
[1] "bin"          "COPYING"      "etc"          "lib"          "library"
[6] "modules"      "site-library" "SVN-REVISION"
> args(dir)
function (path = ".", pattern = NULL, all.files = FALSE, full.names = FALSE,
         recursive = FALSE, ignore.case = FALSE, include.dirs = FALSE,
         no.. = FALSE, fixed = FALSE)
NULL
> file.info("bin")
      size isdir mode          mtime          ctime          atime uid
bin 4096  TRUE  755 2026-08-31 10:51:23 2026-08-31 10:51:23 2026-09-08 21:31:50  0
      gid uname grname
bin   0  root  root
> file.info("bin")$isdir
[1] TRUE
> dir(path="bin")
[1] "BATCH"          "build"          "check"          "COMPILE"        "config"
[6] "exec"           "INSTALL"        "javareconf"     "libtool"        "LINK"
[11] "mkinstalldirs" "pager"          "R"              "Rcmd"           "Rd2pdf"
[16] "Rdconv"         "Rdiff"          "REMOVE"         "Rprof"          "Rscript"
[21] "rtags"          "SHLIB"          "Stangle"        "Sweave"
> dir(pattern="COPY")
[1] "COPYING"
> dir(path="doc")
character(0)
> dir(path="doc", full=TRUE)
character(0)
```

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

```

> setwd(path.expand("~/"))
> file.exists("foo.txt")
[1] TRUE
> file.create("foo.txt")
[1] TRUE
> file.exists("foo.txt")
[1] TRUE
> file.rename("foo.txt", "ap.txt")
[1] TRUE
> file.exists("foo.txt")
[1] FALSE
> file.exists(c("foo.txt", "ap.txt"))
[1] FALSE TRUE
> file.copy("ap.txt", "foo.txt")
[1] TRUE
> file.exists(c("foo.txt", "ap.txt"))
[1] TRUE TRUE
> file.remove("ap.txt")
[1] TRUE
> file.exists(c("foo.txt", "ap.txt"))
[1] TRUE 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] "/home/paulojus/mateRial/aulasR/Rnw"
> dir.create("~/meu.dir")
> file.copy("foo.txt", "~/meu.dir")
Warning in file.copy("foo.txt", "~/meu.dir"): problem copying ./foo.txt to /home/paulojus/meu.dir/foo.t
[1] FALSE
> dir("~/meu.dir")
character(0)
> unlink("~/meu.dir", recursive=TRUE)
> setwd(wdir)      # retorna ao diretório de trabalho original

```

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 diretamente 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` passa 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 existentes 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")

```

35 Usando o Sweave

35.1 O que é e por que adotar o Sweave

O **Sweave** é uma funcionalidade do R implementada por algumas funções do pacote **utils** (no pacote **tools** em versões mais antigas do R) que permite a edição ágil de documentos combinando o L^AT_EX e o R.

Usando o **Sweave** o usuário pode ter comandos, saídas computacionais e/ou gráficos incluídos automaticamente no texto, sem a necessidade de fazer tal inclusão manualmente e passo a passo. Este mecanismo também permite que o texto seja ágil e automaticamente atualizado para qualquer mudança ou inclusão de dados e/ou nas análises, acelerando *muito* o processo de edição de textos.

Uma outra vantagem relevante é a de que todo código usado para análise fica no arquivo texto (fonte) preservando a memória dos procedimentos usados e possibilitando a reprodução ou modificação das análises facilmente e a qualquer tempo.

Com estes aspectos o **Sweave** torna-se uma ferramenta adequada para o que se chama de *pesquisa reproduzível* (*reproducible research*) uma vez que permite que o código das análises seja disponibilizado junto e integrado ao texto.

As possibilidades de uso do **Sweave** são diversas e podemos citar como exemplos a confecção de relatórios, provas, listas de exercícios, textos técnicos, relatórios de análises, artigos científicos e livros. Outro exemplo ainda é este material sobre o R que foi todo originalmente editado em formato *Sweave*.

35.2 Usando o Sweave

Os passos básicos para uso do **Sweave** são:

1. Editar o arquivo `.Rnw`. Neste documento vamos supor que seu arquivo se chama `foo.Rnw`
2. Iniciar o R
3. rodar a função **Sweave()** no seu documento com um comando do tipo:

```
> Sweave("foo.Rnw")
```

Ao final destes passos, a função **Sweave()** irá imprimir uma mensagem na tela como a seguir dizendo que o documento `foo.tex` foi gerado.

```
You can now run LaTeX on 'foo.tex'
```

Caso outra mensagem que não esta apareça na tela algum problema deve ter ocorrido com o código R em seu documento. Leia a mensagem, identifique e corrija o erro e processe novamente com **Sweave()**.

4. Compile e visualize o documento L^AT_EX de forma usual.

35.3 Outras informações úteis para uso do Sweave

- O **Sweave** tem algumas dependências de outros recursos no L^AT_EX. No caso do LINUX certifique-se que você tem os seguintes pacotes instalados: **tetex-bin** e **tetex-extra**. No Windows a instalação do MiKTeX deve prover as ferramentas necessárias.

- A [página oficial do Sweave](#) contém o manual, artigos, exemplos, FAQ ("Frequently asked questions") e informações adicionais.
- Versões mais recentes do R incorporaram o comando **Sweave** de tal forma que é possível processar o documento `.Rnw` para gerar o `.tex` diretamente da linha de comando do LINUX sem a necessidade de iniciar o R. Para isto basta digitar o comando a seguir na linha de comando do LINUX (ou o comando análogo em outros sistemas operacionais).

```
R CMD Sweave foo.Rnw
```

- O mecanismo descrito anteriormente substitui uma versão anterior que recomendava o uso do script `Sweave.sh` que também permitia rodar o **Sweave** no seu documento `.Rnw` diretamente da linha de comando do LINUX, sem a necessidade de iniciar o R, bastando digitar:

```
Sweave.sh foo.Rnw
```

Note que para o comando acima funcionar o "script" `Sweave.sh` deve estar como arquivo executável e disponível no seu `PATH`.

Alternativamente você 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 a 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 código R do arquivo.
- Alguns editores facilitam o uso do **Sweave** (podem haver outros além dos mencionados a seguir): Isto é muito útil na preparação dos documentos pois permite também que o código em R dentro dos *chunks* seja enviado de forma ágil para processamento no R.
 - Os documentos formato **Sweave** (extensões `.Rnw`, `.Snw`, etc) são reconhecidos pelos editores **Emacs** ou **Xemacs** desde que o pacote **ESS** do Emacs esteja instalado.
 - O **Tinn-R** é outro editor que reconhece o formato de documentos do **Sweave**.
- O pacote **odfWeave** do R oferece funcionalidade análoga para edição de documentos utilizando o editor **LibreOffice** (sucessor do BrOffice/OpenOffice)
- O **Sweave** foi concebido por **Friedrich Leisch** da Universidade Técnica de Viena e membro do *R Core Team*.

35.4 Controlando o tamanho dos gráficos

Há duas formas de controlar o tamanho de gráficos no Sweave, a primeira *via R* especificando o tamanho do gráfico a ser gerado, e segunda usando comandos do L^AT_EX para definir o tamanho dos gráficos. Os dois mecanismos podem ser usados em conjunto.

O primeiro mecanismo usa a definição do tamanho da imagem em argumentos do *chunk* como `<<fig=true, width=L, height=A>>=`, onde *L* e *A* são números em unidades de polegadas.

No segundo mecanismo usa-se recursos do pacote `graphicx` do L^AT_EX com o comando `\setkeys{Gin}{width=L}`, onde *L* é uma unidade de medida do L^AT_EX tal como `0.8\textwidth` (80% da largura do texto) ou `5cm`. Pode-se definir também a altura do gráfico, caso contrário a figura é escalonada automaticamente mantendo a proporção entre largura e altura.

35.5 Exemplos de arquivos em Sweave

1. Um exemplo de um arquivo `.Rnw`.
2. Arquivo com o conteúdo da seção sobre distribuições de probabilidades deste material. Para compilar este exemplo você 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.

35.6 Links

- [Página do Sweave](#) mantida por seu autor
- [Texto sobre o Sweave](#) por Fritz Leisch, o criador do Sweave
- [Um tutorial em espanhol](#)
- [Página sobre Sweave](#) mantida por Fernando Ferraz

36 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 disponíveis 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 destes 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 você pode ver se ele já está instalado/disponível. Para isto inicie o R e digite o comando:

```
> (require(NOME_DO_PACOTE))
```

Se ele retornar **TRUE** é porque o pacote já está instalado/disponível e você não precisa instalar. Se retornar **FALSE** siga os passos a seguir.

A instalação e uso dos pacotes vai depender do seu sistema operacional e os privilégios que você tem no seu sistema. Nas explicações a seguir assume-se que você 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 você 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 todos 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 você 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 você pode, alternativamente, instalar usando o menu do R selecionando a opção **PACKAGES - INSTALL FROM CRAN**.

36.1 Pacotes não-oficiais

Além dos pacotes contribuídos existem diversos pacotes *não-oficiais* disponíveis 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 CMD INSTALL PACOTE.tar.gz
```

ou

```
R CMD INSTALL -l ~/Rpacks PACOTE.tar.gz
```

- **Windows:** No menu do R use a opção **PACKAGES - INSTALL FROM LOCAL .ZIP FILE**

37 Construindo pacotes

Os passos básicos para construção 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 cuidado 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 requerida para pacotes. Por exemplo, se o seu pacote 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 você 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". Você não precisa fazer nada neste diretório.
- O subdiretório `man` que contém a documentação 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. Você 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 você 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 subdiretório `R` com o seguinte conteúdo

```
.onLoad <- function(libname, pkgname)
{
  library.dynam("Embrapa", package = pkgname, lib.loc = libname)
}
```

NOTA: o gancho `.First.lib` usado em versões antigas do R para esta finalidade está obsoleto e não é mais chamado em pacotes com arquivo `NAMESPACE` (como os gerados por `package.skeleton()`); use `.onLoad()` como acima.

5. Para testar o seu pacote você pode usar na linha de comando:

```
R CMD check meupack
```

6. Para montar o arquivo fonte `.tar.gz` de distribuição do 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, usando como exemplo a própria construção de um pacote **Embrapa** reunindo as funções e conjuntos de dados utilizados nas aulas.

Os 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*.

Sobre este texto

Este material é produzido e disponibilizado usando exclusivamente recursos de **SOFTWARE LIVRE**.

O texto foi originalmente editado em **L^AT_EX** e combinado com código e saídas do **R**. Em uma primeira revisão passou-se a usar o recurso do **Sweave** para proporcionar reprodutibilidade e tornar o documento dinâmico.

A versão para WEB foi obtida convertendo o documento **L^AT_EX** para 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 navegadores que não possuam suporte a **MATHML**.

Se seu navegador não suporta MATHML (como por exemplo o Internet Explorer) você pode habilitar este suporte instalando o **MathPlayer**.

Após um longo período sem revisões em Setembro de 2026 o texto foi convertido para o formato **knitr**, que aplica as capacidades do **Sweave**. Na ocasião uma revisão de ortografia, gramática, códigos e formatação foi conduzida utilizando **Claude IA**. Na ocasião a versão antiga em **Sweave** foi “congelada” e preservada para registro.

Todo o material foi produzido e disponibilizado em ambiente **Debian-Linux** e **Ubuntu-Linux**. A página WEB é disponibilizada em um servidor **APACHE** rodando em um Debian-Linux.